Abstract

This project is using an adaptive FIR filter to attempt to subtract noise out of a noisy signal. As the filter adapts the noise gets more and more subtracted out of the signal allowing the signal to be cleaner and more like the original signal. The more coefficients there are in this filter the cleaner the signal and more understandable the processed signal became. When the filter length was only 4, the SNR improvement in the signal was about .3 dB, but when the filter length was 256 the improvement in SNR was closer to 10dB.

Results

N	SNR_Improvment	Subjective Rating(1-5)	Best Mu
4	0.2969	1	0.0006
16	1.4297	1	0.0006
64	5.6519	1	0.0008
128	8.0907	2	0.0007
256	9.9099	2	0.0006

Figure 1: SNR and best Mus table

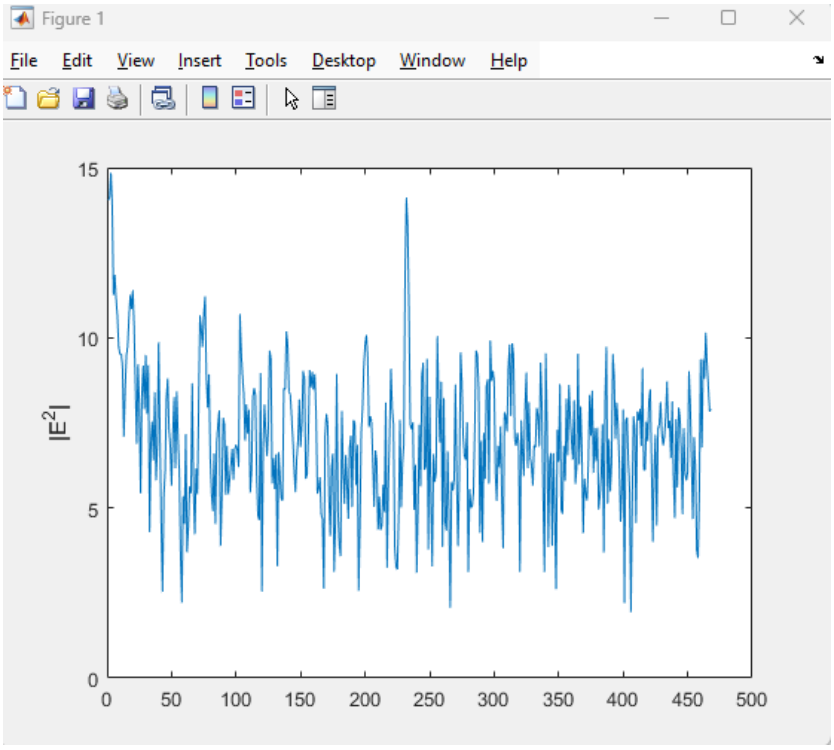

Figure 2: Convergence Curve for N = 128

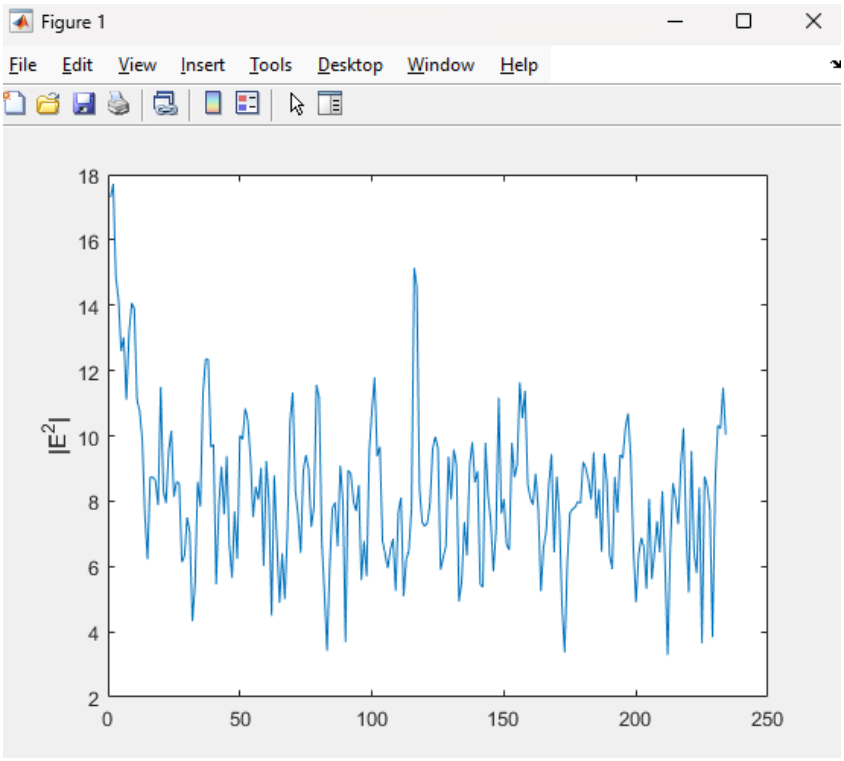

Figure 3: Convergence curve for N = 256

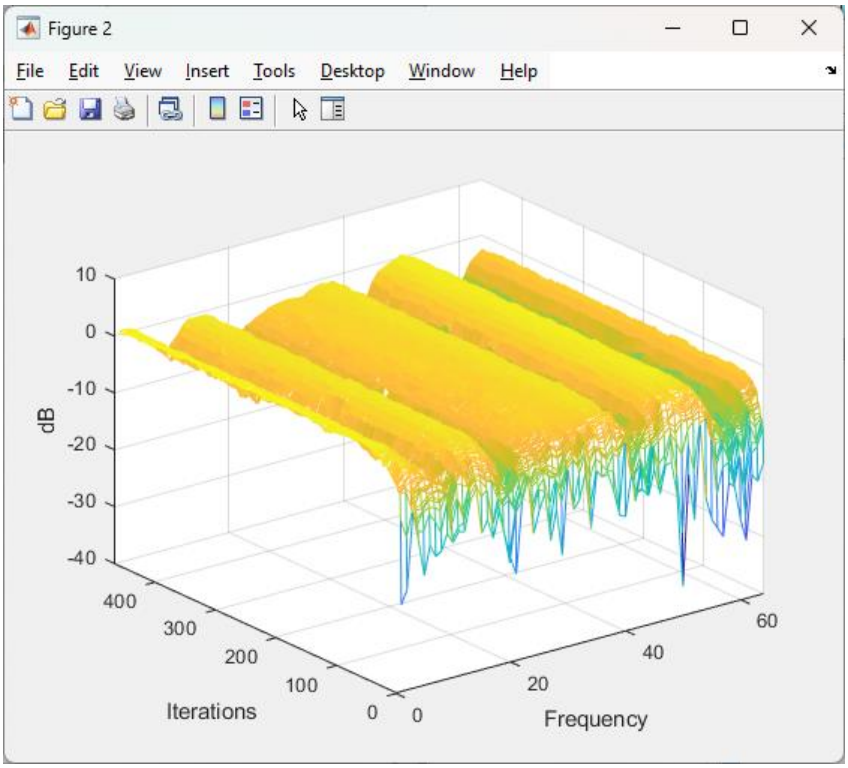

Figure 4: Filter coefficients for N =128, mu = 0.0005

Explanation of results

If you look at figure 2 and figure 3 you can see the error starts high and then levels out as the coefficients adjust. This is expected for an adaptive filter. The lower the error is the cleaner the signal is. The Mesh plot of the B coefficients is showing at each iteration how the filter was performing at the different filter indices.

1. B(k) are the filter components of a particular frame. The filter is trying to allow frequencies that represent the noise through so that we can subtract them out of the signal with the noise.
2. The more samples we are put into the FFT the better the SNR can get.
3. The order of the filter, since it is type FIR, is N-1
4. Mu is how quickly the adaptive filter can adapt. Large mus represent large steps in B. Too large of a mu and the filter can become unstable. Small mus represent small steps in the B, too small of steps and the filter wont converge quickly enough.
5. SNR is the Signal to Noise Ratio. Meaning the smaller the amount of noise we are allowing through our filter the more like the original signal we are passing through.
6. Yes, I noticed at the higher sized filters that there was a fluttering sound. Almost like putting a playing card between spokes on a bicycle.
7. We are trying to get as close to the original signal as possible. The signal e is equal to our original signal + noise – modeled noise. Minimizing e is getting our modeled noise to get as close to the actual noise as possible.
8. For each iteration of B, you can take the inverse FFT to get the impulse response.

Conclusion

This project helped me understand how an adaptive filter works. How the goal is to figure out where in the spectrum the noise is so that we can subtract it out of the signal we are trying to clean up. This project also helped me to understand how to use MATLAB to accomplish a filter like this. This project pushed me to think deeper about how my noise canceling headphones work and to think about where this could be applied to the real world like a conference where there is a public speaker, or satellites where there is background radiation that is corrupting the signals. You would want adaptive filters to clean these signals up. In the real world, static filters are often sufficient but often are not, and with techniques like this we are able to solve some complex problems.

Appendix

```
close all;
[s1, fs1] = audioread('mic1.wav'); % reads file mic1.wav file.
[s2, fs2] = audioread('mic2.wav'); % reads mic2.wav file.
[s,fs]    = audioread('cleanspeech.wav');
%sound(s1);
%sound(s2);
%sound(s);
mu = 0.0007; %step size
N = 128;     %size of the FFT
K = fix(length(s1)/N); %Number of frames
%%
d = ones(1, N);
x = ones(1, N);
B = zeros(N, 1);
e = zeros(1, N);
Xdiag = ones(N,N);
e_total = zeros(length(s1), 1);
E_db = zeros(1,K);
B_plot = zeros(K,N/2);

for k=1:K %looping through the signal to clean it up
    n = (1:N)+(N*(k-1));
    d = s1(n);
    D = fft(d);
    x = s2(n);
    X = fft(x);
    Xdiag = diag(X);

    E = D-Xdiag*B; %creating the error signal
    E_db(k) = 10*log10((E'*E)/N); %save the error calculation
    B = B + 2*mu*Xdiag'*E; %adjust the filter coefficients
    B_plot(k,:) = abs(B(1:N/2).'); %save the filter coefficients
    e = ifft(E);    %Convert the error signal to time domain
    e_total(n) = e; %save the signal in the vector
end

plot(E_db);
ylabel('|E^2|')

figure;
mesh(10*log10(B_plot))
xlabel('Frequency')
ylabel('Iterations')
zlabel('dB')

denom1 = s-e_total;
denom2 = s-s1;
SNRa=10*log10(((s'*s)/N)/((denom1'*denom1)/N))
SNRb=10*log10(((s'*s)/N)/((denom2'*denom2)/N))
SNRImp = SNRa - SNRb
%sound(e_total);
```