ECEn 487 Digital Signal Processing Laboratory

Lab 2, 2023
Least Mean Squares (LMS) Adaptive Filters

Due Dates

This is a two-week lab. The tasks take varying amounts of time, so work to make early progress. Submission via Learning Suit of a copy of your lab book for this lab is due by 2:00 p.m. on Monday February 27.

Learning Objectives

The purpose of this lab is to learn principles of adaptive filtering by deriving and applying the LMS algorithm. Applications of system identification, adaptive noise cancelation, and channel equalization will be studied.

Reading Assignment and MATLAB Set Up

1. This handout: study it thoroughly.

2. MATLAB help documentation for functions `audioplayer, freqz, filter, and randn`.

3. This lab is executed entirely in MATLAB, not on the STM32F7 board. If you must work from home due to illness or university excused absence and you do not have access to MATLAB on your personal computer, please contact Professor Jeffs via email for instructions on how to remotely log into MATLAB.

Introduction

In Lab 2 you used the windowed FIR filter design technique to create a bandpass filter to reject undesired interferers. In future labs and lectures you will learn several additional methods to develop deterministic designs for both FIR and IIR filters to satisfy a desired frequency response specification. However, in this lab you will study adaptive filter architectures which "magically" design themselves to satisfy a mathematical optimality criterion. These are data – driven algorithms which adapt the filter characteristics according to the observed data. This is an advanced signal processing topic taught in graduate classes like ECEN 671, 670, or 777. There are many sophisticated algorithms and methods, but the basic concepts of adaptive filters are easily derived, demonstrated, and understood using the very first such system, the famous yet simple "LMS adaptive filter."

So how can a filter design itself? It seems impossible. The key is that in all cases we need to provide a reference, or desired signal, *d*[*n*], which the algorithm compares to its filter output. The impulse response is adjusted by an iterative algorithm until the filter output is driven as close as possible to matching the desired output. But, this begs the question, "if we must already have the desired signal, why do we need to design a filter to give us the same result?" The answer is that by clever choices of how to generate a desired reference signal and how we connect this and the observed signal of interest up to our adaptive filter, we can accomplish some very useful signal processing. As examples, in this lab we will consider how to use the LMS filter for system identification, noise cancelation, and system equalization.

First, we must derive the adaptive iterative algorithm. Consider the filter architecture in Fig. 1. The feedback path shown from the error output, *e*[*n*], is used to adjust the filter impulse response, *h*[*n*]. Though few details are given, this figure represents the algorithm we will derive for updating filter tap weights. The approach is that at every sample time we make incremental adjustments to *h*[*n*] until the average error power, $\frac{1}{N}\sum_{n=0}^{N-1}|e[n]|^2$, is driven as low as we can get it. To do this, *h*[*n*] must filter *x*[*n*] so that *y*[*n*] looks as much like *d*[*n*] as possible.

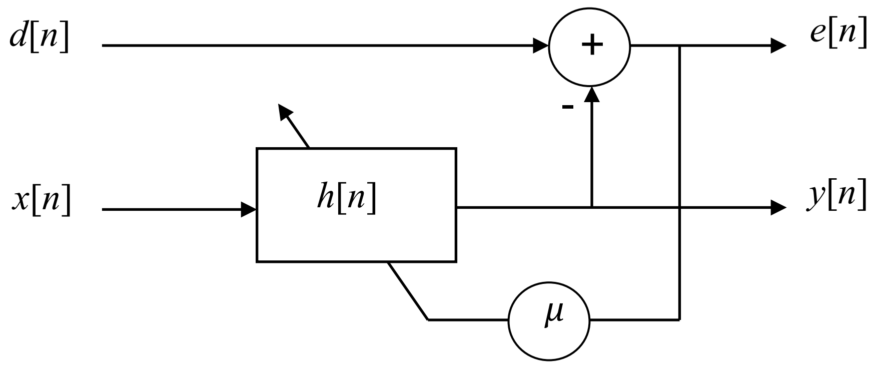

Figure 1. LMS adaptive filter architecture

The squared error signal can be expressed as

$$\begin{aligned}|e[n]|^2 &= \left| d[n] - \sum_{k=0}^{M-1} h[k]x[n-k] \right|^2 \\ &= d^2[n] - 2d[n]\sum_{k=0}^{M-1} h[k]x[n-k] + \sum_{l=0}^{M-1}\sum_{k=0}^{M-1} h[k]h[l]x[n-k]x[n-l]\end{aligned} \tag{1}$$

where *M* is the order of FIR filter *h*[*n*]. To minimize this with respect to *h*[*m*] we will use the iterative method of steepest descent: at each time sample we take a small step in the direction of the negative gradient. This for example is the path that water takes flowing down any surface shape. We first calculate the gradient by computing the partial derivative of $|e[n]|^2$ from (1) with respect to each filter tap value in *h*[*m*]:

$$
\begin{aligned}
\frac{\partial}{\partial h[m]}\left|e[n]\right|^2 &= -2d[n]x[n-m] \;+\; 2\sum_{k=0}^{M-1} h[k]x[n-m]x[n-k] \\
&= -2x[n-m]\underbrace{\left(d[n]-\sum_{k=0}^{M-1} h[k]x[n-k]\right)}_{e[n]} \qquad (2) \\
&= -2x[n-m]e[n]
\end{aligned}
$$

So, moving in the negative gradient direction (2), the steepest descent algorithm is simply:

$$h^{(n+1)}[m] = h^{(n)}[m] + \mu x[n-m]e[n], \quad e[n] = d[n] - y[n] \text{ and } y[n] = \sum_{k=0}^{M-1} h[k]x[n-k] \qquad (3)$$

where μ is a small constant which controls step size, and $h^{(n)}[m]$ is the estimated value at time sample *n* for the *m*th tap of the optimal filter impulse response. The factor "2" from (2) is absorbed into μ. Equation (3) is the LMS filter update equation which you will implement as a MATLAB function to build you adaptive filter.

You may have noticed that it appears as if we forgot about averaging the squared error when we were computing gradients. In fact, we did not forget. By using a small μ value and updating the estimate for every time sample *n*, we get implicit time averaging. Each update is a poor estimate of the true gradient vector direction, but we only go a small distance. Over time, these small steps average out to send us in the right direction towards the optimal *h*[*n*].

Experiment 1: ***Write an LMS adaptive filter code***

Procedure

1. Write a MATLAB function to run a block mode (i.e. assuming all data is in memory) LMS adaptive filter to match the block diagram of Figure 1.

   The function call syntax should be:

```
function    [y, e, h] = lms(x, d, mu, h_init)
%   LMS adaptive filter algorithm
%
%      x:  Input data vector
%      d:  reference, or desired channel input vector
%          x and d must be same length
%     mu:  adaptation step size constant.
% h_init: intial value for h.  Used to chain calls and set filer length
%      y:  Filtered output data
%      e:  error sequence
%      h:  estimated optimal least mean square filter tap values.
```

2. For efficiency and speed, at a given time sample *n* in your "for loop," the filter convolution sum should be computed with a vector inner product of the current *h* and an equal length window of data extracted from *x*. You should not have a two-layer nested loop like you did in your Lab 2 C code, just a single loop over *n* with the convolution handled by a vector product.

3. To make startup conditions easier, it is ok to skip the first *M* samples in the data set and start your loop with $n = M$. That way you won't have to deal with negative indices for your data vectors.

4. For pass off, have the TA review your code to see if there are any obvious errors before running it.

***Experiment 2,* System Identification:**

Suppose you have a black box system which you cannot "look into" but you need to know its impulse response. Figure 2 shows how you can do this with the LMS filter and a probe signal, $p[n]$. This does not need to be an impulse, but any signal which spans at least the frequency range for which the system has some output response. At convergence, $h[n]$ will mirror the impulse response of the black box.

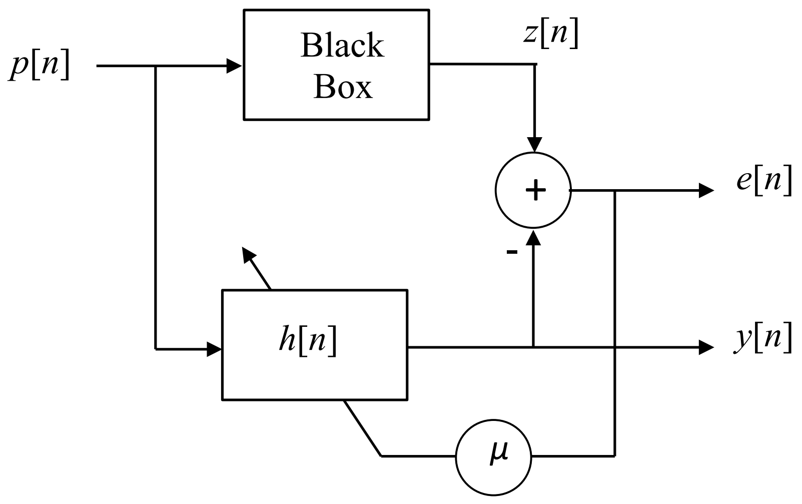

Figure 2. LMS configuration system ID.

Procedure

1. Use your LMS filter function to implement a system identifying adaptive filter as shown in Figure 2.

2. Download the file `sysIDdata.mat` from Learning Suite. It contains several data vectors sampled at 8 kHz, 10 seconds long. Using `audioplayer`, listen to the signals in `p` and `z`. Comment in your lab book on what you hear.

3. Use the `p` ($p[n]$) and `z` ($z[n]$) vectors as your signals for the probe signal, and unknown system outputs respectively. For `h_init`, start with a vector of zeros. The length of this vector sets the order of your adaptive filter. For this data set, a filter length of 120 is adequate. Adjust your `mu` value up and down manually until you see that the error signal $e[n]$ converges nicely to a small value before your reach the middle of the data set. (hint: if μ is too large, the output will "blow up" and go unstable. If μ is too small, convergence is very slow, but the result is more accurate. It is OK to use the same data (`p` and `z`) and repeat several times to achieve better convergence. You should use the `h` achieved at the end of each iteration as the `h_init` input for the next repetition.)

4. Compare the final `h` vector with the downloaded `h_true` vector by plotting them on the same axis. `h_true` is the actual black box system impulse response. Also, make linear-scale freqz plots of the frequency responses for each of these systems. To do this use the calling syntax: `[H, W] = freqz(h,1);  plot(W, abs(H)).` `freqz(h,1)` alone plots on a dB scale. Comment on how well your adaptive filter did in identifying the frequency response. Also, suggest a realistic scenario where you might not have done as well.
5. Record all your observations and experiment details in your lab book, and assess the effectiveness of this adaptive filter. Note what happens as you make mu larger and smaller than your best value.

6. Have the TA check your results and lab book entries and sign it off.

Experiment 3, Noise Cancelation:

If you observe $z[n]$ which includes a desired signal $s[n]$ corrupted by additive noise, and you can get a second sample of the noise, $\eta[n]$, which does not have the signal of interest present, it is possible to use the adaptive filter to remove noise. This is true even if this noise-only sample and the noise seen in the desired signal are not identical, but are related to each other by some unknown transfer function $h_\eta[n]$. Figure 3 illustrates how this is done.

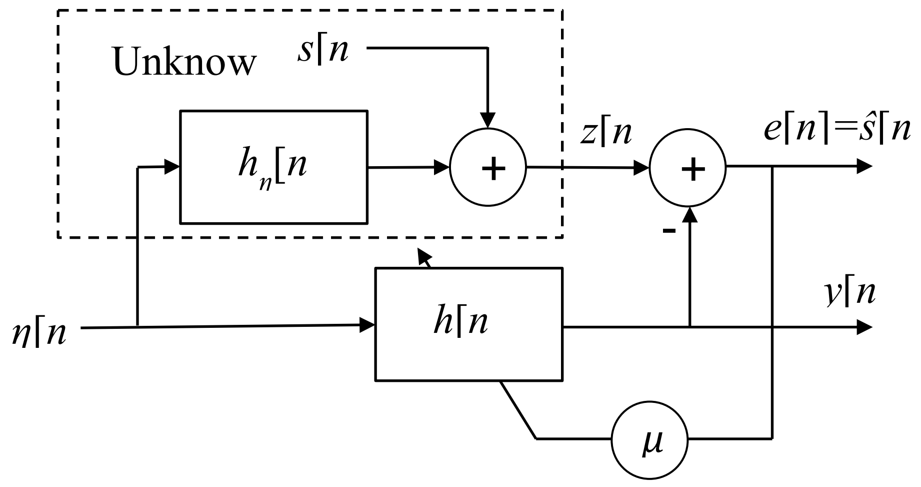

Figure 3. LMS configuration for noise cancelation.

The adaptive filter tries to reduce power in $e[n]$, but since it does not see the desired signal $s[n]$ at its input, the best it can do is filter the noise to try and match the noise component in $z[n]$ so it will be subtracted out from $e[n]$, leaving just $s[n]$.

Procedure

1. Use your LMS filter function to implement a noise cancelation adaptive filter as shown in Figure 3.

2. Download the file `noiseCancelData.mat` from Learning Suite. It contains several data vectors sampled at 8 kHz, 10 seconds long. Using `audioplayer`, listen to the signals in `eta, z, eta2, and z2`. Comment in your lab book on what you hear.

3. Use the `eta` ($\eta[n]$) and `z` ($z[n]$) vectors as your signals for noise reference, and signal-plus-noise respectively. For `h_init,` start with a vector of zeros. The length of this vector sets the order of your adaptive filter. For this data set a filter length of 25 is adequate. Adjust your `mu` value up and down manually until you see that the error signal $e[n]$ converges nicely to a small value before your reach the middle of the data set.

4. Confirm that you are able to hear a periodic signal coming out of the noise by the end of your output sequence. Use audioplayer to play the signal through headphones or speakers and also plot your cleaned up signal. Describe in your lab book what type of signal this is, its period, and your best setting for mu. Record all your observations and experiment details in your lab book, and assess the effectiveness of this adaptive filter. Note what happens as you make mu a larger and smaller than your best value.

5. Plot your final `h` and compare it with the downloaded `h_true`. How well did your adaptive filter do finding the actual $h_\eta[n]$?

6. Repeat steps 3 and 4 using the `eta2` and `z2` vectors. The signal `z2` is noise plus speech. In order to hear the entire message, run you filter twice. The second time, use your previous h output as the h_init input. Record the gist of the message in your log book.

7. For pass off, have the TA check your results and lab book entries and sign it off.

Experiment 4, ***Adaptive Equalization:***

It is often desirable to remove the effect of some system by running the signal through an inverse filter, or equalizing filter. For example, in digital wireless communications the unknown multipath propagation channel, $g[n]$, causes frequency selective fading (attenuation at some frequencies) so the communications channel must be equalized to achieve reliable symbol decoding. The question is "how do I solve for the inverse filter?" Figure 4 shows how this can be done with the LMS adaptive filter and a training data signal, $t[n]$, which is available at both ends of the communications link.

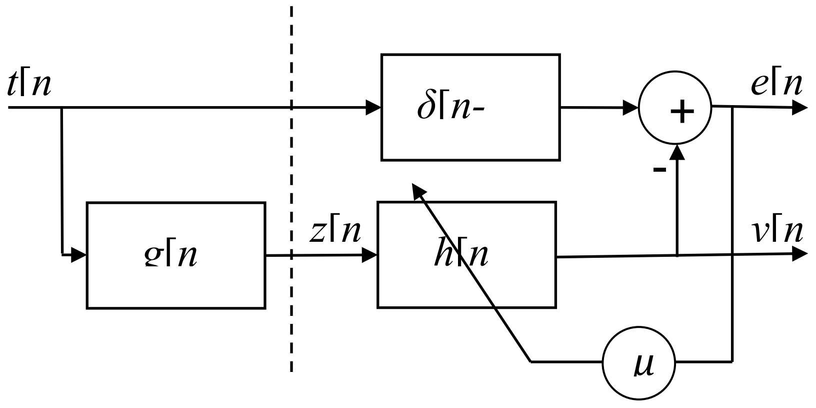

Figure 4. LMS configuration for adaptive equalization

At convergence $h[n]$ approximately removes the effect of $g[n]$ such that $H(z) \approx 1/G(z)$. The D sample delay is needed to time-align $t[n]$ with $y[n]$ since $y[n]$ has undergone the delays introduced by two FIR filters. D should be approximately equal to the total effective time delay through the cascade of the two filters $g[n]$ and $h[n]$. If $g[n]$ is a symmetric impulse response, its time delay is $M_g/2$, where M_g is the filter order of unknown $g[n]$. You know the length, M_h, of $h[n]$ since you specify that. So in practice, initially set $D = (\hat{M}_g + M_h)/2$ where $\hat{M}_g$ is your best guess of the order of unknown FIR filter $g[n]$. You may have to adjust D for best performance, but start with the assumption of $\hat{M}_g = 70$.

Procedure

1. Use your LMS filter function to implement an adaptive equalizer as shown in Figure 4.

2. Download the file `EqualizerData.mat` from Learning Suite. It contains several data vectors sampled at 48 kHz, 10 seconds long. Using `audioplayer`, listen to the signals in `z, x2, and z2.` `z` is the observed system output of filter *g*[*n*] for training data (noise), `z2` is the output for music going through the corrupting filter, *g*[*n*], and `x2` is the original uncorrupted music clip. Comment in your lab book on what you hear.

3. Train your equalizer using `z` and `t.` For `h_init,` start with a vector of zeros. The length of this vector sets the order of your adaptive filter. For this data set a filter length of 80 is adequate. Adjust your `mu` value up and down manually until you see that the error signal *e*[*n*] converges nicely to a small value before your reach the middle of the data set.

4. Using your final `h` solution from the previous training set as a fixed equalizing impulse response, use the MATLAB `filter` command to correct `z2`. Using Audioplayer, listen to this equalized result and compare it to `z2` and the original `x2`. Comment on the comparisons. How well did it equalize and bring back lacking signal components? Comment in your lab book on ways you could see this technology being useful (ask me about my Yamaha home theater receiver system.)

5. Compare the final `h` vector with the downloaded `g_true` vector by plotting them on the same axis. `g_true` is the unknown system impulse response which you are trying to invert. Should the impulse response look the same? Why? Also, make linear-scale freqz plots of the frequency responses for each of these systems. To do this use the calling syntax: `[H, W] = freqz(h,1);  plot(W, abs(H)).` Comment on how well your adaptive filter did in inverting (equalizing) the frequency response.

6. Have the TA check your results and lab book entries to pass it off.