SSY130 – Applied Signal Processing
Project 2: Adaptive Noise Cancellation
Rev 1.1

T. McKelvey, J. Lock

October 22, 2019

1 Introduction

The purpose of this project is to learn how to implement and use an adaptive filter algorithm: the *least mean squares* algorithm (LMS). Your task is to implement and study properties of the LMS algorithm on the same platform that we used for Project 1A (i.e. the DSP-kit). The particular application which we are targeting here is *noise cancellation.*

Assume we are given two signals: one measured signal (which is the result of sending a desired signal contaminated with a disturbance signal through a channel), and a separate measurement of the source of the disturbance. By noise cancellation we mean removing the disturbance from the measured signal through clever use of a filter. Active noise canceling headsets, which today are available on the market, use adaptive filtering techniques similar to the LMS technique you will use in this project.

The project is divided into two parts:

Part A Implement the LMS algorithm and test it.

Part B Investigate the properties of the LMS filter applied to the active noise cancellation problem.

2 Noise cancellation

Consider the case when sound in a room is measured using a microphone. We assume the measured sound can be modeled as the sum of two signals. One signal is the audio we want to listen to, e.g. music, a speaker, or similar. The other signal originates from a disturbance source which also propagates through the room and is also picked up by the microphone. Hence, if $x(n)$ is the sound signal sampled by the audio system at the microphone location we can separate the measured signal into two terms,

$$x(n) = s(n) + v(n) \tag{1}$$

where $s(n)$ is the desired audio signal and $v(n)$ is the noise signal measured at the microphone. Sometimes (which we assume holds here) it is possible to measure the noise directly at its source independently of $s(n)$. We call this noise source signal $y(n)$. We assume the propagation from the noise source $y(n)$ to the microphone which records $x(n)$, *the noise path*, is a linear filter with impulse response h. This implies that

$$v(n) = \sum_{k=0}^{\infty} h\left(k\right) y(n-k). \tag{2}$$

Similarly, defining $s'\left(n\right)$ as the signal source and $h'\left(n\right)$ as the channel the signal passes through we are given that

$$s\left(n\right) = \sum_{k=0}^{\infty} h'\left(k\right) s'\left(n-k\right).$$

In this particular project, the disturbance and desired signals are sent through the same channel, i.e. $h = h'$. In the rest of the project, we will use h to refer to this physical channel.

One approximation of h is to view it as a finite length filter (FIR) with M filter coefficients, which we will denote as $\hat{h}$. As we assume that we can measure the noise signal $y\left(n\right)$ at its source but we do not know the desired signal one estimate of the signal at the microphone is

$$\begin{aligned}\hat{x}\left(n\right) &= 0 + \hat{v}\left(n\right) \\ &= \sum_{k=0}^{M} \hat{h}\left(n\right) y\left(n-k\right),\end{aligned}$$

i.e. we assume that the desired signal is not present in our estimate of the signal at the microphone.

Finally, this shows that we can (approximately) remove the noise from the microphone signal as

$$\begin{aligned}e(n) &= x(n) - \hat{x}\left(n\right) \qquad (3) \\ &= s\left(n\right) + v\left(n\right) - \hat{v}\left(n\right) \\ &= s\left(n\right) + \sum_{k=0}^{\infty} h\left(k\right) y\left(n-k\right) - \sum_{k=0}^{M-1} \hat{h}\left(n\right) y(n-k) \\ &= s\left(n\right) + \varepsilon\left(n\right),\end{aligned}$$

where $\varepsilon\left(n\right)$ collects all mismatch between h and $\hat{h}$. Of course, in the ideal where $\hat{h} = h$ (i.e. the physical channel has a finite length M and is fully known) then $\varepsilon\left(n\right) = 0$ and the *error signal* output $e(n)$ would then be the desired audio signal $s(n)$.

We denote $e(n)$ as the *error signal* because in the case where $s(n) = 0$, i.e. there is no music signal present, perfect channel knowledge (i.e. $\hat{h} = h$) gives $e(n) = 0$. If $\hat{h} \neq h$, usually $e(n) \neq 0$ (though in some cases we can still get zero error, which relates to some of the questions you will later look at).

The remaining question is then: how do we obtain $\hat{h}$, i.e. the estimated filter coefficients of the disturbance path. A closer look at equation (3) indicates that $e(n)$ is the error in the standard adaptive filter setting. If we assume that $s'(n)$ and $y(n)$ are statistically uncorrelated we can obtain the desired result by minimizing the variance of $e(n)$. We can use the LMS filter algorithm to both perform noise cancellation (simply by regarding $e(n)$ as our useful output), and generate $\hat{h}$ at the same time.

A table of the introduced terms is shown in Table 1 for reference.

Table 1: Quantity reference.

Term	Meaning
e	"Error signal", output from LMS filter
x	Measured microphone signal, noise and desired signal through physical channel
$\hat{x} = \hat{v}$	Estimated microphone signal, estimate of passing noise through physical channel
v	Measured microphone signal, noise signal component
s	Measured microphone signal, desired signal component
y	Noise signal at noise source
$h = h'$	Physical channel from signal/noise source to microphone
$\hat{h}$	Estimate of physical channel

2.1 Adaptive Filtering with LMS

Remember that filtering a signal with a FIR filter (i.e. convolution) at sample n with M coefficients can be described by

$$\hat{x}(n) = \hat{\mathbf{h}}^T(n-1)\mathbf{y}(n) \tag{4}$$

$$\hat{\mathbf{h}}(n) = \left[\hat{h}_{M-1}(n), \cdots, \hat{h}_0(n)\right]^T \tag{5}$$

$$\mathbf{y}(n) = [y(n-M+1), \ldots, y(n)]^T \tag{6}$$

where $\hat{\mathbf{h}}(n)$ is a vector of filter coefficients, and $\mathbf{y}(n)$ is a vector that contains the present and $M-1$ past input signals $y(n)$.

2.1.1 Using the dot-product for convolution

Note that in (4) we generate $\hat{x}(n)$ using the dot product operator, instead of the typical convolution operator. If we consider the difference between the dot product and convolution (here for sample $n = 0$) we get

$$y(0) = (h * x)(0) = \sum_k h(0-k)\, x(k) \tag{7}$$

$$y'(0) = (h \bullet x)(0) = \sum_k h(k)\, x(k)\,. \tag{8}$$

Clearly, the only difference between the convolution operator in (7) and dot-product operator in (8) is that h is accessed from last to first in (7), but from first to last in (8). By variable substitution we can always consider $n = 0$ to be the sample of interest, e.g. for some measured signal $\bar{y}(n)$ we "shift" all indices by a constant amount. In essence, we can use the dot product operator to perform convolution so long as we take the reversed order of h or x into account. This result is useful for us as we can now make use of pre-existing dot-product functions in the DSP-kit.

2.1.2 Finding the optimal filter estimate

The optimal estimate of $\hat{\mathbf{h}}(t)$ for a stationary application is defined as the minimizing arguments of the mean-squared error criterion:

$$\hat{\mathbf{h}}_{\text{opt}} = \arg\min_{\hat{\mathbf{h}}} \mathrm{E}[e(n)^2] \tag{9}$$

$$\arg\min_{\hat{\mathbf{h}}} \mathrm{E}[(x(n) - \hat{x}(n))^2] \tag{10}$$

$$\arg\min_{\hat{\mathbf{h}}} \mathrm{E}[\left(\mathbf{y} * \left(\mathbf{h} - \hat{\mathbf{h}}\right)\right)^2] \tag{11}$$

$$\arg\min_{\hat{\mathbf{h}}} \mathrm{E}[\left|\mathbf{Y}\left(\mathbf{H} - \hat{\mathbf{H}}\right)\right|^2] \tag{12}$$

where E is the expectation operator and $\mathbf{Y}, \mathbf{H}, \hat{\mathbf{H}}$ are the Fourier transforms[1] of $y, h, \hat{h}$. Notably, in (12) we can see that the deviation between $\mathbf{H}$ and $\hat{\mathbf{H}}$ is weighted by $\mathbf{Y}$.

An approximate solution to the criterion (9) is obtained by using the LMS algorithm [1]:

$$\hat{\mathbf{h}}(n) = \hat{\mathbf{h}}(n-1) + 2\mu\mathbf{y}(n)\underbrace{\left(x(n) - \hat{\mathbf{h}}^T(n-1)\mathbf{y}(n)\right)}_{e(n)} \tag{13}$$

where 2μ is the referred to as the *step length*. More details on adaptive filters is provided in the lectures and in Chapter 8 of the text book [1].

2.2 Project setup

To illustrate the active noise cancellation application we will use DSP-kit. The noise path will be represented by the propagation of sound between the right loudspeaker

[1] Here we are somewhat sloppy and simply assume that the Fourier transforms exist. For the disturbances y we will test in this project we are ensured that $\mathbf{Y}$ does exist.

and the on-board microphone on the DSP-kit. The DSP-kit transmit the sum of the music signal s_0 and the noise signal y on the right loudspeaker. The microphone will pick up the result of sending this through the physical channel (mostly air), which we call x. The improved signal e, produced by the algorithm, is sent to the left output channel on the board (see Figure 1).

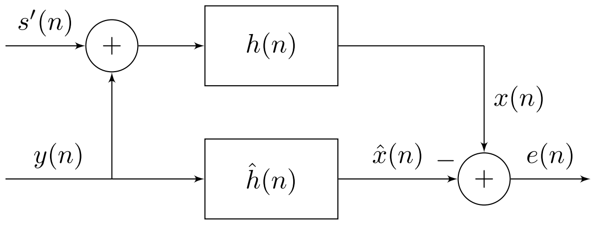


Figure 1: Signal diagram of the noise cancellation setup, see Table 1 for the meaning of each signal.

An image of a typical real-world setup is shown in Figure 2. The right speaker is placed fairly close to and pointed towards the DSP-kit. The left speaker (which has only a single cable) is kept as far away from the DSP-kit as reasonably possible. The purpose of this is to minimize the coupling (shown in Figure 2 as h^*) between the left speaker (which outputs the error signal) and the microphone. Any coupling can be viewed as an additional disturbance signal that *is* correlated with the disturbance transmitted by the right speaker, which will negatively affect the LMS filter's functionality.

Generally, it is a good idea to adjust the volume using the serial monitor and hold the left speaker directly against one group members ear whenever $\mu \neq 0$ (you can also place it face-down on the table). This minimizes the received parasitic signal $e * h^*$; e is reduced by decreasing the amplitude of the right speaker and h' is reduced by increasing the physical distance to the DSP-kit.

When decreasing the length of the filter (i.e. using fewer taps) during run-time the last element will simply be removed while the remainder will be left unchanged. For example, if changing from 5 to 4 elements, the filter coefficients will change as

$$[h_1, h_2, h_3, h_4, h_5] \to [h_1, h_2, h_3, h_4]\,.$$

Similarly, when increasing the length of the filter (ie. using more taps) an additional element will be added with numerical value 0, while the remainder will be left unchanged. E.g. if changing from 4 to 5 elements, the filter coefficients will change as

$$[h_1, h_2, h_3, h_4] \to [h_1, h_2, h_3, h_4, 0]\,.$$

2.2.1 Noise sources

There are two noise sources available for testing:

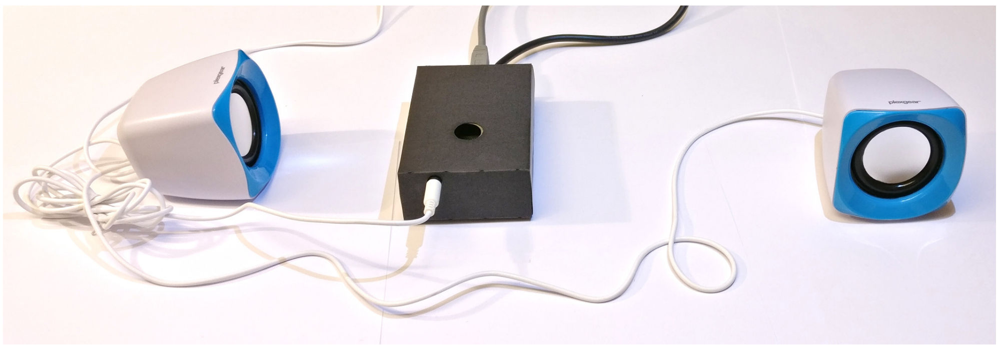


Figure 2: Image of typical real-world setup, with desired channel h and parasitic channel h^*.

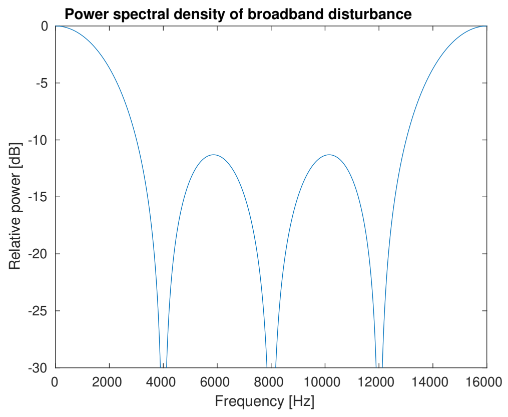


Figure 3: Broadband disturbance shows power primarily in range $[0, f_s/8]$.

Sinusoidal noise This noise source is simply generated as $y[k] = \sin(2\pi f_0 k/f_s)$ where f_0 is set to `LAB_LMS_SINE_TONE_HZ` (in `lab_lms.h`, with a default value of 440 Hz), and f_s is the system sample-rate.

Broadband noise This noise source is generated by up-sampling a Gaussian process sampled at 1/4 the system sample rate, up-sampled using zero-order-hold. This implies a vector of samples will always be of form $y = [x_1, x_1, x_1, x_1, x_2, x_2, x_2, x_2, x_3, x_3, x_3, x_3, \ldots]$ where x_n is randomly sampled from the $N(0,1)$ distribution. Figure 3 illustrates the normalized power spectral density of the up-sampled noise, i.e. $E|Y_{BB}(f)|$, showing that the majority of the signal power is located in the range $[0, f_s/8] = [0, 2000]$ Hz.

2.3 Project Task: Part A code development and testing

In the first part of this project, you will implement an LMS adaptive filtering function in the C language. The file `lab_lms.c` is located in the `src` project folder. In this source file you will find a skeleton of the `my_lms(..)` function. It is your task to complete the code such that the function performs the LMS algorithm on a block of data. See comments in the source code for details about how this should be implemented (and don't forget to uncomment the `SYSMODE_LMS` definition in `config.h`). The function should perform the operations according to the description given by equations (4) to (13).

In the same way as project 1B, the DSP-kit will test your LMS algorithm on start-up and give a pass/fail response. You cannot evaluate the LMS algorithm (and start working on your report) until the self-test succeeds.

Note: instructions on how to control the DSP-kit will be printed over the serial monitor when your LMS algorithm passes the self-test function. Press the indicated keys to control the behavior as needed to answer the questions you will address in your report.

2.3.1 LMS filtering example

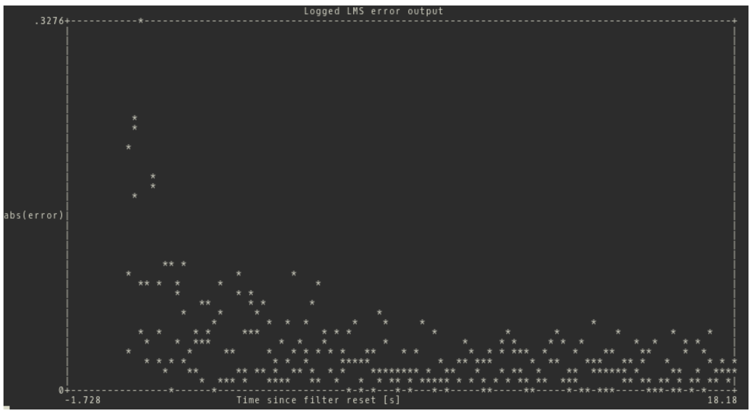
Here we will briefly show the result of a simple test case for

- $\approx$10 cm between DSP-kit and right speaker,
- the left speaker held against the users ear (and volume adjusted for a comfortable level with the filter disabled),
- the users other ear plugged with a finger (to more clearly hear the results from the left speaker),
- in a quiet room.

The following steps were performed:

1. DSP-kit programmed and self-test passed.
2. The music signal was disabled.
3. μ was set to $\approx 1 \cdot 10^{-3}$. This gave a reasonably short convergence time.
4. The left speaker was held against the users ear, volume reduced to make the noise level fairly loud, but comfortable.
5. At this stage, the room was kept quiet. No talking, no rustling of papers, no shuffling feet.
6. The filter was enabled with update enabled.
7. The filter coefficients were reset (to trigger starting recording the error output).

8. After just under 20 seconds, an error plot was generated, the results of which are shown here:



As can be seen, the average error (i.e. the average location of each `*` character at each time) decreased from an initial value of approximately 0.16 to an average value of approximately 0.02. The user listening to the speaker initially noticed that the disturbance signal was clearly audible, but after some ≈10 seconds only background noise was audible. Clearly, the LMS filter was doing its job correctly.

Note that the plotted error is simply

$$e = x - \hat{x} = s(n) + \underbrace{\sum_{k=0}^{\infty} h(k)\,y(n-k)}_{\text{noise through actual channel}} - \underbrace{\sum_{k=0}^{M-1} \hat{h}(k)\,y(n-k)}_{\text{noise through estimated channel}}$$

i.e. we cannot in this plot differentiate between actual/modeled channel mismatch and the music/all other signals. This is why we must keep the ambient noise level as low as possible (i.e. minimize $s(n)$).

We can also deduce that the primary cause of the nonzero error output is be due to background noise rather than a poorly adapted filter, as the user listening to the speaker didn't hear the disturbance, only background noise (e.g. people talking/doors closing in nearby rooms).

In practice, it might be hard for you to generate a nice-looking error plot. Generally speaking the most robust method for checking for convergence and evaluating the performance of the filter is to assign one group member to listen to the left speaker. If you are the group's official listener, keep the speaker directly placed against your ear (before training: adjust the volume using the serial monitor for a comfortable level) and plug your other ear with a finger or earplug. To evaluate the efficacy of the system use the serial monitor to switch between the filtered (improved) signal and the raw microphone signal.

9. At this stage, the LMS filter was regarded as converged (as there was no audible change in the error signal).
10. The music signal was enabled again, which was clearly audible in the speaker held against the users ear.
11. The system operation mode was switched between filtering enabled and disabled. The user could very clearly tell the difference between the two modes and could virtually not hear the disturbance signal at all when the filter was enabled.

2.4 Project Task: Part B real-time noise cancellation

Write a report where you address the following topics (in the listed order). *Most of these questions are fairly short and for the majority your answer shouldn't need to be more than a few lines.* These questions are ordered in roughly increasing difficulty and some later questions build on earlier results. Feel free to discuss additional findings!

We will use the following notation for convenience:

h_{sin}	The filter coefficients when trained for a sinusoidal disturbance
h_{BB}	The filter coefficients when trained for a broad-band disturbance
H_{sin}	The Fourier transform of h_{sin}
H_{BB}	The Fourier transform of h_{BB}

Mandatory coding section: 0 points

- Implement a working LMS algorithm on the DSP-kit and show the relevant source code (include the source code as an appendix to your document).
- Test how fast your C implementation is by increasing the number of LMS taps until the DSP-kit crashes. (Be sure to enable the LMS filter before testing.) The reference code-base (which is written with at least some speed consideration taken into account, but is not optimal in any sense) manages to work for up to around 425±10 taps while the filter is active, updating, and playing music. The exact number of maximum taps will vary during execution depending on (among other things) how much time the music MP3 decoder happens to require, but will usually lie somewhere near this value. Your code should be able to work with at least 200 taps to get good results.

Note: It takes (relatively speaking) a lot of CPU power to draw graphs and output data in a Matlab-compatible format. Be sure to disable the LMS filter when printing to avoid a system crash for long filter lengths!

Optional friendly competition coding section: 0 points

Note: addressing the tasks in this section is entirely optional. The group that completes this section as efficiently as possible will win a symbolic prize. Writing an efficient implementation will require more experience with the C programming language.

- Improve the performance of your functions. Can you make them as fast (or faster) than the reference implementation? How fast is your implementation (i.e. how long can the LMS filter be)? Hint: the micro-controller is very efficient at performing some SIMD (`https://en.wikipedia.org/wiki/SIMD`) instructions for up to four floating point numbers in successive locations in memory. In particular, we can make use of SIMD instructions for multiplication and addition. In order for the compiler to generate the relevant SIMD instructions your code must explicitly address four successive locations in memory and perform a supported numerical operation (which includes, among others, addition and multiplication). See Algorithm 1 for a naive for-loop example that will not generate the efficient SIMD instructions, and compare with Algorithm 2 that has unrolled the for-loop (`https://en.wikipedia.org/wiki/Loop_unrolling`) to explicitly address four successive elements on every iteration which will generate the appropriate SIMD instructions. Studying the source code of the `arm_xxx_f32` functions may be a good starting point for your code.

- Competition rules:
  - You may implement the `my_lms` function in any way of your choosing using the C programming language so long as valid machine code is generated using the standard development environment.
  - Your `my_lms` function must pass the self-test routine.
  - You may not modify any other program sections. Using "clever" `#define` statements to modify code outside of the `my_lms` function, overclock the DSP-kits, etc. is not permitted.

- The winner will be selected as follows:
  - The filter will be configured as:
    - LMS filtering enabled with filter update,
    - wide-band disturbance selected,
    - $\mu = 10^{-3}$,
    - music signal selected.
  - The filter length will first be set to a large value that does not cause the system to crash.
    - If the DSP kit does not crash after 5 seconds, the current length will be regarded as the current high-score and the filter length will then be incremented by one.

Algorithm 1 Code that will *not* compile to efficient SIMD instructions.

```
float sum_vec_8_naive(float * a){
    // Get the sum of the first 8 elements of a
    // using a naive for loop
    // Note: we assume a contains at least 8 elements!
    int i;
    float res = 0;
    for(i = 0; i < 7; i++){
        res += a[i];
    }
    return res;
}
```

Algorithm 2 Code that will compile to efficient SIMD instructions.

```
float sum_vec_8_simd(float * a){
    // Get the sum of the first 8 elements of a
    // using efficient SIMD instructions
    // Note: we assume a contains at least 8 elements!
    int i;
    float res = 0;
    for(i = 0; i < 2; i++){
        res += *(a++);
        res += *(a++);
        res += *(a++);
        res += *(a++);
    }
    return res;
}
```

    - If the DSP kit crashes, the stored high-score is regarded as the final score.
    - Note: when the music track repeats playback the MP3 decoder requires more processing power. Perform all tests no more than 5 seconds from the start/end of the track.
  - In your report, please indicate the maximum stable filter length for your code.
  - The TA will test and verify the performance of the groups with the highest reported performance. If multiple groups have very similar performance, the stability time will be increased from five seconds until one entry fails.
  - The TA reserves the right to disqualify entrants for poor sportsmanship, intentionally misinterpreting the rules, and so on.

Empirical section: 1-2 points per question

1. For a given speaker and DSP-kit setup, create stem plots of h_{sin} and h_{BB} for the longest filter your implementation can handle. Be sure not to move *anything* nearby between these tests, keep the physical channel unchanged, and be sure to reset the filter coefficients after changing the noise source. Also, don't forget to reverse the order of the filter coefficients as given by the DSP-kit. Show plots of the magnitude and wrapped phase (i.e. using MATLAB notation, a plot of `angle(fft(h,N)))` for some suitable length `N`) of H_{sin} and H_{BB}. Be sure to label all axes correctly and indicate their units. You do not need to analyze the plots in this question, we will return to these plots in the analytical section.

2. Stop updating the filter (i.e. set $\mu = 0$) after an initial training period with a step size that gives convergence after 5 – 10 seconds (i.e. a value of μ that gives a reasonably small residual error).

   a) What happens if the character of the input signal or channel is changed (e.g. move the disturbance/signal speaker, place a large book between the speaker and the DSP-kit, or some other change) without updating the filter coefficients? Why?

   b) Try raising/lowering the volume of the speaker outputs. This will raise/lower the amplitude of the disturbance (and the speaker playing the error output) and you can think of this as being equivalent to raising/lowering the amplification in the physical speaker. Why does/doesn't this give the same result as in subquestion 2a. *Note: remember that raising/lowering the volume also raises/lowers the volume of the improved signal speaker! For this question it may be best to use the utility that plots the last 15 seconds of the error signal.*

   c) Now test using a cell phone as a signal source (disable the music signal and play some music of your own using your cell phone). You'll (hopefully) see that the LMS filter successfully removes the disturbance signal. Move the cell phone around. Does the filter still work? Why is/isn't this the same behavior as in subquestion 2a?

3. How does a stem plot of h_{BB} change when changing from the maximum length, to 100, to 50, to 25 elements when the filter is continuously updated? Compare the results you get both when you do and do not reset the filter coefficients to zero after changing the filter length. Does the filter work equally well for all these lengths? Is there some critical length that, if the filter is shorter than, gives very poor performance? Can you determine this length from the stem plot of h_{BB} in question 1? *Hint: remember that the impulse response h_{bb} gives us an indication of how much previous channel inputs affect the output right now.*

4. How does a stem plot of $h_{\sin}$ change when changing from 100 to 10 elements when the filter is continuously updated? What about when changing from 10 to 100 elements? Do these results depend on whether you do or do not reset the filter coefficients after changing the filter length? Does the filter work equally well for all these lengths?

5. Train the LMS filter for the sinusoidal disturbance, then keep the filter coefficients fixed (i.e. set $\mu = 0$), and switch to the broad-band disturbance. How well does the $h_{\sin}$ filter effectively attenuate the broad-band disturbance compared to the h_{BB} filter? Now, try the opposite (train h_{BB}, switch to sinusoidal noise). How well does h_{BB} attenuate the sinusoidal noise compared to $h_{\sin}$? Why do you get these results?

6. Set mu to a modestly large value $\approx 1 \cdot 10^{-2}$, enable the broad-band disturbance, and use your cell phone as a signal source again, but this time hold it very close to the DSP-kit and raise its volume until the red LED is lit almost continuously (we are using the cell phone here only because it can play music louder than the DSP-kit's speakers). This indicates the microphone input is saturated `https://en.wikipedia.org/wiki/Clipping_(audio)`. After a few seconds, generate a plot of the filter coefficients, denoted $h_{bb,sat}$. Does this look significantly different compared to h_{bb}? Why? *Hint: remember that the LMS method assumes a linear channel. For nonlinear channels many of our mathematical tools can no longer be used, e.g. for nonlinear channels we are not ensured that the output can be viewed as the convolution of the input and impulse response. Is clipping a linear operation? E.g. does the additive property hold: $f(x+y) = f(x) + f(y)$, where x, y is a sound pressure signal and f is the microphones operation of sampling the air pressure? If clipping is a nonlinear operation, i.e. giving a nonlinear channel, then why can LMS not estimate the nonlinear channel?*

Analytical section, 1 point per question

1. On the step size μ,
   a) How does the choice of step size affect how quickly the filter adapts to the channel?

b) Will a large enough step size cause the estimated channel to diverge? Why?

c) Does the choice of the step size affect how well the filter works once it has converged?

2. In the previous section we tested changing the filter length in real-time for the two noise sources. We also compared the results where we in one case reset the filter coefficients after changing the length, and in the other case did not reset the filter coefficients. Why do we get so radically different behavior between the noise sources? *Hint: if the filter works equally well with different filter coefficients then the solution that minimizes the LMS objective* (12) *is not unique.*

3. Set $\mu = 1 \cdot 10^{-3}$, fully train the LMS filter for the sinusoidal disturbance, and finally switch to the broad-band disturbance *without resetting the filter coefficients.* Wait 5 seconds, and then save the resulting channel coefficients and create a stem plot of them (denote this channel estimate as $h_{\text{sin}\to\text{BB}}$). Now do the opposite (start with broad-band disturbance, switch to sinusoidal, wait 5 seconds, plot resulting coefficients, denote this channel estimate as $h_{\text{BB}\to\text{sin}}$). How do these channel estimates compare to h_{BB} and h_{sin}? Why do they look the particular way they do? *Hint: consider the LMS objective* (12) *and how* $Y(\omega)$ *differs for the two noise sources. Also, consider solving analytical question6 first.*

4. What is the magnitude and phase of $H_{\text{sin}}(f_0)$ and $H_{\text{BB}}(f_0)$, where $f_0 = 440\,\text{Hz}$ (the frequency of the sinusoidal disturbance)?[2] Do you expect them to be the same, similar, or entirely different? Which estimate do you expect to be most accurate at f_0? Why? Are H_{sin} and H_{BB} identical for other frequencies? Why/why not? *Hint: refer to the LMS objective* (12).

5. For the sinusoidal disturbance, what is the minimal theoretical length that is required? Show why this length is sufficient. *Hint: Consider the LMS objective function* (12). *Do we need* $H = \hat{H}$*for all frequencies to get zero error? We can't know exactly what* H *will be beforehand, but we do know that it will be a complex-valued function. Can you find the filter* $h = [h_0, h_1, \ldots, h_N]$ *with minimum length* N *whose DTFT is correct at the necessary locations in the frequency domain?*

6. For filter lengths longer than the absolute minimum, h_{sin} has a very distinctive form. Why does the LMS algorithm generate this particular solution? In the empirical section, question 4, when you increased the filter length you got a different form, why? *Hint: refer to the filter update step* (13). Note that for long filter lengths ($\gtrsim$100 elements) the filter's impulse response will tend to look like a decaying exponential. You do not need to consider why this is the case and can instead consider the case for ≈ 50 elements, where the impulse response has a simpler shape[3].

[2] Don't forget that you can use zero-padding to increase the resolution of the Fourier transforms to evaluate at something closer to 440 Hz.

[3] For those that are interested, the reason for the decaying exponential is that the sinusoidal disturbance is not perfectly generated. The actual frequency will tend to vary in a range $440 \pm f_\varepsilon$

2.5 Project examination

The work of each project group is evaluated through the report, the C-code and an oral exam. Write a report that describes your work and findings. Try to include answers to the questions posed above. Completed reports for this project should be uploaded to Canvas before the deadline indicated. Include a title-page containing the project number, group name, and student names.

Do not let the report exceed five pages. This limit does not include the cover page, pages with figures, and the appendix. It is OK (preferable even) to interleave figures and text as long as the total length of text does not exceed 5 pages with all figures removed.

In the oral exam the teacher will discuss different aspects of the project with the students. It is an opportunity for the students to ask questions to learn more, and for the teacher to sense the level of understanding in the group. Time for the oral exam is booked on Canvas.

References

[1] B. Mulgrew, P. Grant, and J. Thomson. *Digital Signal Processing - Concepts and Applications.* MacMillan Press Ltd., Basingstoke, United Kingdom, 1999.

[2] T. McKelvey *Applied Signal Processing - An introduction to the C-programming language.* Available on Canvas, 2016.

for some small random time-varying term f_ε. This in turn implies that the phase of the disturbance varies with a small random term. Using this you can, if you wish, describe why the filter coefficients decay exponentially.