# Lab 4

This lab will introduce you to filtering signals in Matlab. We'll start by doing everything explicitly before we get into the easier methods in later labs.

## Basic Filtering

The concept we will be covering in this lab is a first-order feed-forward filter. The basic form of a simple feed-forward filter can be described with the difference equation:

$$y[n] = a_0x[n] + a_1x[n-n_0] + \dots$$

where $x[n-n_0]$ is the value of the input signal $n_0$ samples ago. This is your basic linear combination of delayed versions of the input signal combined to produce a new output signal. A zeroth order filter will only have an x[n] and y[n] component. A first order filter will have one additional component and will look like:

$$y[n] = a_0x[n] + a_1x[n-1]$$

Implementing a filter like this in most systems (say in c++, for example) would probably involve some sort of "for loop" that would look like this:

```
for(int i = 1; i < length(x[n]); i++)
   y[i] = a0*x[i] + a1*x[i-1];
```

**Don't do this in MATLAB.** MATLAB does in fact have for loops, but they are horribly inefficient.  Insetad, you want your code to look something like this

```
y = a0 * x + a1 * xDelayed
```

where x and xDelayed are two vectors. The simple way to do this is to append a 0 onto the beginning of x:

```
xDelayed = [0, x];
```

The signals need to be the same length, though, so we can fix this by appending a 0 onto the end of x.

```
x = [x, 0];
```

In this lab you're going to implement a first-order feed-forward filter using varying coefficients a0 and a1.

## Assignment

Note: In this lab we will use a slightly different function for plotting the magnitude spectra of signals. It is called noiseMagPlot.

### Problem 1

A. Implement a first-order feed-forward filter using coefficients **$a_0$ = 0.5 and $a_1$ = 0.5**

B. Generate a noise signal using rand() with values in the range of +/-1. Filter the noise signal using the above coefficients. Plot both the input signal and the output signal in a 2 x 1 subplot. Plot the magnitude spectrum (using noiseMagPlot) of the input signal and the output signal. Play the signal (no need to export to .wav). **Describe your observations.**

C. Read in a wav file. Filter this signal through the same filter as above. Plot both the input signal and the output signal in a 2 x 1 subplot. Play the signal and save as **YourName-Lab4-1C.wav**. **Describe your observations.** Please include **both** the original and the filtered version with your submission.

D. Repeat Part A using coefficients **$a_0$ = 0.5 $a_1$ = -0.5**

E. Repeat Part B using the new coefficients.

F. Repeat Part C using the new coefficients (save as **YourName-Lab4-1F.wav**)

### Problem 2

In the examples above we only worked with a first-order filter. How would we create a second-order filter?

A. Implement a second-order feed-forward filter using coefficients of your choosing.

B. Filter a noise signal and plot the magnitude spectrum as you did in 1B and 1E.

C. Based on the coefficients you chose, does the magnitude spectrum do what you expected or not? Was the overall amplitude of the signal affected? How does the spectrum differ from those we saw in Problem 1?

# Lab 4 – Basic Filtering

Nate Paternoster

## Part 1

a. *Implement a first-order feed-forward filter using coefficients* ***$a_0$ = 0.5 and $a_1$ = 0.5***

To implement a first-order feed-forward filter you must consider an input x and output y. The basic equation for this filter is: $y[n] = a_0x[n] + a_1x[n-n_0] + \ldots$ For a first-order filter, only x[n] and x[n-1] must be summed. Basically, the original signal is having a 1-sample delayed version of itself added to it. Both the original signal and the delayed signal have their own amplitudes $a_0$ and $a_1$.

b. *Generate a noise signal using rand() with values in the range of +/-1. Filter the noise signal using the above coefficients. Plot both the input signal and the output signal in a 2 x 1 subplot. Plot the magnitude spectrum (using noiseMagPlot) of the input signal and the output signal. Play the signal (no need to export to .wav).* ***Describe your observations.***

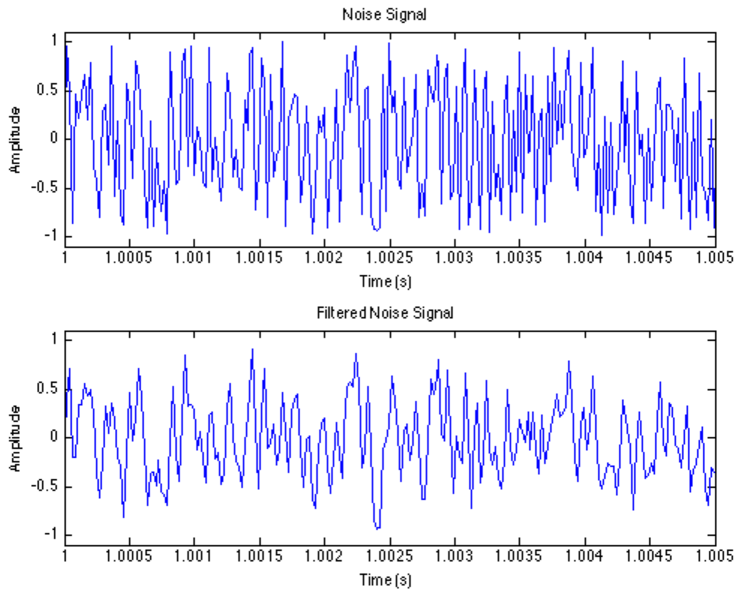


Figure 1a: The input signal compared to the output signal

The input signal is noise with amplitude between -1 and +1. After filtering once by adding a delayed version of the signal, the filtered output is shown below. The waveform overall contains less peaks and has a bit of an overall lowered amplitude. The output appears to be a lower frequency than the input.

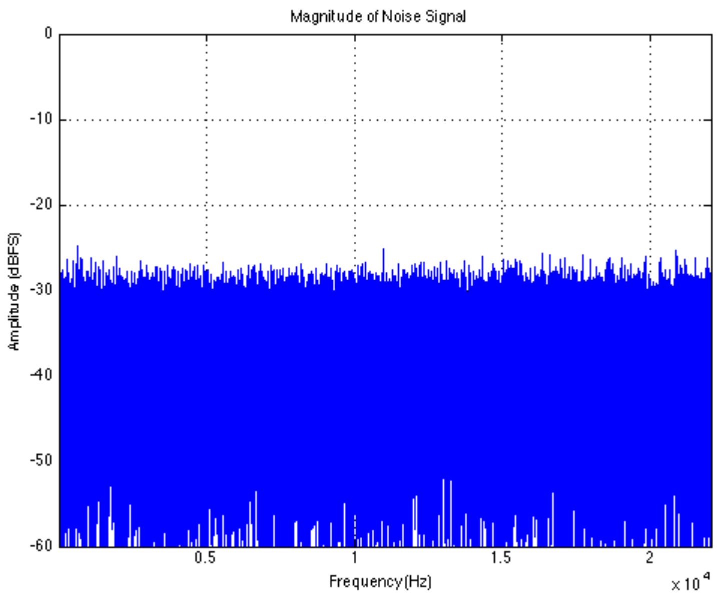


Figure 1b: The magnitude spectrum of the original signal

The magnitude spectrum shows the relative gain and attenuation of the signal over a wide spectrum of frequency (the range of human hearing). As you would expect, the unchanged noise signal has a relatively uniform gain throughout all frequencies.

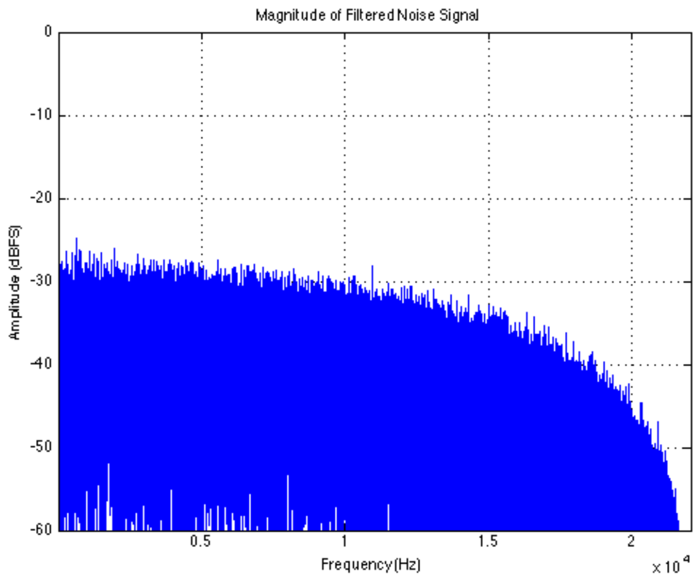


Figure 1c: The magnitude spectrum of the filtered signal

The magnitude spectrum of the filtered signal shows that higher frequencies are being cut out. What we designed was a low-pass filter. The filtered signal sounds noticeably more “muffled” than the original.

c. *Read in a wav file. Filter this signal through the same filter as above. Plot both the input signal and the output signal in a 2 x 1 subplot. Play the signal and save as* ***YourName-Lab4-1C.wav****.* ***Describe your observations.*** *Please include* ***both*** *the original and the filtered version with your submission.*

This time we read in a wav file of anything and put it through the same filter. The results stayed consistent. I chose a wav file with a lot of hi-hat. Although the high frequency attenuation was difficult to see on the graphs, I could make out less of the hi-hat's upper frequencies in the filtered version. Because the sampling rate is 44.1k, the cutoff frequency will begin at ¼ of this (half of $F_n$). In a typical audio recording, this will be hard to notice because there are very small amounts of frequencies above 12k-15kHz.

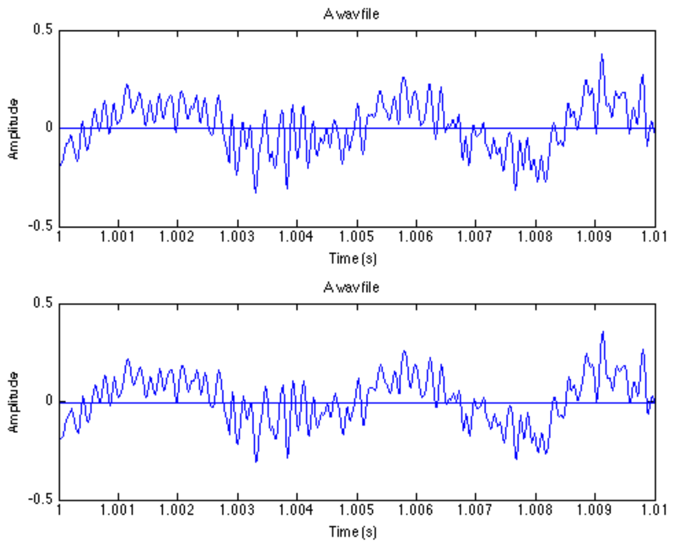


Figure 1d: The input signal compared to the output signal

It is difficult to see the lowpass filter in action here. But if you look closely you can see slightly less pointy peaks (suggesting a general smoothing of the waveform) and some slight attenuation.

d. *Repeat Part A using coefficients* $\boldsymbol{a_0 = 0.5\ a_1 = -0.5}$

By using a negative amplitude on the delayed version of the signal, we end up with the general equation: $y[n] = a_0x[n] - a_1x[n-n_0] - \ldots$ As opposed to Part A, this filter removes the delayed versions of the signal from the original signal. For a first-order filter, x[n-1] is removed from x[n].

e. *Repeat Part B using the new coefficients.*

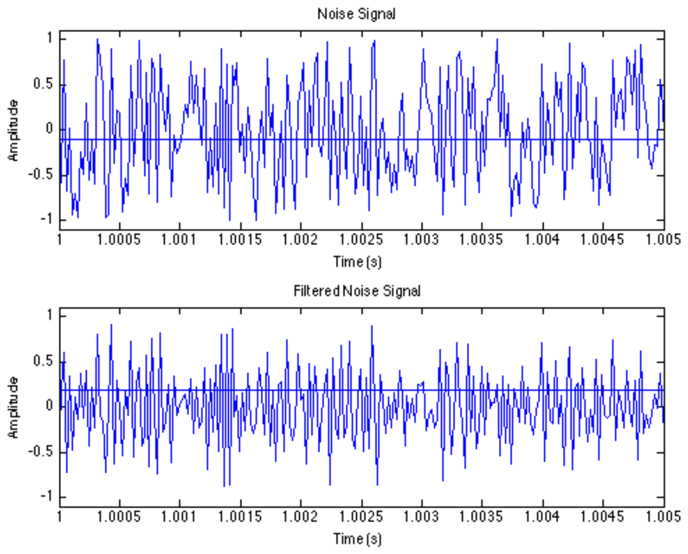


Figure 1e: The input signal compared to the output signal

As opposed to Part B, the new filtered signal looks more "condensed" than the original. It looks like the output waveform is of a higher frequency than the input. Similarly to before, the output experienced general attenuation.

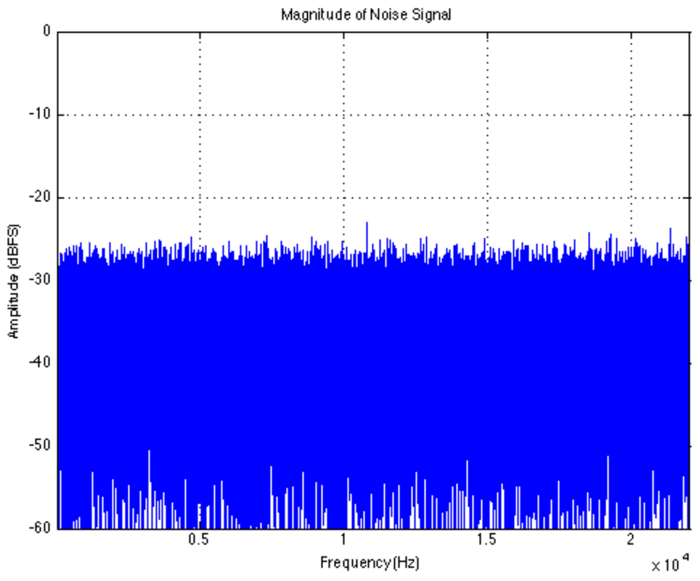


Figure 1f: The magnitude spectrum of the original signal

The input magnitude spectrum is identical to before – uniform gain across all frequencies.

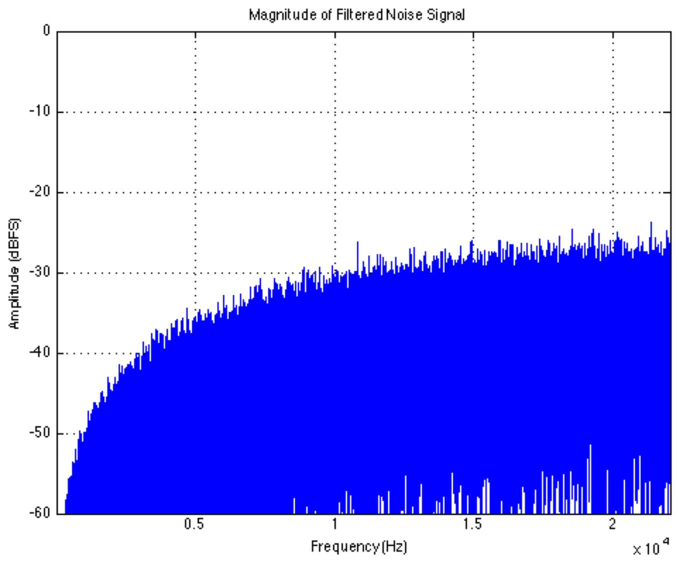


Figure 1g: The magnitude spectrum of the filtered signal

The new filter is a high-pass filter. Subtracting the delayed version of a signal rather than summing it results in the lower frequencies being cut out. The filtered signal sounds brighter than the original and more "tinny."

f. *Repeat Part C using the new coefficients (save as **YourName-Lab4-1F.wav**)*

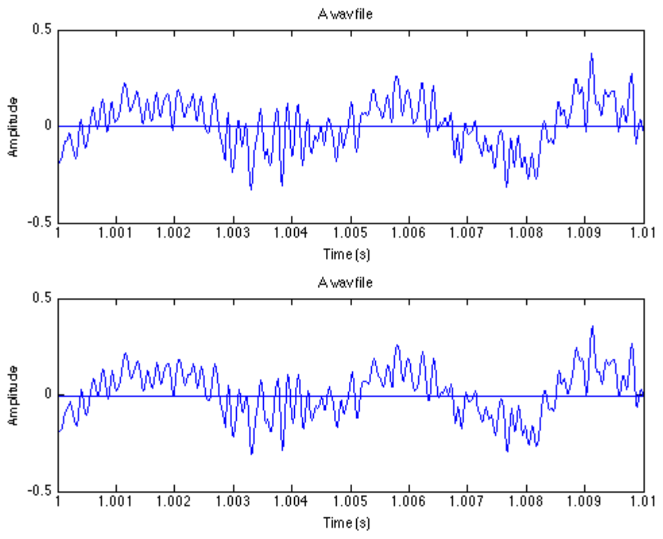


Figure 1h: The input signal compared to the output signal

Like before it was hard to hear a difference. However, the waveform is being put through a high-pass filter and will show a tighter concentration of peaks on the output of the graph.

<u>Part 2</u>

a. *Implement a second-order feed-forward filter using coefficients of your choosing.*

If we consider the general equation for a feed-forward filter to be $y[n] = a_o x[n] + a_1 x[n-n_o] + \ldots$, then we can build a *second-order* filter by adding both a 1-sample delayed version of and the signal and a 2-sample delayed version of the signal to the original signal. The equation would become $y[n] = a_o x[n] + a_1 x[n-1] + a_2 x[n-2]$. The three amplitudes $a_0$, $a_1$, and $a_2$ are to be chosen arbitrarily.

b. *Filter a noise signal and plot the magnitude spectrum as you did in 1B and 1E.*

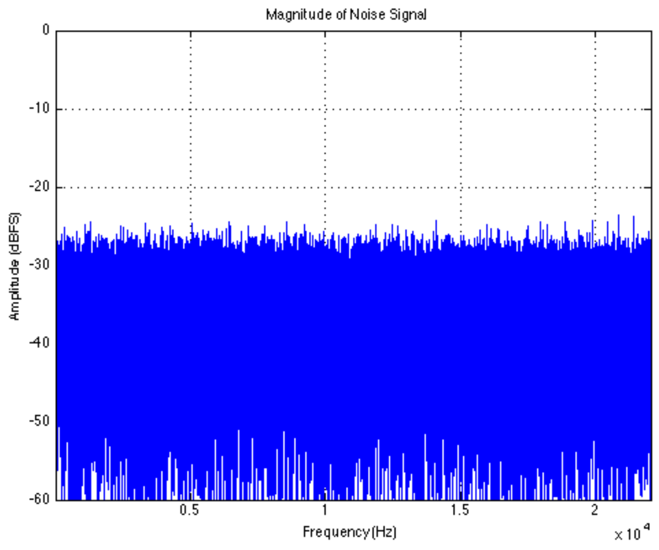


Figure 2a: The magnitude spectrum of the input signal.

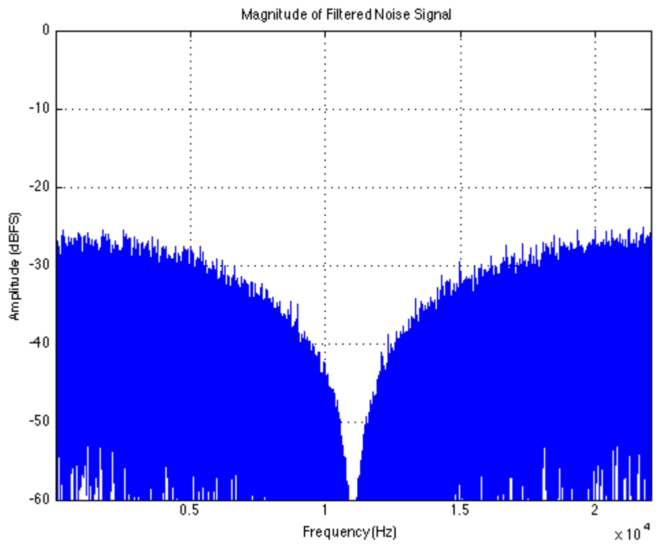
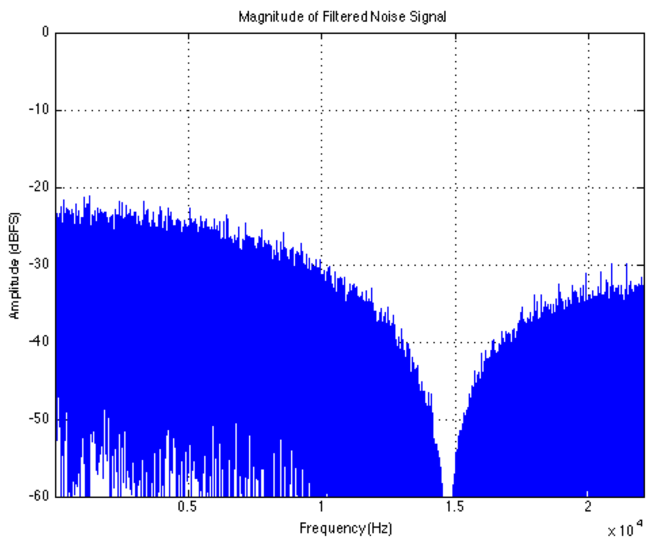


Figure 2b: The magnitude spectrum of the output signal.

c. *Based on the coefficients you chose, does the magnitude spectrum do what you expected or not? Was the overall amplitude of the signal affected? How does the spectrum differ from those we saw in Problem 1?*

I chose 0.5 for all three coefficients for this graph pictured. I expected the graph to be a steeper sloping lowpass filter. Even though it is steeper sloping, it also rises up again after cutting off so that it looks more like a notch filter. After messing with some amplitude values I got various graphs, one of which looked like this:



I found that $A_0$ and $A_2$ could affect the slope of the cutoff while $A_1$ could not really. If both $A_1$ and $A_2$ are made negative, or $A_0$ is made negative, the graph looks more like a bandpass filter, cutting off both extremes.

The overall amplitude of the signal was not really affected. At its highest it remained roughly around -20 to -25 dBFS. Unlike the plots from first-order filters, this plot contains a pole and has more than one slope.