Final Project in Digital Signal Processing – 2023

Submitted by Ariel Fuxman & Gilad Zusman

Tuesday 14th February, 2023

Echo generation and cancellation

1. Imagine a standard echo scenario: a microphone is set in the center of a round cave, and someone claps near the microphone. The claps are measured by $x(n)$ and the microphone records $y(n)$. The microphone, at unit time n, simultaneously records the current clap but also records previous claps, which happened d time units ago. d is the time that takes the sound to travel to the cave's wall and back, and since the sound waves traveled a distance and got reflected of the wall, they are suppressed by a factor of α.

2. We can find $h(n)$ by equating the coefficients on the two sides of the convolution equation:

$$y(n) = x(n) * h(n) = \sum_{m=-\infty}^{\infty} x(n-m)h(n) = x(n) \cdot \underbrace{1}_{h(0)} + x(n-d) \cdot \underbrace{\alpha}_{h(d)}$$

summarizing,

$$h(n) = \begin{cases} 1, & n = 0 \\ \alpha, & n = d \\ 0, & \text{otherwise} \end{cases}$$

We find the transfer function $H(z)$ by definition:

$$H(z) = \sum_{n=-\infty}^{\infty} h(n) z^{-n} = 1 + \alpha z^{-d}$$

Similarly, the Fourier transform $\hat{h}(u)$ can also be found by its definition:

$$\hat{h}(u) = \sum_{n=-\infty}^{\infty} h(n)e^{-2\pi iun} = 1 + \alpha e^{-2\pi iud} = 1 + \alpha\cos(2\pi ud) - i\sin(2\pi ud)$$
$$\left|\hat{h}(u)\right|^2 = (1 + \alpha\cos(2\pi ud))^2 + (-\alpha\sin(2\pi ud))^2 = 1 + \alpha^2 + 2\alpha\cos(2\pi ud)$$

this is a cosine wave with frequency d, so on the interval $[0, \frac{1}{2}]$ which we are interested in it completes $\frac{d}{2}$ cycles. The kind of filter depends on the value of d. As long as $\alpha \in (0, 1)$, it is just responsible for stretching. We sketch a graph of $|\hat{h}(u)|$ on the interval $[0, \frac{1}{2}]$ for different values of d, picking $\alpha = \frac{1}{2}$ arbitrarily:

Figure 1: For $d = 1$ we get a low-pass filter

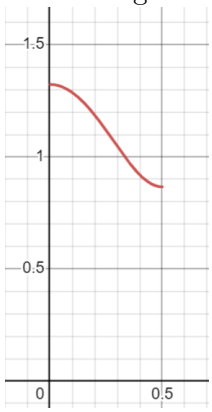


Figure 2: For $d = 2$ we get a band-stop filter

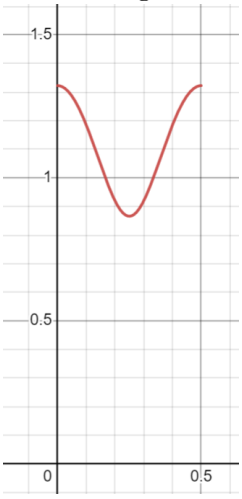


Figure 3: For $d = 3$ our filter is sort of a low-pass, but also has the effect of weakening a certain range of frequencies

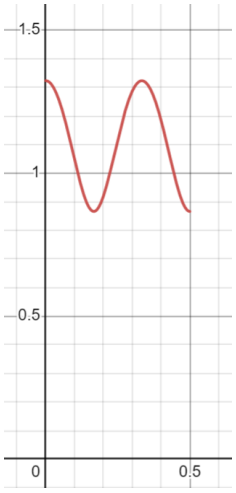
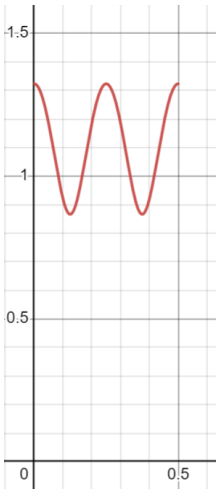


Figure 4: For $d = 4$ we get a band-stop filter, but which reduces weakens frequencies in two different frequency ranges



In general, if we represent $d = 2q + r$, $q \in \mathbb{N} \cup \{0\}$, $r \in \{0, 1\}$, then:

- For $r = 0$, the filter is a band-stop that weakens frequencies in q different frequency ranges.
- For $r = 1$, the filter is a low-pass that also weakens frequencies in q different frequency ranges.

3. Denote the transfer function that transforms $y(n)$ to $x(n)$ in the z-plane as $\tilde{H}(u)$ and the impulse response as $\tilde{h}(n)$.

$$x(n) = y(n) * \tilde{h}(n) \Longrightarrow X(z) = Y(z)\tilde{H}(z) \Longrightarrow \tilde{H}(z) = \frac{X(z)}{Y(z)} \Longrightarrow \tilde{H}(z) = \frac{1}{H(z)}$$

$$\tilde{H}(z) = \frac{1}{1+\alpha z^{-d}} = \frac{z^d}{z^d+\alpha}$$

Now, we find the poles:

$$\begin{aligned} z^d + \alpha &= 0 \\ z^d &= -\alpha \\ z^d &= \alpha e^{i\pi} \\ z &= \alpha^{\frac{1}{d}} e^{\frac{\pi(1+2k)}{d}},\ k = 0, ..., d-1 \end{aligned}$$

The poles are of magnitude $\alpha^{\frac{1}{d}}$, so they lie inside the unit circle since $\alpha \in (0,1)$. The difference equation clearly shows the inverse filter is causal, so $R_2 = \infty$ and the range of convergence is $\alpha^{\frac{1}{d}} < |z| < \infty$. The unit circle $|z| = 1$ is inside the range of convergence, hence it is stable and its Fourier transform converges. To find the Fourier transform, we substitute $z = e^{2\pi i u}$ in the transfer function:

$$\frac{1}{1+\alpha e^{-2\pi i u d}}$$

The magnitude spectrum is just the reciprocal of $|\hat{h}(u)|$:

$$\frac{1}{\sqrt{1+\alpha^2+2\alpha\cos(2\pi u d)}}$$

So, being the reciprocal, we can conclude the filter kind by imagining drawing the reciprocal. Assume again $d = 2q + r$,

- For $r = 0$, the filter is a band-pass that strengthens frequencies in q different frequency ranges
- For $r = 1$, the filter is a high-pass that also strengthens frequencies in q different frequency ranges.

4.

$$\phi_{yy}(k) = \sum_{n=-\infty}^{\infty} y(n)y(n-k) = \sum_{n=-\infty}^{\infty} \left(x(n) + \alpha x(n-d)\right)\left(x(n-k) + \alpha x(n-k-d)\right) =$$

$$= \underbrace{\sum_{n=-\infty}^{\infty} x(n)x(n-k)}_{S_1} + \alpha \underbrace{\sum_{n=-\infty}^{\infty} x(n)x(n-k-d)}_{S_2} + \alpha \underbrace{\sum_{n=-\infty}^{\infty} x(n-d)x(n-k)}_{S_3} + \alpha^2 \underbrace{\sum_{n=-\infty}^{\infty} x(n-d)x(n-k-d)}_{S_4}$$

To express S_3 and S_4, we substitute $m = n - d$:

$$\begin{aligned}
S_1 &= \phi_{xx}(k) \\
S_2 &= \phi_{xx}(k+d) \\
S_3 &= \sum_{n=-\infty}^{\infty} x(n-d)x(n-k) = \sum_{m=-\infty}^{\infty} x(m)x(m-(k-d)) = \phi_{xx}(k-d) \\
S_4 &= \sum_{n=-\infty}^{\infty} x(n-d)x(n-k-d) = \sum_{m=-\infty}^{\infty} x(m)x(m-k) = \phi_{xx}(k)
\end{aligned}$$

$$\phi_{yy}(k) = (1+\alpha^2)\phi_{xx}(k) + \alpha\left(\phi_{xx}(k+d) + \phi_{xx}(k-d)\right)$$

5. We can think of our signals as elements in the inner product space:

$$l_2 = \left\{ \{x(n)\}_{n=-\infty}^{\infty} : \sum_{n=-\infty}^{\infty} |x_n|^2 < \infty \right\}, \text{ (Here our field is taken to be } \mathbb{R})$$

with the inner product defined as:

$$\langle x(n), y(n) \rangle = \sum_{n=-\infty}^{\infty} x(n)y(n)$$

Then, it is apparent that:

$$\phi_{xx}(k) = \langle x(n), x(n-k) \rangle$$

The norm of a signal is invariant under time-shifting:

$$||x(n-k)||^2 = \sum_{n=-\infty}^{\infty} x(n-k)x(n-k) \underbrace{=}_{m=n-k} \sum_{m=-\infty}^{\infty} x(m)x(m) = ||x(n)||^2$$

Recall Cauchy-Schwartz inequality:

Theorem. *Any two elements $x, y \in V$ in an inner product space satisfy:*

$$|\langle x, y \rangle| \leq ||x|| \cdot ||y||$$

and equality is achieved if and only if x, y are linearly dependent.

Applying this to our case, we get:

$$\phi_{xx}(k) = \langle x(n), x(n-k) \rangle \leq ||x(n)|| \cdot ||x(n-k)|| = ||x(n)||^2$$

This means that $\phi_{xx}(k)$ has a maximum at $k = 0$ (the maximal possible value is achieved there). It is also a global maximum (assuming $x(n)$ it is not the zero signal), for if $x(n) = bx(n-k)$ for some $k \in \mathbb{Z}, b \in \mathbb{R}$, then if $b = 1$ then $x(n)$ is a periodic sequence and if $b \neq 1$ then it has a geometric sub-sequence which diverges at one side. Both cases contradict the fact that $x(n) \in l_2$. It also means that ϕ_{yy} has a global maximum at $k = 0$ (since our proof applies for any signal). Finally, looking again at the expression:

$$\phi_{yy}(k) = (1 + \alpha^2)\phi_{xx}(k) + \alpha\left(\phi_{xx}(k+d) + \phi_{xx}(k-d)\right)$$

we see that to maximize $\phi_{yy}(k)$, we can maximize $\phi_{xx}(k+d), \phi_{xx}(k-d)$ or $\phi_{xx}(k)$, meaning that there is a maximum at $k = -d, 0, d$.

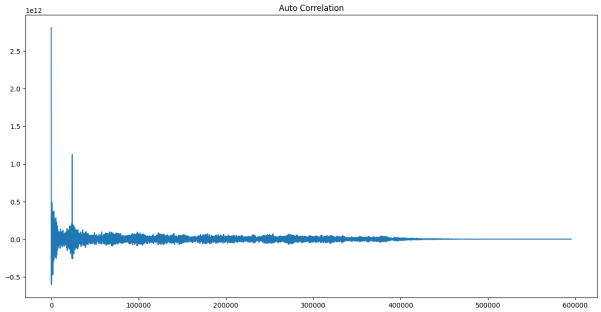
6. In practice, our signal $x(n)$ is actually a finite array of length N:

$$x = [x(0), \ldots, x(N-1)]$$
$$y = [x(0), \ldots, x(d-1), x(d) + \alpha x(0), \ldots, x(N-1) + \alpha X(n-d-1), \alpha x(N-d), \ldots, \alpha x(N-1)]$$

In the first step we compute the auto-correlation array. To compute $\phi_{yy}(k)$ for each $k = 0, 1, \ldots, N+d-1$, the direct method takes a total of $(N+d-1)+(N-d)+\ldots+1 = O((N+d)^2)$ operations. An alternative is realizing that the auto-correlation array can be written as $\phi_{yy}(k) = y(n) * y(-n)$. The convolution of two finite signals can be efficiently computed by performing point-wise multiplication in the Fourier domain, and returning to the time domain (this is similar to other convolution theorems we have learned, although there are some subtleties like the convolution needing to be circular, but we won't get into detail since there is a library function in Python which exactly deals with that). This takes only $O\left((N+d)\log(N+d)\right)$ operations, by using the FFT and the IFFT algorithms. In practice, we saw the running time decrease from 1.5 minutes to barely a second.

Now, look at the following heuristic:

Figure 5: A typical graph of the auto-correlation of y, with parameters $d = 24000$ and $\alpha = 0.5$



As we see, $\phi_{yy}(k)$ has its strongest peak at $k = 0$, this settles with what we proved. Then, the graph descends and stays quite uniformly around 0, until reaching its second peak at d. This hints us that to reconstruct d, we start at $k = 0$, descend with the curve until it starts increasing again at some point q, and from that point we can just take $d = \text{argmax}_{k \geq q} \phi_{yy}(k)$.

To reconstruct α, we notice that y is of length $N + d$. After reconstructing d we can find N by the formula $N = \text{len}(y) - d$. To continue, we set $m = \lceil \frac{N}{d} \rceil$ and look at the following set of $m + 1$ equations labeled $0, 1, \ldots, m$:

$$\begin{aligned} y(0) &= x(0) \\ y(d) &= x(d) + \alpha x(0) \\ y(2d) &= x(2d) + \alpha x(d) \\ \vdots &= \vdots \\ y((m-1)d) &= x((m-1)d) + \alpha x((m-2)d) \\ y(md) &= \alpha x((m-1)d) \end{aligned}$$

At step number j we add to equation (m) the equation $(m - j)$, but times $(-\alpha)^j$. The result is that if we do l such steps $(l = 0, \ldots, m - 1)$,

$$\sum_{j=0}^{l} (-\alpha)^j y((m-j)d) + (-\alpha)^{l+1} x((m-l-1)d) = 0$$

In particular, because $y(0) = x(0)$, for $l = m - 1$ we get:

$$\sum_{j=0}^{m} (-\alpha)^j y((m-j)d) = 0$$

The last equation allows us to solve for α numerically, since the numbers $y(0), y(d), \ldots, y(md)$ are known.

7. We took a larger d because small d values gave us trouble, and with a d as small as 3, one cannot even hear the echo effect.

Code Listing 1: Our Python code that demonstrates echo generation and cancellation

```
import wave
import numpy as np
from matplotlib import pyplot as plt
from scipy.optimize import fsolve
from scipy.signal import correlate
```

```
#Our code does the following:
#1) Read the WAV file named Our Voice.wav
#2) With d = 0.5 seconds and alpha = 0.5, apply the echo filter
#3) Plot the original recording together with the echoed recording
#4) Plot the autocorrelation array of the echoed recoding
#5) Reconstruct the original recording from the echoed recording
#6) Save the echoed recording and the reconstructed recording

def SaveSignalLikeAnotherObj(signal, original_obj, filename):
    with wave.open(filename, "wb") as obj_new:
        obj_new.setnchannels(original_obj.getnchannels())
        obj_new.setsampwidth(original_obj.getsampwidth())
        obj_new.setframerate(obj.getframerate())
        obj_new.writeframes(signal.astype(np.int16).tobytes())

def ExtractSignalFromObj(obj):
    frames = obj.readframes(-1) #Read all frames
    return np.frombuffer(frames, dtype = np.int16).astype(np.double)

def PlotSignal(signal, framerate):
    #A general function to plot a signal
    #The framerate is needed for us to get a true time scale in seconds
    time = np.linspace(0, len(signal) / framerate, num = len(signal))
    plt.plot(time, signal)

def ApplyEchoFilter(x, d, alpha):
    #Using np.convolve ran a lot slower
    #Better just use the difference equation
    y = np.empty(len(x) + d)
    for n in range(0, d):
        y[n] = x[n]
    for n in range(d, len(x)):
         y[n] = x[n] + alpha * x[n-d]
    for n in range(len(x) + 1, len(y)):
        y[n] = alpha * x[n-d]
    return y

def RestoreD(y):
    #method = "fft" allows us to use the efficient computation algorithm
    #that we described in the PDF
```

```
    auto_correlation = correlate(y, y, mode = "full", method = "fft")
    #The definition in the documentation is a bit different from ours
    #We truncate the auto_correlation array
    auto_correlation = auto_correlation[len(y) - 1:]
    plt.title("Auto Correlation")
    plt.plot(auto_correlation)
    plt.show()
    #The algorithm is as described in the PDF
    prev = auto_correlation[0]
    k = 1
    for k in range(1, len(auto_correlation)):
        curr = auto_correlation[k]
        if curr > prev:
            break
        prev = curr
    return k + np.argmax(auto_correlation[k:])

def GetN(y, d):
    return len(y) - d

def GetM(N, d):
    return int(np.ceil(N / d))

def TargetFunction(y, d, m, alpha):
    #The root of this function is the alpha we need to restore
    sum = 0
    coeff = 1
    for j in range(0, m):
        coeff *= -alpha
        sum += y[(m - j) * d] * coeff
    return sum

def RestoreAlpha(y, d, N):
    m = GetM(N, d)
    initial_guess = 0.6
    #0.6 is an arbitrary choice of a number in (0,1)
    #It is the initial point of search for fsolve
    return fsolve(lambda alpha : TargetFunction(y, d, m, alpha), initial_guess)[0]


def RestoreX(y):
```

```
    d = RestoreD(y)
    print("Restored d =", d)
    N = GetN(y,d)
    alpha = RestoreAlpha(y, d, N)
    print("Restored alpha =", alpha)
    #After restoring d and alpha,
    #we can restore x by using the difference equation
    x = np.empty(N)
    for n in range(0, d):
        x[n] = y[n]
    for n in range(d + 1, N):
        x[n] = y[n] - alpha * x[n - d]
    return x

#Import the recording file
with wave.open("Our Voice.wav", "rb") as obj:
    framerate = obj.getframerate()
    x = ExtractSignalFromObj(obj)
    print("Length of x = ", len(x))
    d = 3 * obj.getframerate() #Echo with d = 3 seconds
    alpha = 0.5
    print("d = ", d)
    print("alpha =", alpha)
    y = ApplyEchoFilter(x, d, alpha)
    print("Length of y =", len(y))
    plt.subplot(121)
    plt.title("Original Recording")
    PlotSignal(x, framerate)
    plt.subplot(122)
    plt.title("Recording With Echo")
    PlotSignal(y, framerate)
    plt.show()
    x = RestoreX(y)
    SaveSignalLikeAnotherObj(y, obj, "Echoed Recording.wav")
    SaveSignalLikeAnotherObj(x, obj, "Restored Recording.wav")
```

Least mean squares for coefficient adjustment

1. If $y(n)$ and $d(n)$ have a common length of M, then we can consider them as elements in $\mathbb{R}^M$. Then, Λ exactly matches the definition of the distance squared between the two elements of that space (With the metric induced by the 2-norm). Since we know this is a metric, we can think of it also as an error measurement - the closer they are, the less error we have.

2. First, we need to realize what the length of $x(n)$ is. In general, when taking a convolution between two arrays $a(n), b(n)$ of lengths L_1, L_2, their convolution is of length $L_1 + L_2 - 1$. Indeed,

$$a(n) * b(n) = \sum_{k=-\infty}^{\infty} a(k)b(n-k) = \sum_{k=0}^{L_1-1} a(n)b(n-k) = a(0)b(n) + \ldots + a(L_1-1)b(n-L_1+1)$$

(here we apply the convention that arrays are indexed from 0 and the value is 0 for out-of-bounds indices. Shifting an array in time is thought of as zero padding). The last array $b(n-L_1+1)$ has maximal index $L_2 + L_1 - 2$, meaning overall $a(n) * b(n)$ is an array of length $L_1 + L_2 - 1$. This means that because $y(n)$ is of length M and $h(n)$ is of length N, the array $x(n)$ must be at least of length $M - N + 1$. Following this setup, we can redefine:

$$r_{xx}(k) := \sum_{n=0}^{M-N} x(n)x(n+k)$$

$$r_{dx}(k) := \sum_{n=0}^{M-1} d(n)x(n-k)$$

$$\Lambda := \sum_{n=0}^{M-1} \left(y(n) - d(n)\right)^2$$

Moving to the actual proof,

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(n-k)h(k) = \sum_{k=0}^{N-1} x(n-k)h(k)$$

$$\Lambda = \sum_{n=0}^{M-1} \left(y(n) - d(n)\right)^2 = \underbrace{\sum_{n=0}^{M-1} y(n)^2}_{S_1} - 2\underbrace{\sum_{n=0}^{M-1} y(n)d(n)}_{S_2} + \sum_{n=0}^{M-1} d(n)^2$$

$$S_1 = \sum_{n=0}^{M-1} y(n)^2 = \sum_{n=0}^{M-1}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} x(n-k)h(k)x(n-l)h(l) = \sum_{k=0}^{N-1}\sum_{l=0}^{N-1} h(k)h(l)\sum_{n=0}^{M-1} x(n-k)x(n-l)$$

Now, after making a change of index: $m = n - k$,

$$S_1 = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} h(k)h(l) \underbrace{\sum_{m=-k}^{M-k-1} x(m)x(m+k-l)}_{S_3}$$

Looking at S_3, since $k \leq N-1$ and $x(m) = 0$ for all m such that $m < 0$ or $m > M - N$, we realize that:

$$S_3 = \sum_{m=-k}^{M-k-1} x(m)x(m+k-l) = \sum_{m=0}^{M-N} x(m)x(m+k-l) = r_{xx}(k-l)$$

Substituting this into S_1, we are left with

$$S_1 = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} h(k)h(l)r_{xx}(k-l)$$

$$S_2 = \sum_{n=0}^{M-1} y(n)d(n) = \sum_{n=0}^{M-1} d(n) \sum_{k=0}^{N-1} x(n-k)h(k) = \sum_{k=0}^{N-1} h(k) \sum_{n=0}^{M-1} d(n)x(n-k) = \sum_{k=0}^{N-1} h(k)r_{dx}(k)$$

Looking at the expressions for S_1, S_2, it exactly proves that

$$\Lambda = \sum_{n=0}^{M} d(n)^2 - 2\sum_{k=0}^{N-1} h(k)r_{dx}(k) + \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} h(k)h(l)r_{xx}(k-l)$$

3. We again need to consider an adequate space for this proof. Instead of choosing l_2 and starting analytical discussions regarding completeness and Hilbert spaces, we restrict ourselves solely to the finite-dimensional space $\mathbb{R}^M$ with the standard inner product.

$$y(n) = x(n) * h(n) = \sum_{k=0}^{N-1} h(k)x(n-k) = h(0)x(n) + h(1)x(n-1) + \ldots + h(n-N+1)x(n-N+1)$$

so $y(n)$ is a linear combination of the signals $x(n), \ldots x(n-N+1)$, meaning that

$$y(n) \in \text{Span}\{x(n), \ldots, x(n-N+1)\} := W \subseteq \mathbb{R}^M.$$

In this setting, $\Lambda = ||y(n) - d(n)||^2$, where $d(n)$ is a given vector in $\mathbb{R}^M$. Recall the following Linear Algebra theorem:

Theorem. *Suppose V is a finite dimensional inner product space, and let $W \subseteq V$ be a subspace. Then,*

for each vector $v \in V$, there exists a unique vector w^* such that

$$dist(v, W) := inf\ \{||v - w|| : w \in W\} = ||v - w^*||$$

Moreover, w^* is the projection of v onto the subspace W.

Our problem setting exactly matches the conditions of the theorem above, with $V = \mathbb{R}^M$ and $v = d(n)$. This means that the filter is optimal if and only if $y(n) = \text{proj}_W d(n)$. As a projection onto a subspace, it must satisfy that for all $s(n) \in W$,

$$\langle d(n) - y(n), s(n) \rangle = 0 \iff \langle d(n), s(n) \rangle = \langle y(n), s(n) \rangle$$

In particular, looking at the definition of W, for all $m = 0, \dots N-1$, $x(n-m) \in W$, so

$$\langle d(n), x(n-m) \rangle = \langle y(n), x(n-m) \rangle ,$$

with the left side being:

$$\langle d(n), x(n-m) \rangle = \sum_{n=0}^{M-1} d(n)x(n-m) = r_{dx}(m),$$

and the right being:

$$\langle y(n), x(n-m) \rangle = \sum_{n=0}^{M-1} y(n)x(n-m) = \sum_{n=0}^{M-1} \sum_{k=0}^{N-1} x(n-m)h(k)x(n-k) = \sum_{k=0}^{N-1} h(k) r_{xx}(k-m)$$

where the explanation of the last equality is similar to what we did when simplifying S_3 in the previous question.

4. To find the optimal filter, we solve an optimization problem. We can think of Λ as a function

$$\Lambda : \mathbb{R}^N \to \mathbb{R}, \qquad \Lambda\left(h_0, \dots h_{N-1}\right) = \sum_{n=0}^{M-1} \left(d(n) - y(n)\right)^2$$

LMS is basically performing gradient descent on this function. Now we derive a formula for its gradient:

$$\frac{\partial \Lambda}{\partial h_k} = \frac{\partial}{\partial h_k} \sum_{n=0}^{M-1} \left(d(n) - y(n)\right)^2 = \sum_{n=0}^{M-1} \frac{\partial}{\partial h_k} \left(d(n) - y(n)\right)^2 = -\sum_{n=0}^{M-1} 2 \underbrace{\left(d(n) - y(n)\right)}_{e(n)} \frac{\partial y(n)}{\partial h_k}$$

recall that

$$y(n) = x(n) * h(n) = h(0)x(0) + h(1)x(n-1) + \ldots + h(n-N+1)h(N-1) \Longrightarrow \frac{\partial y(n)}{\partial h_k} = x(n-k)$$

$$\Longrightarrow \frac{\partial \Lambda}{\partial h_k} = -2 \sum_{n=0}^{M-1} e(n)x(n-k)$$

We can get rid of the constant factor of 2 when doing the iterations (it just gets absorbed by Δ). Denote $h^{(s)}$ as the impulse response in iteration number s. We set $h^{(0)} = 0$, and perform the iterations:

$$h^{(s)}(k) = h^{(s-1)}(k) + \Delta \sum_{n=0}^{M-1} e(n)x(n-k)$$

Moreover, the sum $\sum_{n=0}^{M-1} e(n)x(n-k)$ can be realized as the correlation between $e(n)$ and $x(n)$ at point k. As we have seen in question 1, the correlation array can be efficiently computed using the FFT method. So in the implementation we actually perform:

$$h^{(s)} = h^{(s-1)} + \Delta r_{ex}$$

where r_{ex} is the correlation array between $e(n)$ and $x(n)$ (truncated so that we take N elements).

Code Listing 2: Our implementation of the adaptive LMS-algorithm

```
import numpy as np
from scipy.signal import correlate
from scipy.signal import convolve
from scipy.optimize import minimize
from numpy.linalg import norm

def ApplyFilter(x, h):
    #scipy.signal.convolve might use FFT method if it is faster
    #than the direct method
    return convolve(x, h)

def FixBounds(y, d):
    if len(y) < len(d):
        temp = np.zeros(y)
        temp[:len(y)] = y
        y = temp
    elif len(y) > len(d):
        y = y[:len(d)]
```

```
    return y

def FilterWithBounds(x, h, d):
    y = ApplyFilter(x, h)
    return FixBounds(y, d)

#This function is for comparison
#Both functions output similar results
def OptimizeUsingLibraryFunction(x, d, N):
    h0 = np.zeros(N)
    h = minimize(lambda h: norm(d - FilterWithBounds(x, h, d)) ** 2, h0).x
    return h, FilterWithBounds(x, h, d)

def LMS(x, d, N, delta = 0.001, number_of_iterations = 1000):
    #Performing gradient descent
    h = np.zeros(N)
    for i in range(number_of_iterations):
        e = d - FilterWithBounds(x, h, d)
        step_direction = correlate(e, x, mode = "full", method = "auto")
        pivot = max(len(e), len(x)) - 1
        step_direction = step_direction[pivot:]
        step_direction = step_direction[:N]
        h += delta * step_direction
    return h, FilterWithBounds(x, h, d)
```

5. We chose a different signal: $s = \sin(t)$ on the interval $[0, 4\pi]$

Code Listing 3: Our Python code that demonstrates noise cancellation

```
from Adaptive import LMS
import numpy as np
from matplotlib import pyplot as plt
from numpy.random import normal
from scipy.fft import fft
from scipy.signal import tf2zpk as ZTransform
from scipy.signal import group_delay as GroupDelay
from scipy.signal import freqz as FrequencyResponse

def RightShift(x, D):
    shifted_x = np.zeros(len(x) + D)
    shifted_x[D:] = x
    return shifted_x
```

```
def CancelNoise(x, N, D = 1):
    shifted_x = RightShift(x, D)
    return LMS(shifted_x, x, N, delta = 0.000001)

M = 1000
t = np.linspace(0, 4 * np.pi, num = M)
s = np.sin(t)
x = s + normal(loc = 0, scale = 0.2, size = M)
h, filtered_signal = CancelNoise(x, N = 11)
plt.subplot(131)
plt.title("Original With Noisy")
plt.plot(x, color = "darkblue")
plt.plot(s, color = "red")
plt.subplot(132)
plt.title("Noisy With Filtered")
plt.plot(x, color = "darkblue")
plt.plot(filtered_signal, color = "deepskyblue")
plt.subplot(133)
plt.title("Original With Filtered")
plt.plot(filtered_signal, color = "deepskyblue")
plt.plot(s, color = "red")
plt.show()
denominator = np.zeros_like(h)
denominator[0] = 1
_, group_delay = GroupDelay((h, denominator))
w, phase_response = FrequencyResponse(h, denominator)
plt.subplot(121)
plt.xlabel("Frequency (rad)")
plt.title("Group Delay")
plt.plot(w, group_delay)
plt.subplot(122)
plt.xlabel("Frequency (rad)")
plt.title("Phase Response")
plt.plot(w, np.angle(phase_response))
plt.show()
u = np.linspace(0, 1, num = 200)
frequency_spectrum = np.abs(fft(h, 200))
plt.title("Magnitude Specturm")
plt.plot(u, frequency_spectrum)
plt.show()
```

```
zeros, _, _ = ZTransform(h, denominator)
print("The zeros of the Z transform are:", zeros)
```

Figure 6: Our noise cancellation results

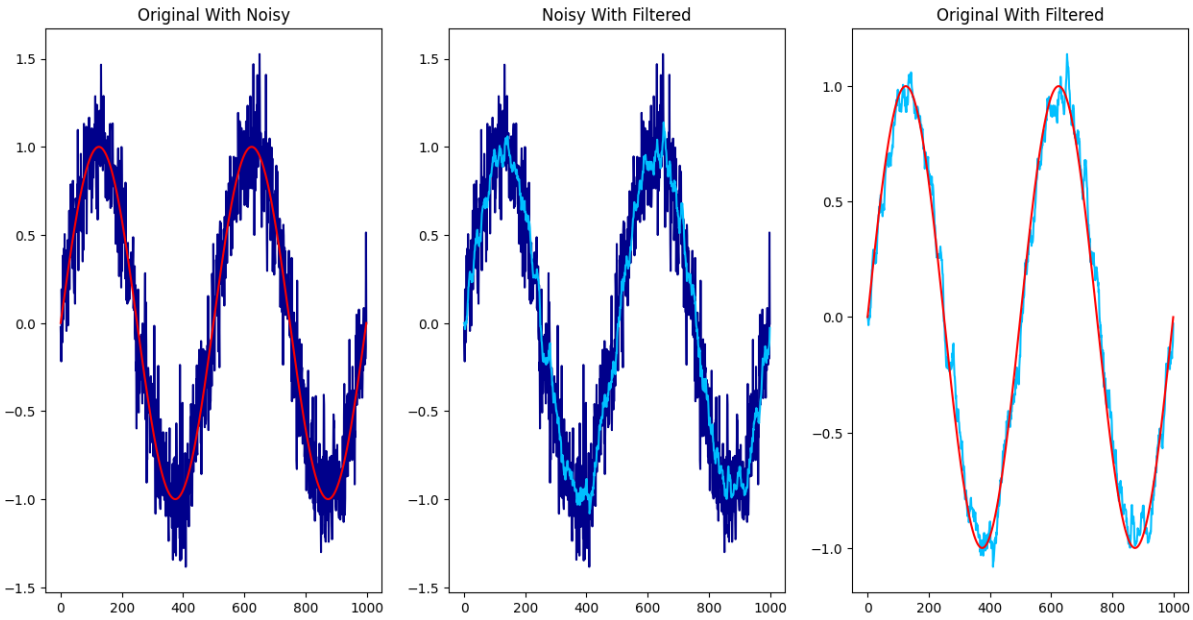
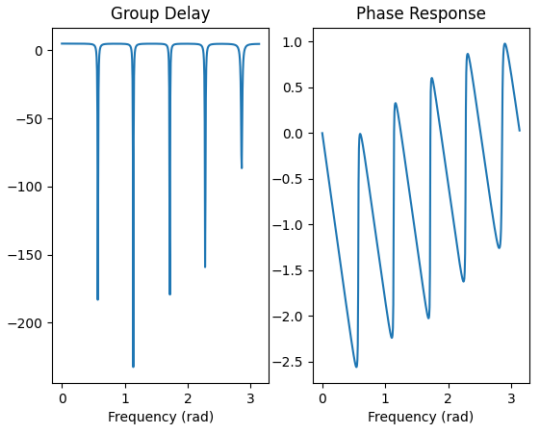
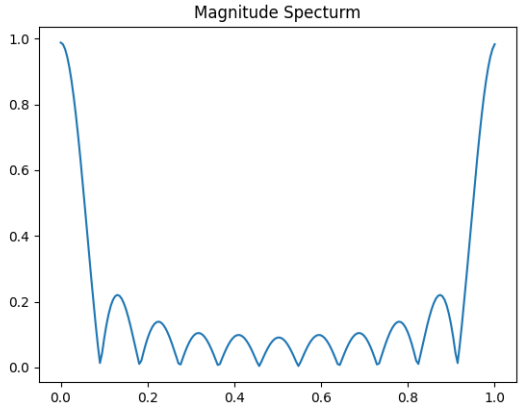


Figure 7: The phase of the frequency response and the group delay



As we can see, the group delay is not constant and the phase of the frequency response is not linear.

Figure 8: A graph of the magnitude spectrum (200 samples)



The filter is evidently low-pass. It also makes sense, since the noisy signal seems more oscillatory than the original signal, so removing the noise corresponds to weakening high frequencies. The code prints out the zeros of the Z-transform.