Sushant Sundaresh

EE119C, Caltech: Noise Canceling Headphones

Notes, July 27, 2015 – start of week 7.

**Efforts so far** For weeks 1-3 over the summer I worked on the assumption that the LMS feedback structure would succeed. Without really understanding the details of LMS, I screwed with various parameters and found failure at every turn – noise at the input, multiple frequencies both caused the filter coefficients to blow up, though slowly, and the output blew up quickly somehow as well.

**First Goal for today** I’d like to get a better feeling for the kinds of solutions that exist. For example, forget LMS for now. I will start with a random walk. Let’s ignore how we’re going to implement things at first. Consider a system sampled at 1/100 second intervals. Consider two frequencies added at input, at 1000 sample and 333 sample periods, with amplitudes equal at 1 unit. Suppose these are delayed 10 samples at the ‘ear input.’ That’s about 1 ms delay – about 10x what’s actually expected. Suppose for the moment that the output of our filter suffers no equivalent delay – no other distortion either. A best-case of sorts.

Let’s just take a look at what kinds of solutions are sufficient to match that 10 sample delay for those two frequencies, as we vary the frequencies. Let’s use a 32-mer FIR filter, and update our filter as follows: we create a new filter by adding a random 32-mer with elements in -1,1 to our best filter so far. For 32 samples we keep this new filter, adding up the errors between the output and input at the ear. We then update the filter if its error is lower than our best error.

* I am interested in the sizes of the coefficients for the ‘best filters’ that result.
* I am interested in how few filter coefficients I can get away with as a function of how many frequencies I want to deal with (and how widely spaced they are, or how small my sample frequency is relative to the target frequencies).
* I am interested in how the frequency of the input affects how long I need to track error for the summed-error per filter to be meaningful. One filter length’s worth of samples? Two? One-half?
* I am interested in how the error associated with a particular filter changes over time (maybe it fits better at certain times – how likely am I to miss a godo solution just because I checked it at the wrong time?).
* I am interested in generally, how long it takes to converge to good solutions this way, and how consistent this convergence time is.

I’ve tested this, see, and I know this methods can work quite well, if slowly. This is all to get a feeling for the kinds of solutions LMS might converge to, why it’s helpful to use the input itself to guide the search in filter space, etc.

**Second Goal for Today** I found yesterday that in plain LMS vs the feedback structure I’ve played with, there’s a difference in the error calculations (a delay, in my simple examples) that could explain why things blow up. I want to play with this a bit more.

**Third Goal for Today** I want a schedule for when I’m going to get this done. At first, I want to know what I will get done this week. Without a plan, I won’t prioritize this.

**First Goal**

What I’m noticing is, for the 1000/333 sample frequencies, with a filter of length 32, that is, about 1/10 the # samples for even one input period, if the filters update in -1,1, you might not notice anything has converged, because those are large steps in filter space. But, an optimal solution is reached, because if I track that, and at some point just stop updating, then usually I get a trace that’s really close to the input at the ear. So we are hitting it, but the perturbations to the filter are so large that if I just let the filter keep updating it’ll never sound like I’ve found a good solution. So some level of adaption in filter steps would be helpful.

I do also notice that filters with coefficients in -1,1 are more than sufficient. I don’t know whether filters with integer values in, say, 0-255 would also work.

If I make the filter update steps in -.01,0.01, or 0.1, we rarely if ever hit an optimal filter. So large steps are important at first when we have no idea which way to go, but they hurt us later.

Right now it is almost seems as though the system does terribly until the input gets a bit lower, close to 0, then it traces and figures out a decent sol. That doesn’t make a lot of sense to me, but I can test it just by limiting the time at which the system stops updating the filter.

Here’s an interesting observation: if I have no DC offset to my input, and I start the filter at 0, and the input starts at 0, then that tends to be the ‘best filter’ for quite a while, and that’s where we keep reverting to as we try and step. In this case, if I limit the number of steps till I stop updating, I rarely get a good solution for future input prediction. However, with a DC offset, this initial point has pretty terrible error (a 0 – filter totally fails), so the first few steps become the ‘best filter’ and then, even at the same time-cutoff for updating the filter, we get good solutions most of the time. So even a random seed away from zero to build away from, is a good idea. Maybe this has something to do with the input amplitudes.. or maybe solutions are dense and you just need something to work with.

For example, with a DC offset of 0.2, with inputs at 1000/333 sample periods at amplitude 1 each, summed, and a 32-mer filter updated at -1,1 every 32 samples, I find that even 1000 samples (~30 filter update intervals) is enough to find a decent solution – most of the time if I stop updating after that, the optimal solution does a damn fine job of matching future input. Ok, maybe “most” was an overstatement. Maybe 50% of the time, and not perfectly, you’d still probably hear it, but it’s visually ~20-fold attenuated.

The errors in these cases, by the way, range about 1E-6 to 50, summed over the 30 samples. That’s a hell of a range, and we’d probably saturate on both ends if in 8 or 9 bit fixed point arithmetic.

Here’s something to try: change the DC offset to 0 again, and see how often we get decent solutions now. Barely any at all – definitely much, much less than with a DC offset. I guess the initial alignment between the two just wins out as all steps from 0 are blind, and even if not terrible, lose out to the initial near-perfect match between small sinusoids (near 0) and 0 itself. So we never really.. update, and we can’t adapt. It’s like being chained, rather than exploring.

Also, seems to me that I’ve tested basic LMS and it doesn’t work so well with more than a few frequencies. Even two did pretty poorly. I remember fast-block lms in Simulink working well with a real signal, but that was with zero error/desired/input delay. Not the same issue. Also fast-block is taking errors over a time window equal to the filter width. So it’s also averaging error, which we know should help. So for now, screw LMS in feedback – the delay is unknown, or only known within some error, so it’s not quite DLMS, therefore we don’t have the exact LMS structure where they derived the optimal guess that is basic LMS.

First, let’s see if we can get this to work. It’ll be a start, and you have to make decisions and move forward. you’ve seen this work time after time in View/Solar, View/Barcode, Caltech/EE113, you need to try, iterate.

Next steps could include trying to derive the LMS equations, to include the effect of an unknown (or intentionally characterized) output filter.

08-02-15

First, Palash’s idea: directly use the error as feedback. That is, try a single freq, and try just always feeding back the negative.

So, take an input of one frequency, and consider a starting guess of 0, consider the error, and output the negative. Then add the two with some delay on the guess, and continue. What happens?

Nothing good – we’ve actually tried something like this before. Focus on the other one for now.

One thing I want to try right now: let’s see what happens if we use the error (without delayed input) as the random step, and try it again with a delayed filter output too!

Yeah bad idea for error as random step. Also haven’t tried delayed output yet but not my major concern.

Trying delayed filter output posed no issue. Fixed bugs in code – now all time-based in seconds. Noticed with a step size exponentially decaying from 0.1, in about 10 s with ms sampling found could get single tone down to 0.1% error. Now, LMS itself was able to handle that much faster, but this one doesn’t blow up at least. Tried listening to 1% vs. 100% amplitude in Audacity – it’s almost inaudible. That is, if this holds for multiple frequencies, with small amounts of noise, and with my own RNG & arithmetic limiting error resolution & overall accuracy, then we’re good to go. So those are the next things to check. For now let the user keep pressing the button when they want to update it again – I need to hear what this sounds like.

I think now would be a good time to try more organized tests. Let’s try and understand chances of success and minimum timescales.

**Parameters**

**Tests**