Worked Examples and Solutions for the Book: Adaptive Filtering Primer with MATLAB by Alexander Poularikas and Zayed Ramadan

John L. Weatherwax*

December 13, 2015

*wax@alum.mit.edu

Introduction

This is a wonderful little book on adaptive filtering. I found the examples enjoyable and the text very easy to understand. To better facilitate my understanding of this material I wrote some notes on the main text and worked a great number of the problems while I worked through the book. For some of the problems I used `MATLAB` to perform any needed calculations. The code snippets for various exercises can be found at the following location:

`http://waxworksmath.com/Authors/N_Z/Poularikas/poularikas.html`

I've worked hard to make these notes as good as I can, but I have no illusions that they are perfect. If you feel that that there is a better way to accomplish or explain an exercise or derivation presented in these notes; or that one or more of the explanations is unclear, incomplete, or misleading, please tell me. If you find an error of any kind – technical, grammatical, typographical, whatever – please tell me that, too. I'll gladly add to the acknowledgments in later printings the name of the first person to bring each problem to my attention.

Chapter 2 (Discrete-time signal processing)

Problem 2.2.1 (a comparison between the FT and the DTFT)

The Fourier transform of the given signal $x(t)$ is given by

$$\begin{aligned}
X(\omega) &= \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt \\
&= \int_{-\infty}^{0} e^{-|t|}e^{-j\omega t}dt + \int_{0}^{\infty} e^{-|t|}e^{-j\omega t}dt \\
&= \int_{-\infty}^{0} e^{t}e^{-j\omega t}dt + \int_{0}^{\infty} e^{-t}e^{-j\omega t}dt \\
&= \int_{0}^{\infty} e^{-t}e^{j\omega t}dt + \int_{0}^{\infty} e^{-t}e^{-j\omega t}dt \\
&= \int_{0}^{\infty} e^{-(1-j\omega)t}dt + \int_{0}^{\infty} e^{-(1+j\omega)t}dt \\
&= \frac{e^{-(1-j\omega)t}}{-(1-j\omega)}\Big|_{0}^{\infty} + \frac{e^{-(1+j\omega)t}}{-(1+j\omega)}\Big|_{0}^{\infty} \\
&= \frac{1}{1-j\omega} + \frac{1}{1+j\omega} \\
&= \frac{1+j\omega}{1+\omega^2} + \frac{1-j\omega}{1+\omega^2} \\
&= \frac{2}{1+\omega^2}\,.
\end{aligned}$$

Evaluating this expression at $\omega = 1.6$ rad/s gives 0.5618.

Next, we evaluate the discrete time Fourier transform (DTFT) of $x(t) = e^{-|t|}$ with two different sampling intervals $T = 1$s and $T = 0.1$s. We begin by computing the DTFT of $x(t)$ as

$$\begin{aligned}
X(e^{j\omega T}) &= T\sum_{n=-\infty}^{\infty} x(nT)e^{-j\omega nT} \\
&= T\left(\sum_{n=-\infty}^{-1} e^{nT}e^{-j\omega nT} + 1 + \sum_{n=1}^{\infty} e^{-nT}e^{-j\omega nT}\right) \\
&= T\left(\sum_{n=1}^{\infty} e^{-nT}e^{j\omega nT} + \sum_{n=0}^{\infty} e^{-nT}e^{-j\omega nT}\right)
\end{aligned}$$

$$
\begin{aligned}
&= T\left(\sum_{n=0}^{\infty}(e^{-T}e^{j\omega T})^n - 1 + \sum_{n=0}^{\infty}(e^{-T}e^{-j\omega T})^n\right) \\
&= T\left(\frac{1}{1-e^{-T}e^{j\omega T}} - 1 + \frac{1}{1-e^{-T}e^{-j\omega T}}\right) .
\end{aligned}
$$

To convert this expression into its real and imaginary parts we will multiply each fraction above by a conjugate of its denominator. Denoting this common product as D (for denominator) we have

$$
\begin{aligned}
D &= (1-e^{-T}e^{j\omega T})(1-e^{-T}e^{-j\omega T}) \\
&= 1 - e^{-T}e^{-j\omega T} - e^{-T}e^{j\omega T} + e^{-2T} \\
&= 1 - e^{-T}(e^{-j\omega T} + e^{j\omega T}) + e^{-2T} \\
&= 1 - 2e^{-T}\cos(\omega T) + e^{-2T} .
\end{aligned}
$$

So that the expression for the DTFT $X(e^{j\omega T})$ becomes

$$
\begin{aligned}
X(e^{j\omega T}) &= T\left(\frac{1-e^{-T}e^{-j\omega T}}{D} + \frac{1-e^{-T}e^{j\omega T}}{D} - 1\right) \\
&= \frac{T}{D}(2 - 2e^{-T}\cos(\omega T) - D) \\
&= \frac{T}{D}(1-e^{-2T}) \\
&= \frac{T(1-e^{-2T})}{1-2e^{-T}\cos(\omega T) + e^{-2T}} .
\end{aligned}
$$

When $T = 1$s and $\omega = 1.6$ rad/s we find $X(e^{j(1.6)}) = 0.7475$. When $T = 0.1$s this expression becomes $X(e^{j(0.16)}) = 0.5635$. The sampling interval with $T = 0.1$s is obviously a better approximation to the full Fourier transform at this point.

Problem 2.2.2 (an example with the DFT)

See the MATLAB file `prob_2_2_2.m` for the calculations required for this problem. They basically follow the discussion in the book in the section entitled "the discrete Fourier transform (DFT)". My results don't exactly match the answer presented at the end of the chapter but I don't see anything incorrect with what I've done.

Problem 2.3.1 (examples with the z-transform)

For this problem using the definition of the z-transform we will directly compute each of the requested expressions.
Part (a): In this case we have

$$\begin{aligned} X(z) &= Z\{x(n)\} = \sum_{n=-\infty}^{\infty} x(n)z^{-n} \\ &= \sum_{n=-\infty}^{\infty} \cos(n\omega T)u(n)z^{-n} = \sum_{n=0}^{\infty} \cos(n\omega T)z^{-n} \\ &= \frac{1}{2}\sum_{n=0}^{\infty}(e^{jn\omega T} + e^{-jn\omega T})z^{-n} \\ &= \frac{1}{2}\left(\sum_{n=0}^{\infty}(e^{j\omega T}z^{-1})^n + \sum_{n=0}^{\infty}(e^{-j\omega T}z^{-1})^n\right) \\ &= \frac{1}{2}\left(\frac{1}{1-e^{j\omega T}z^{-1}} + \frac{1}{1-e^{-j\omega T}z^{-1}}\right). \end{aligned}$$

To further simplify this expression in each fraction above we will multiply by a "form of one" determined by the conjugate of the fractions denominator. In both cases this gives a denominator D given by

$$\begin{aligned} D &= (1-e^{j\omega T}z^{-1})(1-e^{-j\omega T}z^{-1}) \\ &= 1 - e^{j\omega T}z^{-1} - e^{-j\omega T}z^{-1} + z^{-2} \\ &= 1 - 2z^{-1}\cos(\omega T) + z^{-2}. \end{aligned}$$

With this expression we obtain

$$\begin{aligned} X(z) &= \frac{1}{2D}(2 - 2z^{-1}\cos(\omega T)) \\ &= \frac{z^2 - z\cos(\omega T)}{z^2 - z\cos(\omega T) + 1}. \end{aligned}$$

Part (b): In this case we find

$$X(z) = Z\{x(n)\} = \sum_{n=-\infty}^{\infty} na^n u(n)z^{-n} = \sum_{n=0}^{\infty} na^n z^{-n} = \sum_{n=0}^{\infty} n\left(\frac{a}{z}\right)^n.$$

From the identity

$$\frac{d}{d\left(\frac{a}{z}\right)}\left(\frac{a}{z}\right)^n = n\left(\frac{a}{z}\right)^{n-1},$$

we have

$$\left(\frac{a}{z}\right)\left(\frac{d}{d\left(\frac{a}{z}\right)}\left(\frac{a}{z}\right)^n\right) = n\left(\frac{a}{z}\right)^n,$$

and the above becomes

$$\begin{aligned} X(z) &= \left(\frac{a}{z}\right)\frac{d}{d\left(\frac{a}{z}\right)}\sum_{n=0}^{\infty}\left(\frac{a}{z}\right)^n \\ &= \left(\frac{a}{z}\right)\frac{d}{d\left(\frac{a}{z}\right)}\left(\frac{1}{1-\left(\frac{a}{z}\right)}\right) \\ &= \left(\frac{a}{z}\right)\frac{1}{\left(1-\frac{a}{z}\right)} \\ &= \frac{az}{(a-z)^2}. \end{aligned}$$

Chapter 3 (Random variables, sequences, and stochastic processes)

Problem 3.2.1 (an example of the autocorrelation function)

The autocorrelation function for this random process $x(n)$ is defined as

$$\begin{aligned} r_x(n,m) &= E[x(n)x(m)] \\ &= E[(a\cos(n\omega+\theta))(a\cos(m\omega+\theta))] \\ &= a^2\int_{-\pi}^{\pi} cos(n\omega+\theta)\cos(m\omega+\theta)\left(\frac{1}{2\pi}\right)d\theta\,. \end{aligned}$$

Using the fact that

$$\cos(\alpha)\cos(\beta) = \frac{1}{2}(\cos(\alpha+\beta)+\cos(\alpha-\beta))\,,$$

the above becomes

$$r_x(n,m) = \frac{a^2}{4\pi}\int_{-\pi}^{\pi}(\cos((n-m)\omega)+\cos((n+m)\omega+2\theta))d\theta\,.$$

Now

$$\int_{-\pi}^{\pi}\cos((n+m)\omega+2\theta)d\theta = \left.\frac{\sin((n+m)\omega+2\theta)}{2}\right|_{-\pi}^{\pi} = 0\,.$$

Using this the above then gives for the autocorrelation the following

$$r_x(n,m) = \frac{a^2}{4\pi}\cos((n-m)\omega)(2\pi) = \frac{a^2}{2}\cos((n-m)\omega)\,.$$

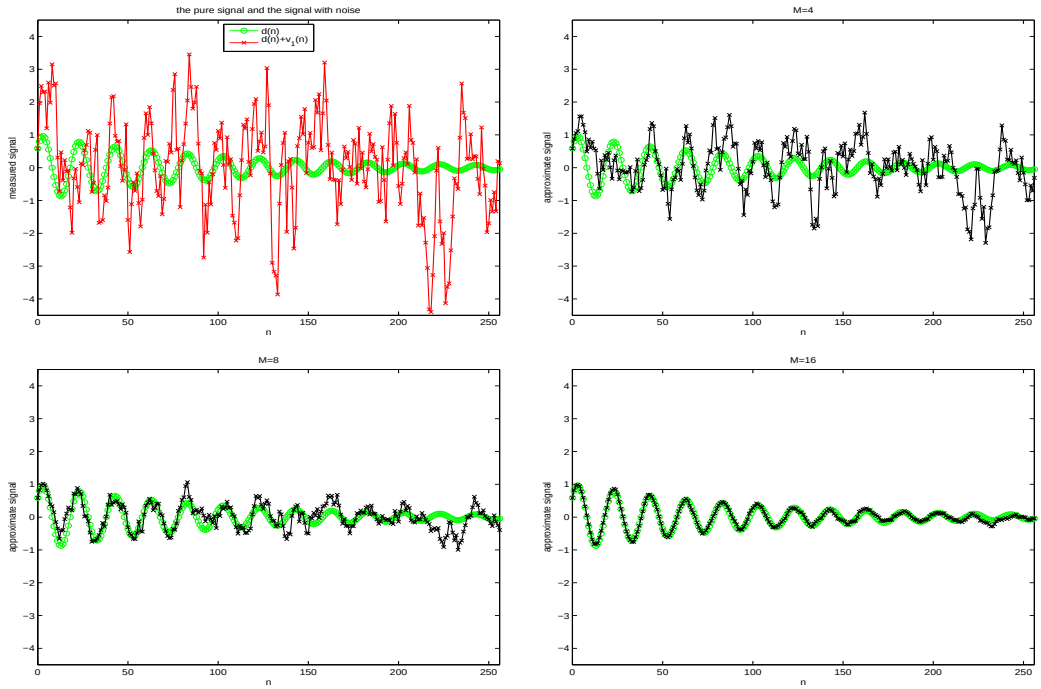

Figure 1: **Upper Left:** The pure signal $d(n)$ in green and its noised counterpart $d(n) + v_1(n)$ in red. **Upper Right:** The Wiener filtered results using $M = 4$ filter coefficients in black plotted with the signal $d(n)$ in green. **Lower Left:** Using $M = 8$ filter coefficients. **Lower Right:** Using $M = 16$ filter coefficients.

Chapter 4 (Wiener filters)

Notes on the Text

To aid in my understanding of this chapter I choose to duplicate the results from the book example that deals with the use of the Wiener filter to perform noise cancellation (Example 4.4.4). To do this I added several comments and fixed a few small bugs in the MATLAB function `aawienernoisecancelor.m` supplied with the book . I then created a script `noise_canceling_script.m` that performs the statements suggested in this section of the text. When this is run it first plots the true signal $d(n)$ along with the noised version $d(n) + v_1(n)$. Then for three choices of filter lengths $M = 4, 8, 16$ the estimated

signal $\hat{d}(n)$ is derived from the noised signal $x(n)$ by removing an estimate of the noise $\hat{v}_1(n)$. All of these results are shown in Figure 1.

Notice that when the number of filter coefficients becomes large enough ($M \geq 16$) we are able to reconstruct the desired signal $d(n)$ quite well given the amount of noise that is present.

Problem Solutions

Problem 4.3.1 (calculation of the minimum error)

We begin this problem by recalling that the optimum filter weights w^o are given by

$$w^o = R_x^{-1} p_{dx} \,. \tag{1}$$

Here R_x is the autocorrelation of the process $x(n)$ and p_{dx} is the cross-correlation vector between $d(n)$ and $x(n)$. We also recall the quadratic cost $J(w)$ we sought to minimize is defined as

$$J(w) = E\{(d(n) - \hat{d}(n))^2\} = E\{e^2(n)\} \,.$$

When represented in terms of the statistics of our desired signal $d(n)$ and our observed signal $x(n)$ becomes

$$J(w) = \sigma_d^2 - 2w^T p_{dx} + w^T R_x w \,.$$

When this expression is evaluated at the optimum w^o it becomes

$$\begin{aligned} J(w^o) &= \sigma_d^2 - 2p_{dx}^T R_x^{-1} p_{dx} + p_{dx}^T R_x^{-1} R_x R_x^{-1} p_{dx} \\ &= \sigma_d^2 - p_{dx}^T R_x^{-1} p_{dx} \qquad (2) \\ &= \sigma_d^2 - p_{dx}^T w^o \,. \qquad (3) \end{aligned}$$

In the above we have used the fact that the autocorrelation matrix, R_x, is symmetric.

Problem 4.3.2 (modeling an unknown system)

This problem can be worked numerically using code similar to that found in Example 4.3.1 of the book. See the MATLAB script `prob_4_3_2.m` for an example of how this is done. Alternatively, one can compute many of the required correlations analytically due to the problem specification and

solve this problem directly. First, since we are told that our input data $\{x(n)\}$ is a stationary white process with zero mean and unit variance we conclude that its autocorrelation matrix R_x is the identity matrix. Next, the cross-correlation vector, $p_{xd}(m)$, can be computed from its definition as

$$\begin{aligned} p_{xd}(m) &= E\{x(n)d(n-m)\} \\ &= E\{x(n)(b_0x(n-m)+b_1x(n-m-1)+v(n-m))\} \\ &= b_0E\{x(n)x(n-m)\}+b_1E\{x(n)x(n-m-1)\} \\ &= b_0r_x(m)+b_1r_x(m+1)\,. \end{aligned}$$

Assuming a filter with only two values for m i.e. $M=2$ so that $m=0,1$ from the above see that $p_{xd}(0)=b_0$ and $p_{xd}(1)=b_1$, so that as a vector $p_{xd}=\begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$. Now σ_d^2 is given by

$$\begin{aligned} \sigma_d^2 &= E\{d^2(n)\} = E\{(b_0x(n)+b_1x(n-1)+v(n))^2\} \\ &= b_0^2E\{x^2(n)\}+b_1^2E\{x^2(n-1)\}+E\{v^2(n)\} \\ &+ 2b_0E\{x(n)v(n)\}+2b_1E\{x(n-1)x(n)\}+2b_0b_1E\{x(n)x(n-1)\} \\ &= b_0^2\sigma_x^2+b_1^2\sigma_x^2+\sigma_v^2\,. \end{aligned}$$

With everything from above solving the discrete Wiener-Hopf equation gives

$$w^o = R_x^{-1}p_{xd} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}\,.$$

So that we find a minimum cost $J_{\min}$ given by Equation 3 which in this case specifies to

$$J_{\min} = \sigma_d^2 - p_{dx}^T w^o = (b_0^2+b_1^2+\sigma_v^2)-(b_0^2+b_1^2) = \sigma_v^2 = 0.15\,.$$

These results match quite nicely those found when this problem is worked numerically in `prob_4_3_2.m` when we take the number of time samples, N, large enough (to be confident in our statistics). Runs where $N > 200$ seemed to work well.

Problem 4.3.3 ($J_{\min}$ with the orthogonality principle)

The definition of $J_{\min}$ is given by

$$J_{\min} = E\{e^o(n)e^o(n)\}\,,$$

where the optimal error $e^o(n)$ is given by the difference between the desired signal, $d(n)$, and the estimated signal $\hat{d}(n) = \sum_{m=0}^{M-1} w_m x(n-m)$ as

$$e^o(n) = d(n) - \sum_{m=0}^{M-1} w_m x(n-m)\,.$$

Here w_m are the *optimal* weights and the superscript o above stands for "optimal". We can now compute the product of this expression directly with $e^o(n)$ "unexpanded" as

$$e^o(n)e^o(n) = e^o(n)d(n) - \sum_{m=0}^{M-1} w_m e^o(n)x(n-m)\,.$$

Taking the expectation of both sides of the above and using the orthogonality condition of $E\{e^o(n)x(n-m)\} = 0$ for $m = 0, 1, \cdots, M-1$ we find the expectation of the second term vanish and we are left with

$$\begin{aligned} J_{\min} &= E\{e^o(n)d(n)\} \\ &= E\{\left(d(n) - \sum_{m=0}^{M-1} w_m x(n-m)\right) d(n)\} \\ &= E\{d(n)^2 - \sum_{m=0}^{M-1} w_m x(n-m)d(n)\} \\ &= \sigma_d^2 - \sum_{m=0}^{M-1} w_m E\{x(n-m)d(n)\}\,. \end{aligned}$$

By definition, this last expectation $E\{x(n-m)d(n)\}$ is the cross-correlation between $d(n)$ and $x(n)$ or $p_{dx}(m)$ and the above becomes

$$J_{\min} = \sigma_d^2 - \sum_{m=0}^{M-1} w_m p_{dx}(m)\,,$$

the same result as Equation 3 but in terms of the components of the vectors w^o and p_{dx}.

Problem 4.4.1 (a specific Wiener filter)

For this problem we are told the autocorrelation function for the signal and the noise are given by the quoted expressions for $r_d(m)$ and $r_v(m)$ respectively. Using these the cross-correlation function, $p_{dx}(m)$, can be computed

from its definition as

$$\begin{aligned} p_{dx}(m) &= E\{d(n)x(n-m)\} \\ &= E\{d(n)(d(n-m)+v(n-m))\} \\ &= E\{d(n)d(n-m)\} + E\{d(n)v(n-m)\} \\ &= r_d(m)\,. \end{aligned}$$

Where we have used the fact that the term $E\{d(n)v(n-m)\} = 0$ since $E\{v(n)\} = 0$ and the processes $d(n)$ and $v(n)$ are uncorrelated. Recall that the optimal Wiener filtering weights w^o are given by $w^o = R_x^{-1}p_{dx}$. To compute this expression we next need to compute the autocorrelation matrix R_x. This is a Toeplitz matrix that has its (i,j) elements when $|i-j| = m$ given by $r_x(m)$. Here $r_x(m)$ is computed as

$$\begin{aligned} r_x(m) &= E\{x(n)x(n-m)\} \\ &= E\{(d(n)+v(n))(d(n-m)+v(n-m))\} \\ &= E\{d(n)d(n-m) + d(n)v(n-m) + v(n)d(n-m) + v(n)v(n-m)\} \\ &= r_d(m) + r_v(m)\,. \end{aligned}$$

With the specified functional forms for $r_d(m)$ and $r_v(m)$ quoted in this problem the autocorrelation matrix R_x looks like (assuming the length of the filter M, is 4)

$$\begin{aligned} R_x &= \begin{bmatrix} r_x(0) & r_x(1) & r_x(2) & r_x(3) \\ r_x(1) & r_x(0) & r_x(1) & r_x(2) \\ r_x(2) & r_x(1) & r_x(0) & r_x(1) \\ r_x(3) & r_x(2) & r_x(1) & r_x(0) \end{bmatrix} = \begin{bmatrix} 2 & 0.9 & 0.9^2 & 0.9^3 \\ 0.9 & 2 & 0.9 & 0.9^2 \\ 0.9^2 & 0.9 & 2 & 0.9 \\ 0.9^3 & 0.9^2 & 0.9 & 2 \end{bmatrix} \\ &= \begin{bmatrix} 2.00 & 0.90 & 0.81 & 0.72 \\ 0.90 & 2.00 & 0.90 & 0.81 \\ 0.81 & 0.90 & 2.00 & 0.90 \\ 0.72 & 0.81 & 0.90 & 2.00 \end{bmatrix}. \end{aligned}$$

Since in practice we don't know the optimal filter length M to use we let $M = 2, 3, \cdots$, compute the optimal filter weights w^o, using the Wiener-Hopf Equation 1, and for each evaluate the resulting $J_{\min}$ using Equation 3. One then takes M to be the first value where where the $J_{\min}$ first falls below a fixed threshold, say 0.01. If we specify $M = 2$ the optimal weights w^o and minimum error $J_{\min}$ are found in the MATLAB script `prob_4_4_1.m`. Running this gives numerical results identical to those found in the book.

We now compute the SNR of the original signal $d(n)$ against the noise signal $v(n)$. Using the definition that the power a signal $d(n)$ is given by $E\{d(n)^2\}$ see that

$$\begin{aligned} \text{Power in the signal} &= E\{d^2(n)\} = r_d(0) = 1 \\ \text{Power in the noise} &= E\{v^2(n)\} = r_v(0) = 1\,. \end{aligned}$$

Since they have equal power, the SNR before filtering is then

$$\text{SNR} = 10\log_{10}(\frac{1}{1}) = 0\,.$$

Note: I don't see any errors in my logic for computing the power in the filtered signals below, but my results do not match the book exactly. If anyone sees anything wrong with what I have here please let me know. Since these topics are not discussed much in the book I'm going to pass on this for now.

After filtering our observed signal $x(n)$, to obtain $\hat{d}(n)$, we would like to compute the power of the filtered signal $\hat{d}(n)$. This can be done by calculating $E\{\hat{d}^2(n)\}$. We find

$$\begin{aligned} E\{\hat{d}^2(n)\} &= E\{(w^T x)^2\} = E\{(w^T x)(w^T x)^T\} = E\{w^T x x^T w^T\} \\ &= w^T E\{xx^T\} w = w^T R_x w\,. \end{aligned}$$

An estimate of the noise $\hat{v}(n)$ is given by subtracting the estimated signal $\hat{d}(n)$ from our measured signal $x(n)$ i.e.

$$\hat{v}(n) = x(n) - \hat{d}(n)\,.$$

Thus the power in the estimated noise is given by $E\{\hat{v}^2(n)\}$. We can compute this as follows

$$\begin{aligned} E\{\hat{v}^2(n)\} &= E\{(x(n) - \hat{d}(n))^2\} = E\{x^2(n) - 2x(n)\hat{d}(n) + \hat{d}^2(n)\} \\ &= r_x(0) - 2E\{x(n)\hat{d}(n)\} + r_{\hat{d}}(0)\,. \end{aligned}$$

Now $\hat{d}(n) = w^T x = \sum_{m=0}^{M-1} w_m x(n-m)$ so the middle expectation above becomes

$$E\{x(n)\hat{d}(n)\} = \sum_{m=0}^{M-1} w_m E\{x(n)x(n-m)\} = \sum_{m=0}^{M-1} w_m r_x(m) = w^T r_x\,.$$

Thus we find

$$E\{\hat{v}^2(n)\} = r_x(0) - 2w^T r_x + r_{\hat{d}}(0)\,.$$

Again, these results may be different than what the book has.

Problem 4.4.2 (signal leaking into our noise)

The optimal Wiener filter coefficients w in this case is given by solving

$$R_y w^o = p_{v_1 y} \,,$$

where we have defined our signal input, $y(n)$, to be the additive noise $v_2(n)$ plus some amount, α, of our desired signal $d(n)$. That is

$$y(n) = v_2(n) + \alpha d(n) \,.$$

Once these Wiener coefficients w^o are found, they will be used to construct an estimate of $v_1(n)$ given the signal $y(n)$. For this signal $y(n)$ the autocorrelation matrix R_y has elements given by values from its autocorrelation function, $r_y(m)$. Since in operation we will be measuring the signal $y(n)$ we can compute its autocorrelation matrix using samples from our process. If, however, we desire to see how this autocorrelation matrix R_y depends on its component parts $v_2(n)$ and $d(n)$ we can decompose $r_y(m)$ as follows.

$$\begin{aligned} r_y(m) &= E\{y(n)y(n-m)\} \\ &= E\{(v_2(n) + \alpha d(n))(v_2(n-m) + \alpha d(n-m))\} \\ &= E\{v_2(n)v_2(n-m)\} + \alpha^2 E\{d(n)d(n-m)\} \,, \end{aligned}$$

since $E\{v_2(n)d(n-m)\} = 0$. This shows that the autocorrelation matrix for y is related to that of v_2 and d as

$$R_y = R_{v_2} + \alpha^2 R_d \,.$$

It should be noted that in an implementation we don't have access to $d(n)$ and thus cannot form R_y using this decomposition. Instead we have to estimate R_y using the input samples of $y(n)$. After R_y to complete the Wiener filter we need to compute the components of the cross-correlation vector $p_{v_1 y}(m)$. As any realizable implementation of this filter will need to estimate this cross-correlation using the two signals $y(n)$ and $x(n)$. We can decompose the cross-correlation vector $p_{v_1 y}$ as follows

$$\begin{aligned} p_{v_1 y}(m) &= E\{v_1(n)y(n-m)\} \\ &= E\{(x(n) - d(n))y(n-m)\} \\ &= E\{x(n)y(n-m)\} - E\{d(n)y(n-m)\} \\ &= E\{x(n)y(n-m)\} - E\{d(n)(v_2(n-m) + \alpha d(n-m))\} \\ &= E\{x(n)y(n-m)\} - \alpha E\{d(n)d(n-m))\} \,. \end{aligned}$$

The term (not shown) that would have had the product $d(n)v_2(n-m)$ vanished since $v_2(n)$ and $d(n)$ are uncorrelated and the process $v_1(n)$ has zero mean. Since we don't know the function $d(n)$[1] we cannot calculate $E\{d(n)d(n-m)\}$ using discrete samples. I see two ways to proceed with this textbook exercise. The first is to assume we have access to the statistics of $d(n)$ i.e. to the expectation above. We could obtain this information by "training" on a pre-specified set of $d(n)$ signals before the actual filters implementation. The second interpretation of this exercise would be to ignore the term $\alpha E\{d(n)d(n-m)\}$ and show how much the performance of a noise canceling algorithm will suffer from the fact that we are running it without the correct system model. In that case we would expect that when α is small the error will be less since then the dropped term $\alpha E\{d(n)d(n-m)\}$ may be negligible. That is, we could approximate p_{v_1y} with

$$p_{v_1y} \approx p_{xy}\,,$$

in our Wiener filter implementation above.

To finish this problem I'll assume that we somehow have access to the expectation $E\{d(n)d(n-m)\}$ before running this filter live. If anyone sees a way to compute the optimal filter coefficients w^o directly from the the given signals $x(n)$ and $y(n)$ please let me know.

From all of this discussion then our optimal filtering weights w^o are given by solving the Wiener-Hopf equation given by

$$R_y w^o = p_{xy} - \alpha p_{dd}\,. \tag{4}$$

In Example 4.4.4 we are told that $d(n)$, $v_1(n)$, and $v_2(n)$ have analytic expressions given by

$$\begin{aligned} d(n) &= 0.99^n \sin(0.1n\pi + 0.2\pi) \\ v_1(n) &= 0.8v_1(n-1) + v(n) \\ v_2(n) &= -0.95v_2(n-1) + v(n)\,. \end{aligned}$$

Here $v(n)$ a driving white noise process (a zero mean and unit variance Gaussian process). We then implement a modification of the book MATLAB function `aawienernoisecancelor.m` here denoted `aaWNC_with_leaking_signal.m`

[1]If we knew $d(n)$ we would have the perfect noise canceler already!

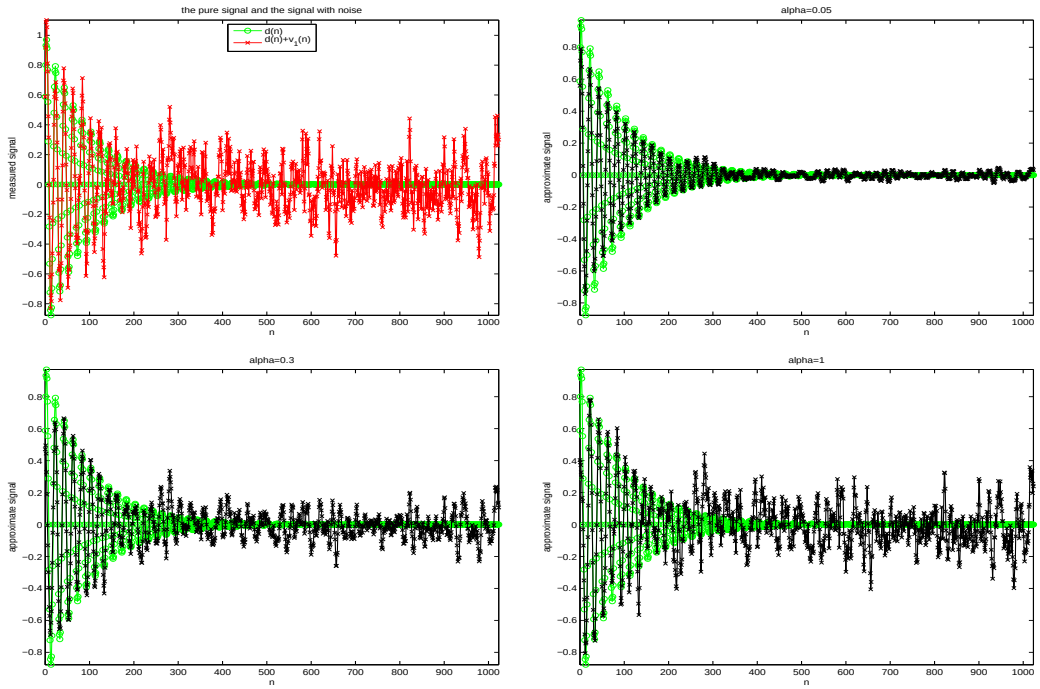

Figure 2: **Upper Left:** The original desired signal $d(n)$ with the noised version $d(n) + v_1(n)$. **Upper Right:** The reconstructed signal $\hat{d}(n)$ (after noise remove of $\hat{v}_1(n)$ from the signal $y(n)$) when $\alpha = 0.05$. **Lower Left:** The reconstructed signal $\hat{d}(n)$ when $\alpha = 0.3$. **Lower Right:** When $\alpha = 1.0$.

to estimate from given sample paths of $x(n)$, $y(n)$, and statistics of $E\{d(n)d(n-m)\}$ the values of the following correlation

$$r_y(m)\,, \quad p_{xy}(m)\,, \quad \text{and} \quad r_d(m)\,,$$

which are the needed discrete correlation functions. For example, we can derive an estimate $\hat{p}_{xy}(m)$ of $p_{xy}(m)$ using averaging as

$$\hat{p}_{xy}(m) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)y(n-m) \quad \text{for} \quad m = 0, 1, \cdots, M-1\,.$$

Here we should assume that the maximum allowed delay $M-1$ is much smaller than the number of samples in our time series N i.e. $M \ll N$.

This problem as formulated is worked in the MATLAB script `prob_4_4_2.m`. The optimal coefficients w^o as a function of the specified α are presented in Figure 2. There in the first plot in the upper left we see the desired signal $d(n)$ plotted along with its noised counter part $d(n) + v_1(n)$. We then plot three reconstructions for $\alpha = 0.05, 0.3, 1.0$. The reconstructions for small α seem to be much better.

Note: I'm not sure why the reconstructions for small α are better. In the formulation above the explicit dependence of α is be statistically considered and should not present a problem for the reconstruction. I would expect that if I had *not* modeled the α dependence I would see results like we are seeing here. I've checked this code several times and have not been able to find any errors. If anyone sees anything wrong with what I have done here, please let me know.

Problem 4.4.3 (two example MSE surfaces)

Part (a): The diagram shown for this problem looks like what might be a system modeling problem. In that we are seeking a coefficient w_0 such that $s(n) + w_0 s(n-1)$ is an approximate $d(n)$. Using this expression the value of the error at time step n can be written as

$$e(n) = d(n) - (s(n) + w_0 s(n-1)) .$$

We then desire to find a value for w_0 so that our estimate of $d(n)$ (using the signal $s(n)$) is as small as possible. In this case the cost function $J(w_0)$ we want to minimize can be defined as

$$J(w_0) = \min_{w_0} E\{e^2(n)\} .$$

Where $e^2(n)$ is given by

$$\begin{aligned} e^2(n) &= d^2(n) - 2d(n)(s(n) + w_0 s(n-1)) \\ &+ s^2(n) + 2w_0 s(n-1)s(n) + w_0^2 s^2(n-1) . \end{aligned}$$

With this expression for $e^2(n)$ the expectation when we use the provided values then becomes

$$\begin{aligned} E\{e^2(n)\} &= 3 - 2(-0.5 + w_0(0.9)) + 0.9 + 2w_0(0.4) + w_0^2(0.9) \\ &= 4.0 - w_0 + 0.9w_0^2 . \end{aligned}$$

To minimize this, we take its derivative, set the result equal to zero, and solve for w_0. We find that w_0 must satisfy

$$-1 + 1.8w_0 = 0 \quad \text{or} \quad w_0 = 0.5556\,.$$

Part (b): This problem is no difference than in Part (a) but now $e(n)$ is defined as

$$e(n) = d(n) + (w_0 s(n) + s(n-1))\,,$$

and we desire to minimize $J(w_0) = E\{e^2(n)\}$. The remaining parts of this problem is worked as in Part (a) above.

Chapter 5 (Eigenvalues of R_x - properties of the error surface)

Problem 5.1.1 (the correlation matrix R is positive definite)

From the definition of the autocorrelation matrix R, and with a is a constant vector we have that the product $a^T R a$ is given by

$$a^T R a = a^T E\{xx^T\}a = E\{a^T xx^T a\} = E\{(x^T a)^T(x^T a)\} = E\{||x^T a||^2\} \geq 0\,,$$

since the last expression is the expectation of a positive quantity. Since $a^T R a \geq 0$ for all a the autocorrelation matrix R is positive definite.

Problem 5.1.2 (eigenvalues of R^k)

If λ_i is an eigenvalue of R then by definition there exists an eigenvector q_i such that $Rq_i = \lambda_i q_i$. Then if $k \geq 1$ multiplying this equation by the matrix R^{k-1} on both sides we obtain

$$\begin{aligned} R^k q_i &= \lambda_i R^{k-1} q_i \\ &= \lambda_i R^{k-2}(Rq_i) = \lambda_i R^{k-2}\lambda_i q_i \\ &= \lambda_i^2 R^{k-2} q_i = \cdots = \lambda_i^k q_i\,, \end{aligned}$$

which shows that λ_i^k is an eigenvalue of R^k as claimed.

Problem 5.1.3 (distinct eigenvalues have independent eigenvectors)

To be linearly independent means that any finite (non-zero) sum of the given vectors cannot be zero. Thus if c_i for $i = 1, 2, \cdots, M$ are non-zero constants we require $\sum_{i=1}^M c_i q_i \neq 0$. Assume by way of contradiction that the c_i are not all zero but that

$$\sum_{i=1}^M c_i q_i = 0\,. \tag{5}$$

Then taking the dot-product of this equation with the vector q_j gives

$$\sum_{i=1}^M c_i q_j^T q_i = 0\,.$$

Now from Problem 5.1.5 $q_j^T q_i = 0$ for $i \neq j$ and the above equation reduces to $c_j = 0$ which is a contradiction. The book gives another solution where we generate M non-singular equations for c_i by multiplying the Equation 5 by R_x $0, 1, 2, \cdots, M$ times. The fact that the equations are non-singular and have a zero right-hand-side implies again that $c_i = 0$.

Problem 5.1.4 (the eigenvalues of R are real and non-negative)

We let q we be an eigenvector of R_x, then by definition $R_x q = \lambda q$. Multiplying this expression by q^H (the Hermitian conjugate of q) on both sides gives

$$q^H R_x q = \lambda q^H q \quad \text{or} \quad \lambda = \frac{q^H R_x q}{q^H q} \,.$$

Since R_x is positive definite $q^H R_x q \geq 0$ (by Problem 5.1.1) and $q^H q > 0$ everything in the ratio in the equation on the right hand side above is *real* and non-negative. Thus we can conclude that λ must be real and non-negative.

Problem 5.1.5 (distinct eigenvalues have orthogonal eigenvectors)

Let q_i and q_j be two eigenvectors of R_x corresponding to distinct eigenvalues. Then by definition

$$R_x q_i = \lambda_i q_i \quad \text{and} \quad R_x q_j = \lambda_j q_j \,,$$

Take the Hermitian inner product of q_j with the first equation we find

$$q_j^H R_x q_i = \lambda_i q_j^H q_i \,,$$

Taking the conjugate transpose of this expression and remembering that λ is a real number we have

$$q_i^H R_x q_j = \bar{\lambda}_i q_i^H q_j = \lambda_i q_i^H q_j \,. \tag{6}$$

Now the left hand side of this expression (since q_j is an eigenvector of R_x) is given by

$$q_i^H R_x q_j = q_i^H \lambda_j q_j = \lambda_j q_i^H q_j \,, \tag{7}$$

Thus subtracting Equation 6 from 7 we have shown the identity

$$\lambda_i q_i^H q_j - \lambda_j q_i^H q_j = 0 \,,$$

or

$$(\lambda_i - \lambda_j) q_i^H q_j = 0 \,.$$

Since we are assuming that $\lambda_i \neq \lambda_j$ the only way the above can be true is if $q_i^H q_j = 0$ that is the vector q_i and q_j are orthogonal.

Problem 5.1.6 (the eigenvector decomposition of R_x)

We begin by forming the matrix Q as suggested. Then it is easy to see that when we left multiply by R_x we obtain

$$\begin{aligned} R_x Q &= \begin{bmatrix} R_x q_1 & R_x q_2 & \cdots & R_x q_M \end{bmatrix} \\ &= \begin{bmatrix} \lambda_1 q_1 & \lambda_2 q_2 & \cdots & \lambda_M q_M \end{bmatrix} = Q\Lambda \,. \end{aligned}$$

Multiplying this last equation by Q^H on the left then because of the orthogonality of q_i and q_j under the Hermitian inner product we find

$$Q^H R_x Q = \Lambda \,,$$

with Λ a diagonal matrix containing the eigenvalues of R_x. These manipulations assumes that the vectors q_i and q_j are orthogonal to each other. This in fact can be made true when these vectors are constructed.

Problem 5.1.7 (the trace of the matrix R_x)

Now from Problem 5.1.6 we know that

$$\mathrm{tr}(Q^H R_x Q) = \mathrm{tr}(\Lambda) = \sum_{i=1}^{M} \lambda_i \,.$$

In addition to this identity, one can show that the trace operator satisfies a type of permutation identity in its arguments in that

$$\mathrm{tr}(ABC) = \mathrm{tr}(BCA) = \mathrm{tr}(CAB) \,, \tag{8}$$

provided that all of these products are defined. Using this identity we see that

$$\mathrm{tr}(Q^H R_x Q) = \mathrm{tr}(R_x Q Q^H) = \mathrm{tr}(R_x I) = \mathrm{tr}(R_x) \,,$$

the desired identity.

Problem 5.2.1 (the equation for the difference from optimal w^o)

We begin by recalling the definition of our error function $J(w)$ in terms of second order statistics of our processes $x(n)$ and $d(n)$

$$J(w) = \sigma_d^2 - 2w^T p + w^T R_x w \,, \tag{9}$$

or

$$w^T R_x w - 2p^T w - (J - \sigma_d^2) = 0 \,.$$

We now "center" this equation about the optimal Wiener-Hopf solution w^o which is given by $w^o = R_x^{-1} p_{xd}$. We do this by introducing a vector ξ defined as $\xi = w - w^o$. This means that $w = \xi + w^o$ and our quadratic form equation above becomes

$$(\xi + w^o)^T R_x (\xi + w^o) - 2p^T(\xi + w^o) - (J - \sigma_d^2) = 0 \,,$$

or expanding everything

$$\xi^T R_x \xi + 2\xi^T R_x w^o + w^{oT} R_x w^o - 2p^T \xi - 2p^T w^o - (J - \sigma_d^2) = 0 \,.$$

Since $R_x w^o = p$ some terms cancel and we get

$$\xi^T R_x \xi - p^T w^o - (J - \sigma_d^2) = 0 \,. \tag{10}$$

Recalling that

$$J(w^o) = J(\xi = 0) = J_{\min} = \sigma_d^2 - p^T w^o \,,$$

we see that Equation 10 is given by

$$\xi^T R_x \xi - p^T w^0 - (J - \sigma_d^2) = \xi^T R_x \xi - J + J_{\min} = 0 \,.$$

Thus $J - J_{\min} = \xi^T R_x \xi$, which states by how much J is greater than than $J_{\min}$ when $\xi \neq 0$.

Chapter 6 (Newton and steepest-decent method)

Problem 6.1.1 (convergence of w in the gradient search algorithm)

Recall that the difference equation satisfied by the one-dimensional filter coefficients $w(n)$ when using the gradient search algorithm is given by

$$w(n+1) = (1 - 2\mu r_x(0))w(n) + 2\mu r_x(0)w^o \tag{11}$$

To solve this difference equation define $v(n)$ as $v(n) = w(n) - w^o$, so that w in terms of v is given by $w(n) = w^o + v(n)$ and then Equation 11 becomes

$$w^o + v(n+1) = (1 - 2\mu r_x(0))(w^o + v(n)) + 2\mu r_x(0)w^o \,,$$

or

$$v(n+1) = (1 - 2\mu r_x(0))v(n) \,. \tag{12}$$

The solution of this last equation is

$$v(n) = (1 - 2\mu r_x(0))^n v(0) \,,$$

which can be proven using mathematical induction or by simply substituting this expression into the difference equation 12 and verifying that it is a solution. Replacing $v(n)$ with $w(n) - w^o$ we have that $w(n)$ is given by

$$w(n) = w^o + (1 - 2\mu r_x(0))^n (w(0) - w^o) \,,$$

as we were to prove.

Problem 6.1.2 (visualizing convergence)

Equation 6.15. is the iterative solution to the one-dimensional gradient-decent search algorithm

$$w(n) = w^o + (1 - 2\mu r_x(0))^n (w(0) - w^o) \,.$$

To generate these plots we took $w^o = 0.5$, $w(0) = -0.5$, and $r_x(0) = 1$ and several values for μ. We then plot the iterates of $w(n)$ as a function of n in Figure 3 (left). These plots can be generated by running the MATLAB script `prob_6_1_2.m`.

Part (a): When $0 \leq \mu \leq \frac{1}{2r_x(0)}$ from the given plots we see that the convergence is monotonic to the solution of 0.5.

Part (b): When $\mu \approx \frac{1}{2r_x(0)}$ we see that the convergence is also monotonic but converges faster than before to the true solution.
Part (c): When $\frac{1}{2r_x(0)} < \mu < \frac{1}{r_x(0)}$ the convergence is oscillatory around the true solution and eventually converges to it.
Part (d): When $\mu > \frac{1}{r_x(0)}$ the iterates oscillate and then diverge. That is the iterates and don't converge to the value of $w^o = 0.5$. In the given plot we only plot the first five samples so as to not clutter the graph.

Problem 6.1.3 (convergence of J in the gradient search algorithm)

Recall the solution to the iteration scheme for the weights $w(n)$

$$w(n) = w^o + (1-2\mu)^n (w(0) - w^o)\,,$$

When we put this into the books Equation 5.2.7 the shifted and rotated form for J we obtain

$$\begin{aligned} J(w(n)) &= J_{\min} + (w(n) - w^o)^T R_x (w(n) - w^o) \quad (13) \\ &= J_{\min} + (1-2\mu)^{2n} (w(0) - w^o)^T R_x (w(n) - w^o) \quad (14) \end{aligned}$$

Evaluating Equation 13 when $n = 0$ results in

$$J(0) = J(w(0)) = J_{\min} + (w(0) - w^o)^T R_x (w(0) - w^o)\,.$$

or solving for the quadratic form the expression

$$(w(0) - w^o)^T R_x (w(0) - w^o) = J(0) - J_{\min}\,.$$

Using this result back in Equation 14 gives the desired form for the iterates of $J(n)$

$$J(n) = J(w(n)) = J_{\min} + (1-2\mu)^n (J(0) - J_{\min})\,. \quad (15)$$

Problem 6.2.1 (convergence of the vector SD algorithm)

The vector steepest-decent (SD) algorithm results in the iterative scheme for the vector $\xi'(n)$ defined as

$$\xi'(n) = Q^T (w(n) - w^o)\,.$$

Here $R_x = Q \Lambda Q^T$. That is, Q is the orthogonal matrix that diagonalizes the autocovariance matrix R_x. The iterative scheme that results for $\xi'(n)$ is given by

$$\xi'(n+1) = (I - \mu' \Lambda) \xi'(n)\,,$$

Then taking the kth component of the above matrix expression results in the following scalar iterative scheme

$$\xi_k'(n+1) = (1-\mu'\lambda_k)\xi_k'(n)\,.$$

Problem 6.2.2 (the scalar equations in the vector GD algorithm)

The ith scalar equation in the vector gradient-decent algorithm is given by equation 6.2.25

$$w_i'(n+1) = (1-\mu'\lambda_i)w_i'(n) + \mu' p_{idx}' \quad \text{for} \quad 0 \le i \le M-1\,. \tag{16}$$

Here $\mu' = 2\mu$ is the learning rate. Letting $n = 0$ in this equation we obtain

$$w_i'(1) = (1-\mu'\lambda_i)w_i'(0) + \mu' p_{idx}'\,.$$

Letting $n = 1$ we obtain

$$\begin{aligned} w_i'(2) &= (1-\mu'\lambda_i)w_i'(1) + \mu' p_{idx}' \\ &= (1-\mu'\lambda_i)^2 w_i'(0) + \mu'(1-\mu'\lambda_i)p_{idx}' + \mu' p_{idx}'\,, \end{aligned}$$

when we use the previously computed value of $w_i'(1)$. Continuing with $w_i'(3)$ we let $n = 2$ in Equation 16 to obtain

$$\begin{aligned} w_i'(3) &= (1-\mu'\lambda_i)w_i'(2) + \mu' p_{idx}' \\ &= (1-\mu'\lambda_i)^3 w_i'(0) + \mu'(1-\mu'\lambda_i)^2 p_{idx}' + \mu'(1-\mu'\lambda_i)p_{idx}' + \mu' p_{idx}'\,. \end{aligned}$$

By induction then we recognize that in the general case

$$w_i'(n) = (1-\mu'\lambda_i)^n w_i'(0) + \mu' p_{idx}' \sum_{j=0}^{n-1} (1-\mu'\lambda_j)^j\,, \tag{17}$$

as we were to show.

Problem 6.2.3 (analytic solutions for the iterates $w(n)$)

The exact iterates for the transformed weight components $w_i'(n)$ satisfy (assuming we begin our iterations with w initialized to zero) the following

$$w_i'(n) = \frac{p_{idx}'}{\lambda_i}(1-(1-\mu'\lambda_i)^n)\,. \tag{18}$$

Using this expression and the given autocorrelation matrix $\hat{R}_x$ and cross-correlation vector $\hat{p}_{dx}$ we can *analytically* evaluate the transformed filter weights $w_i'(n)$ as a function of n. From these we can translate $w_i'(n)$ into analytic expressions for the filter weights $w_i(n)$ themselves. To do this we require we see from Equation 18 requires the eigenvalues of the given $\hat{R}_x$. Computing them we find their values given by

$$\lambda_1 = 0.3 \quad \text{and} \quad \lambda_2 = 1.7\,.$$

While the eigenvectors for R_x are given by the columns of a matrix say Q or

$$Q = \begin{bmatrix} -0.7071 & 0.7071 \\ 0.7071 & 0.7071 \end{bmatrix}.$$

We also require

$$p'_{xd} = Q^T p_{xd} = \begin{bmatrix} -0.1414 \\ 0.8485 \end{bmatrix}.$$

Using these (and recalling that $\mu' = 2\mu$) we find

$$\begin{aligned} w_1'(n) &= -\frac{0.1414}{0.3}(1-(1-2\mu 0.3)^n) = -\frac{0.1414}{0.3}(1-(1-0.6\mu)^n) \\ w_2'(n) &= +\frac{0.8485}{1.7}(1-(1-2\mu 1.7)^n) = +\frac{0.8485}{1.7}(1-(1-3.4\mu)^n)\,, \end{aligned}$$

as the functional form for $w_i'(n)$. Given these expressions we can compute the filter weights themselves $w_i(n)$ by multiplying them by the Q matrix[2]. In components then this is given by

$$\begin{aligned} w_1(n) &= -0.7071 w_1'(n) + 0.7071 w_2'(n) \\ &= +\frac{0.7071(0.1414)}{0.3}(1-(1-0.6\mu)^n) + \frac{0.7071(0.8485)}{1.7}(1-(1-3.4\mu)^n) \\ w_2(n) &= +0.7071 w_1'(n) + 0.7071 w_2'(n) \\ &= -\frac{0.7071(0.1414)}{0.3}(1-(1-0.6\mu)^n) + \frac{0.7071(0.8485)}{1.7}(1-(1-3.4\mu)^n)\,. \end{aligned}$$

For convergence of this iterative scheme recall that the learning rate $\mu' \leq \frac{2}{\lambda_{\max}}$. Since $\mu' = 2\mu$ this means that

$$\mu \leq \frac{1}{\lambda_{\max}} = 0.5882\,.$$

[2]Recall that the transformed weights $w'(n)$ were obtained from the untransformed weights by $w'(n) = Q^T w(n)$.

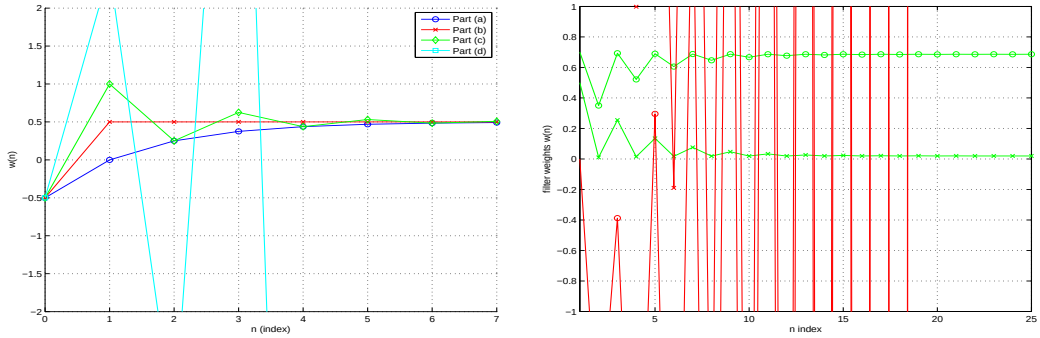

Figure 3: **Left:** Plots of iterates of the 1d gradient-search algorithm. **Right:** Plots of the two filter coefficients $w_1(n)$ and $w_2(n)$ as a function of n for two different values of the learning parameter μ. See the text on Problem 6.2.3 for more details.

These expressions for the weights w_i are plotted in Figure 3 (right) for $\mu = 0.5$ and $\mu = 0.7$ as a function of n. The two numerical values of μ were chosen to straddle the stability threshold of $\mu \approx 0.5882$. The value of the iterates for $\mu = 0.5$ is plotted in green while that for $\mu = 0.7$ is plotted in red. We only plot the first ten elements of the $\mu = 0.7$ curves since the large oscillations that result from the divergence would cloud the entire picture otherwise. We also clip the y-axis of the plot at reasonable limits since the diverging weights quickly reach very large numbers. This plot can be generated and studied by running the MATLAB script `prob_6_2_3.m`.

Problem 6.2.4 ($J(w(n))$ in terms of $p_{dx}(n)$ and R_x)

Given the expression for $J(w(n))$

$$J(w(n)) = J_{\min} + \xi'(n)^T \Lambda \xi'(n)\,,$$

expressed in terms of the rotated and translated vectors $\xi'(n) = Q^T\xi(n) = Q^T(w(n) - w^o)$ in terms of Q and $w(n)$ this becomes

$$\begin{aligned} J(n) &= J_{\min} + (w(n) - w^o)^T Q \Lambda Q^T (w(n) - w^o) \\ &= J_{\min} + (w(n) - w^o)^T R_x (w(n) - w^o)\,. \end{aligned}$$

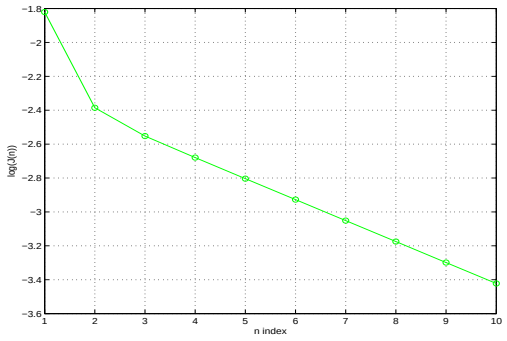

Figure 4: Plots of $\ln(J(n))$ as a function of n for Problem 6.2.5.

Recalling that the optimal Wiener filter weights w^o satisfy $w^o = R_x^{-1} p_{dx}$ and that $J_{\min} = \sigma_d^2 - p_{dx}^T w^o$ the above becomes

$$\begin{aligned} J(n) &= \sigma_d^2 - p_{dx}^T ({R_x}^{-1} p_{dx}) + (w(n) - w^o)^T R_x (w(n) - w^o) \\ &= \sigma_d^2 - p_{dx}^T {R_x}^{-1} p_{dx} \\ &+ w(n)^T R_x w(n) - w(n)^T R_x {R_x}^{-1} p_{dx} - p_{dx}^T {R_x}^{-1} R_x w(n) + p_{dx}^T {R_x}^{-1} R_x {R_x}^{-1} p_{dx} \\ &= \sigma_d^2 - 2 p_{dx}^T w(n) + w(n)^T R_x w(n) \,, \end{aligned}$$

or the desired expression.

Problem 6.2.5 (plots of the learning curve)

One form of the learning curve, $J(n)$, in terms of the eigenvalues of the autocorrelation matrix R_x, is

$$\begin{aligned} J(n) &= J(w(n)) = J_{\min} + \xi'(n)^T \Lambda \xi'(n) \\ &= J_{\min} + \sum_{k=0}^{M-1} \lambda_k (1 - \mu' \lambda_k)^{2n} {\xi'}_k(0)^2 \,. \end{aligned}$$

For the correlation matrix with the given eigenvalues we have that the learning curve has the following specific expression

$$J(n) - J_{\min} = 1.85(1 - 1.85\mu')^{2n} {\xi'}_1(0)^2 + 0.15(1 - 0.15\mu')^{2n} {\xi'}_2(0)^2 \,.$$

The two values for the constants ${\xi'}_1(0)^2$ and ${\xi'}_1(0)^2$ are determined by what we take for our initial guess at the filter coefficients (rotated by the eigenvectors of the system correlation matrix R_x). The value of μ' must be selected

such that we have convergence of the gradient decent method which in this case means that

$$\mu' < \frac{2}{\lambda_{\max}} = \frac{2}{1.85} = 1.0811\,.$$

We plot the log of the expression $J(n)$ for this problem in Figure 4 for $\mu' = 0.4$, ${\xi'}_1(0)^2 = 0.5$, and ${\xi'}_1(0)^2 = 0.75$. The time constants, τ_k, for the error decay are given by

$$\tau_k = -\frac{1}{\ln(1-\mu'\lambda_k)}\,,$$

which in this case gives

$$\tau_1 = 0.7423 \quad \text{and} \quad \tau_2 = 16.1615\,.$$

This plot can be generated by running the MATLAB command `prob_6_2_5`.

Problem 6.2.6 (the optimal value for μ')

The optimal value for μ' lies between $\frac{1}{\lambda_{\min}}$ and $\frac{1}{\lambda_{\max}}$ so that

$$\begin{aligned} |1-\lambda_{\min}\mu'| &= \lambda_{\min}\left|\frac{1}{\lambda_{\min}} - \mu'\right| \\ &= \lambda_{\min}\left(\frac{1}{\lambda_{\min}} - \mu'\right) \\ &= 1-\lambda_{\min}\mu'\,, \end{aligned}$$

since $\mu' < \frac{1}{\lambda_{\min}}$. In the same way we have

$$\begin{aligned} |1-\lambda_{\max}\mu'| &= \lambda_{\max}\left|\frac{1}{\lambda_{\max}} - \mu'\right| \\ &= \lambda_{\max}\left(\mu' - \frac{1}{\lambda_{\max}}\right) \\ &= \lambda_{\max}\mu' - 1\,, \end{aligned}$$

since $\mu' > \frac{1}{\lambda_{\max}}$. So solving $|1-\lambda_{\min}\mu'| = |1-\lambda_{\max}\mu'|$ is equivalent to solving

$$1-\lambda_{\min}\mu' = \lambda_{\max}\mu' - 1\,,$$

or

$$\mu' = \frac{2}{\lambda_{\min} + \lambda_{\max}} .$$

With this optimal value of μ' the convergence is determined by

$$\alpha = 1 - \mu'_{\text{opt}} \lambda_{\min} = 1 - \frac{2\lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} = \frac{\frac{\lambda_{\min}}{\lambda_{\max}} - 1}{\frac{\lambda_{\min}}{\lambda_{\max}} + 1} ,$$

as claimed in the book.

Chapter 7 (The least mean-square (LMS) algorithm)

Notes From the Text

Using the LMS algorithm for linear prediction

In this subsection we will duplicate the linear prediction example from the book. We assume we have a zero-mean white noise driving process $v(n)$ and that the observed process $x(n)$ is given by an AR(2) model of the form

$$x(n) = 0.601x(n-1) - 0.7225x(n-2) + v(n)\,,$$

We desire to predict the next value of x at the timestep n using the previous two values at $n-1$ and $n-2$. That is we desire to compute an estimate $\hat{x}(n)$ of $x(n)$ from

$$\hat{x}(n) = \sum_{i=0}^{1} w_i(n)x(n-1-i)\,.$$

To do this we will use the LMS algorithm. The computations are performed in the MATLAB file `linear_prediction_w_lms.m`. We perform lms learning with two different learning rates $\mu = 0.02$ and $\mu = 0.005$. We expect that on "easy" problems all things begin equal lms algorithm with a larger learning rate μ will produce faster convergence and a give optimal results sooner. The results from this experiment are shown in Figure 5. There we plot the original signal $x(n)$ and its prediction $\hat{x}(n)$ at the n-th step. We can see that after a certain amount of time we are predicting $x(n)$ quite well. Next we plot the error between the true observed value of $x(n)$ and our estimate $\hat{x}$. This error is centered on zero and has the same variance as the unknown innovation term $v(n)$ in our AR(2) model. Finally, we plot the estimates of the weights found during the lms learning procedure. We see nice convergence to the truth values (shown as horizontal green lines). Note that we are assuming that we are estimating $M = 2$ coefficients from the signal $x(n)$. An attempt to estimate more coefficients $M > 2$ will work but the coefficients require more iterations to estimate their values sufficiently. For example when $M = 3$ we begin to estimate the third weight w_3 as zero after sufficient time.

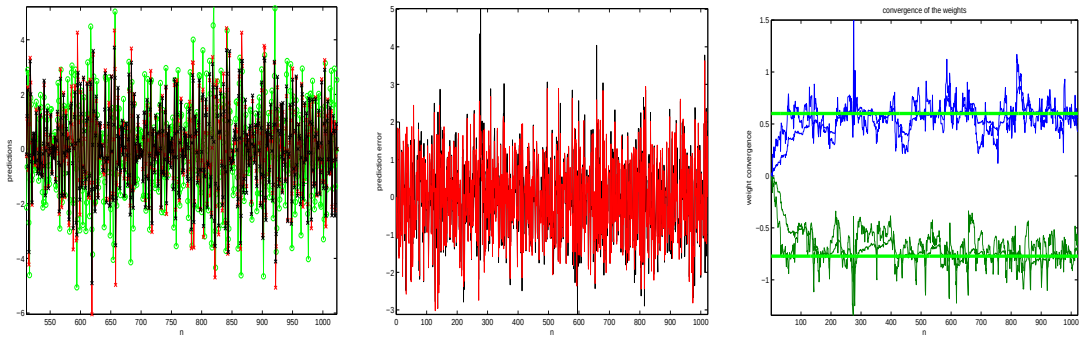

Figure 5: Using the LMS algorithm for linear prediction. **Left:** The signal $x(n)$ from which we use the previous two time-steps $x(n-1)$ and $x(n-2)$ in predicting the next value $x(n)$. **Center:** The error $x(n) - \hat{x}(n)$ at each timestep. **Right:** The convergence of the AR(2) weights $[w_1, w_2]^T$ as a function of timestep n.

Using the LMS algorithm for modeling unknown systems

In this subsection we will use the LMS algorithm to estimate the unknown coefficients of a MA(3) model given the measured signal input signal $d(n)$. That is we assume (this fact is unknown to the lms algorithm) that our observed signal $d(n)$ is given by a MA(3) model based on $x(n)$ with coefficients

$$d(n) = x(n) - 2x(n-1) + 4x(n-2)\,.$$

Here we will take $x(n)$ to be the AR(2) signal model used in the previous example. We then assume a model of our observed signal $d(n)$ given by a MA(M) or

$$\hat{d}(n) = \sum_{i=0}^{M} w_i x(n-i)\,.$$

We will estimate the parameters w_i using the lms algorithm and the book MATLAB function `aalms1.m`. We do this in the MATLAB script called `modeling_w_lms.m`. The results from these experiments are shown in Figure 6. If we specify to a MA(3) model of $x(n)$ we see that our LMS algorithm is able to learn quite nicely the three weights w_i.

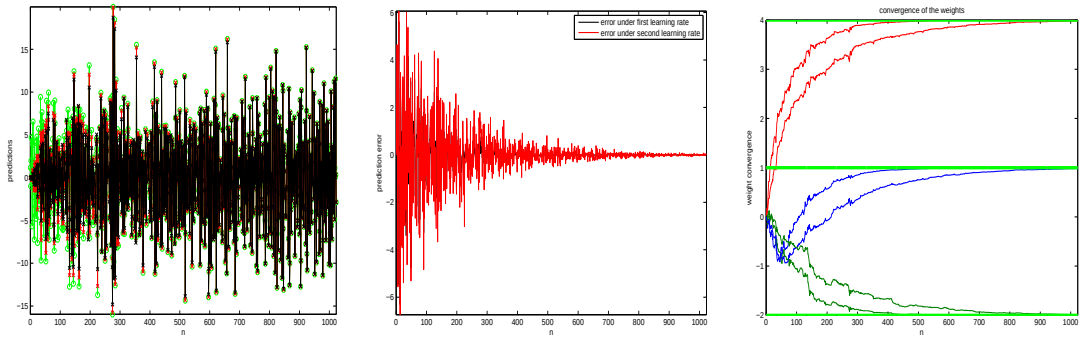

Figure 6: Using the LMS algorithm for modeling. **Left:** The output signal $d(n)$ and our estimated signal $\hat{d}(n)$, which we assume is modeled as a MA(M) processed based on the input signal $x(n)$. **Center:** The error $x(n) - \hat{x}(n)$ in our prediction at each timestep. Notice that as the number of time-steps increases the approximation gets better. **Right:** The convergence of the MA(3) weights $[w_0, w_1, w_2]^T$ as a function of timestep n.

Using the LMS algorithm for noise cancellation

In this example we will use the LMS algorithm for noise cancellation. This means that we assume that we are given a signal $s(n)$ that has been modified by additive noise $s(n)+v(n)$. We then desire to use the previous sample $s(n-1)+v(n-1)$ to predict the current sample $s(n)+v(n)$. Here our filter input is $x(n) = s(n-1)+v(n-1)$, while our desired signal is $d(n) = s(n)+v(n)$. This example is worked in the MATLAB script `noise_cancellation_w_lms.m`.

Using the LMS algorithm for inverse system identification

For this subsection we will try to apply the LMS algorithm for numerically computing the inverse of an unknown system. To do this we will assume that we have a sinusoidal input signal $s(n)$ given by

$$s(n) = \sin(0.2\pi n)\,.$$

To this signal we add some random Gaussian noise $v(n)$, that we are not able to predict. This modified signal $m(n) \equiv s(n) + v(n)$ is then passed into an unknown filter. For this example we will assume that this filter is MA(4) system the coefficients of which are unknown to our inverse system

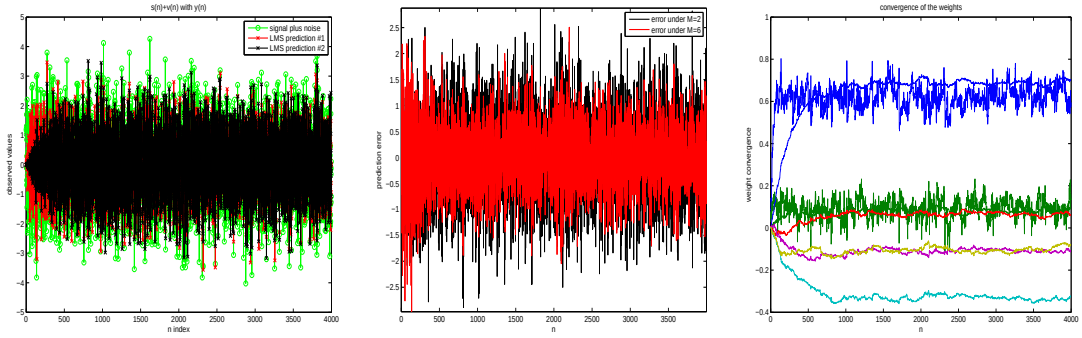

Figure 7: Using the LMS algorithm for inverse system identification. **Left:** A plot of the input signal $x(n)$ and two reconstructions with different number of coefficients M. The first has $M = 2$ while the second has $M = 6$. **Center:** The instantaneous error $(m(n) - y(n))^2$ for each model. Note that the second model with $M = 6$ ends with a smaller error overall. **Right:** The convergence of the filter coefficients w_i for each of the models.

algorithm. For this example we assume that we measure the output $x(n)$ of a MA(4) system given by

$$x(n) = 1.0m(n) + 0.2m(n-1) - 0.5m(n-2) + 0.75m(n-3)\,.$$

Given this modified signal $x(n)$, and the input signal, $m(n)$ to the unknown filter we want to construct a FIR filter that will estimate the *inverse* of the input filter. That is we are looking for a set of filter coefficients w_i such that we have an estimate $\hat{m}(n)$ of $m(n)$

$$y(n) = \hat{m}(n) = \sum_{i=0}^{M} w_i x(n-i)\,.$$

Now in a physical realization of this filter because it will take four time-steps before the first output of our system $x(n)$ appears and an additional number M of steps before the output from our inverse system appears we would need to compare the output of this combined filter system with a value of $m(n)$ that has already passed. The practical implementation of this is that we would need to *delay* the signal $m(n)$ by some amount before we can calculate an error term. Strictly in a software digital environment this is *not* needed since we desire to compare $m(n)$ with the output the combined filter.

This example is implemented in the MATLAB script `inv_system_w_lms.m`. The results obtained when running this code can be seen in Figure 7. We plot the signal that is observed after obtained noise $m(n)$ along with the LMS learned reconstructed signal for two filter designs $y_1(n)$ and $y_2(n)$. The two filter designs are different in that they have a different number of filter coefficients w_i and in the learning rate used for each implementation. The first experiment has $M = 2$ and $\mu = 0.01$, while the second has more filter coefficients at $M = 6$ and a smaller learning rate $\mu = 0.001$ The consequence of this is that the filter with $M = 6$ weights should do a better job at approximating the inverse of our system. This can be seen when we consider a plot of the error in the prediction $y_2(n)$ compared with that in $y_1(n)$. The error in $y_2(n)$ is uniformly smaller with a standard deviation of 0.637990 compared to the same thing under the $M = 2$ filter coefficient where we obtain an error standard deviation of 0.831897. In addition, the smaller learning rate in the second filter means that (all things begin equal) it will take more time for the filter to obtain optimal performance. This can be seen in the signal plot where the first filter produces better approximations earlier and in the error plots where the second filter has a larger error initially. We finally plot the convergence of the filter coefficients where we see that the $M = 6$ filter's initial two weights w_0 and w_1 converge to the same thing as the $M = 2$'s filter's weight.

The performance analysis of the LMS algorithm

In this subsection we expand and discuss the algebra and presentation on the LMS algorithm presented in the text. This section of the text was developed to further expand my understanding of these manipulations. We begin with equation 7.4.5 given by

$$\xi(n+1) = \xi(n) + 2\mu x(n)(e^o(n) - x(n)^T\xi(n))\,. \tag{19}$$

Where $\xi(n) = w(n) - w^o$ are the weight error vectors. We multiply this equation on the left by Q, the matrix which orthogonalizes the autocorrelation matrix for $x(n)$, that is $R_x = Q\Lambda Q^T$. If we multiply Equation 19 on the left by Q^T and recalling the definitions $\xi' = Q^T\xi$ and $x' = Q^T x$ and that Q is an orthogonal matrix we find

$$\begin{aligned}\xi'(n+1) &= \xi'(n) + 2\mu x'(n)(e^o(n) - (Q^T x)^T(Q^T\xi)) \\ &= \xi'(n) + 2\mu x'(n)(e^o(n) - x'(n)^T\xi'^T)\end{aligned}$$

$$
\begin{aligned}
&= 2\mu x'(n)e^o(n) + \xi'(n) - 2\mu x'(n)(x'^T\xi') \\
&= (I - 2\mu x'(n)x'(n)^T)\xi'(n) + 2\mu e^o(n)x'(n)\,, \qquad (20)
\end{aligned}
$$

which is equation 7.4.24 in the book. The transpose of Equation 20 is given by

$$
\xi'(n+1)^T = \xi'(n)^T(I - 2\mu x'(n)x'(n)^T) + 2\mu e^o(n)x'(n)^T\,.
$$

Now for notational simplicity in the remaining parts of this derivation of an expression for $K'(n)$ we will drop the prime on the vectors x and ξ. This prime notation was used to denote the fact that x and ξ are viewed in the space rotated by the eigenvalues of R_x i.e. $x' = Q^T x$ with $R_x = Q\Lambda Q^T$. We will also drop the n notation which indicates that we have processed up to and including the nth sample, both notations would be present on all symbols and just seem to clutter the equations. Multiplying 20 by its transpose (computed above) gives

$$
\begin{aligned}
\xi(n+1)\xi(n+1)^T &= (I - 2\mu xx^T)\xi\xi^T(I - 2\mu xx^T) \\
&+ 2\mu e^o(I - 2\mu xx^T)\xi x^T \\
&+ 2\mu e^o x\xi^T(I - 2\mu xx^T) \\
&+ 4\mu^2(e^o)^2 xx^T \\
&= \xi\xi^T \qquad (21) \\
&- 2\mu\xi\xi^T xx^T \qquad (22) \\
&- 2\mu xx^T\xi\xi^T \qquad (23) \\
&+ 4\mu^2 xx^T\xi\xi^T xx^T \qquad (24) \\
&+ 2\mu e^o\xi x^T - 4\mu^2 e^o xx^T\xi x^T \qquad (25) \\
&+ 2\mu e^o x\xi^T - 4\mu^2 e^o x\xi^T xx^T \qquad (26) \\
&+ 4\mu^2(e^o)^2 xx^T\,. \qquad (27)
\end{aligned}
$$

We won't use this fact but the above could be simplified by recalling that that inner products are symmetric that is $x\xi^T = \xi x^T$. If we take the expectation $E\{\cdot\}$ of this expression we obtain the desired expression the the books equation 7.4.25. To explicitly evaluate the above expectation we will use the independence assumption, which basically states that the data, $x(n)$, going into our filter is independent of the filter coefficient estimates, $w(n)$, at least with respect to taking expectations. Since the filter coefficients and data are represented in the transformed space as $\xi'(n)$ and $x'(n)$, this means that the

expectation of the term 24 (equation 7.4.28 in the book) can be computed as

$$E\{xx^T\xi\xi^T xx^T\} = E\{xx^T E\{\xi\xi^T\}xx^T\}\,,$$

where we have used the independence assumption in passing an expectation inside the outer expectation. Since $E\{\xi'(n)\xi'^T(n)\} = K'(n)$ the above becomes

$$E\{x'(n)x'(n)^T K'(n)x'(n)x'(n)^T\}\,.$$

To further simplify this consider the quadratic part in the middle of this expression. That is write $x'(n)^T K'(n)x'(n)$ in terms of the components of $x'(n)$ and $K'(n)$. In standard matrix component notation we have

$$x'(n)^T K'(n)x'(n) = \sum_{i=0}^{M-1}\sum_{j=0}^{M-1} x'_i(n)x'_j(n)K'_{ij}(n)\,.$$

Multiplying this inner product scalar by the vector $x'(n)$ on the left, by $x'(n)^T$ on the right, and taking the lm-th component of the resulting matrix we obtain

$$(x'(n)x'(n)^T K'(n)x'(n)x'(n)^T)_{lm} = x'_l(n)x'_m(n)\sum_{i=0}^{M-1}\sum_{j=0}^{M-1} x'_i(n)x'_j(n)K'_{ij}(n)\,.$$

The expectation of the lm-th element is then given by

$$\begin{aligned} E\{(\cdot)_{lm}\} &= \sum_{i=0}^{M-1}\sum_{j=0}^{M-1} E\{x'_l(n)x'_m(n)x'_i(n)x'_j(n)K'_{ij}(n)\} \\ &= \sum_{i=0}^{M-1}\sum_{j=0}^{M-1} K'_{ij}(n)E\{x'_l(n)x'_m(n)x'_i(n)x'_j(n)\}\,. \end{aligned}$$

This last expression involves evaluating the expectation of the product of four Gaussian random variables. Following the solution in the text we recall an identity that expands such products in terms of *pairwise* products. We note that this is an advantage of using Gaussian random variables in that the higher order moments can be determined explicitly from the second (and possibly lower) moments. The needed identity is that the expectation of the fourth product of the x_i's is given by

$$E\{x_1x_2x_3x_4\} = E\{x_1x_2\}E\{x_3x_4\} + E\{x_1x_3\}E\{x_2x_4\} + E\{x_1x_4\}E\{x_2x_3\}\,. \tag{28}$$

In the specific case considered here these pairwise products are given by

$$E\{x'_i x'_j\} = (E\{x' x'^T\})_{ij} = (E\{Q^T x x^T Q\})_{ij} = (Q^T R_x Q)_{ij} = (\Lambda)_{ij} = \lambda_i \delta(i-j)\,.$$

Thus using Equation 28 the expectation of this lm-th component is given by

$$\begin{aligned} E\{(\cdot)_{lm}\} &= \sum_{i=0}^{M-1}\sum_{j=0}^{M-1} K'_{ij}(n)\lambda_l\lambda_i\delta(l-m)\delta(i-j) \\ &+ \sum_{i=0}^{M-1}\sum_{j=0}^{M-1} K'_{ij}(n)\lambda_l\lambda_m\delta(l-i)\delta(m-j) \\ &+ \sum_{i=0}^{M-1}\sum_{j=0}^{M-1} K'_{ij}(n)\lambda_l\lambda_m\delta(l-j)\delta(m-i) \\ &= \lambda_l\delta(l-m)\sum_{i=0}^{M-1}\lambda_i K'_{ii}(n) + \lambda_l\lambda_m K'_{lm}(n) + \lambda_l\lambda_m K'_{ml}(n)\,. \end{aligned}$$

As $K'(n)$ is symmetric $K'_{lm}(n) = K'_{ml}(n)$ and the last two terms are equal and we obtain

$$E\{(\cdot)_{lm}\} = \lambda_l\delta(l-m)\sum_{i=0}^{M-1}\lambda_i K'_{ii}(n) + 2\lambda_l\lambda_m K'_{lm}(n)\,.$$

Now the diagonal sum over the elements of $K'(n)$ is the trace as

$$\sum_{i=0}^{M-1}\lambda_i K'_{ii}(n) = \mathrm{tr}(\Lambda K'(n)) = \mathrm{tr}(K'(n)\Lambda)\,.$$

Here $\Lambda K'(n)$ multiplies the ith row of $K'(n)$ by λ_i where as $K'(n)\Lambda$ multiplies the i-th column of $K'(n)$ by λ_i, so that the diagonal terms of each product are identical. In matrix form then $E\{(\cdot)_{lm}\}$ is given by

$$\mathrm{tr}(\Lambda K'(n))\Lambda + 2\Lambda K'(n)\Lambda\,,$$

or the expression 7.4.28.

From the direct expectation of the component equations: 22, 23, 24, 25, 26, 27, and what we calculated above for $E\{(\cdot)_{lm}\}$, the recursive equation for the

matrix $K'(n+1)$ then becomes

$$\begin{aligned} K'(n+1) &= K'(n) - 2\mu(\Lambda K'(n) + K'(n)\Lambda) \\ &+ 4\mu^2(2\Lambda K'(n)\Lambda + \mathrm{tr}\{\Lambda K'(n)\}\Lambda) \\ &+ 4\mu^2 J_{\min}\Lambda\,, \end{aligned}$$

which is equation 7.4.34.

To study convergence of the LMS algorithm as $K'(n)$ is a correlation matrix we have, ${k'_{ij}}^2 \leq k'_{ii}k'_{jj}$, so the off-diagonal terms are bounded by the diagonal terms and it is sufficient to consider the iith element of $K'(n)$. From the above recursive expression for the entire matrix $K'(n)$ the ii-th component of 7.4.34 is given by

$$\begin{aligned} k'_{ii}(n+1) &= k'_{ii}(n) - 4\mu\lambda_i k'_{ii}(n) + 8\mu^2\lambda_i^2 k'_{ii}(n) \\ &+ 4\mu^2\lambda_i \sum_{j=0}^{M-1} \lambda_j k'_{jj}(n) + 4\mu^2 J_{\min}\lambda_i \\ &= (1 - 4\mu\lambda_i + 8\mu^2\lambda_i^2)k'_{ii}(n) \\ &+ 4\mu^2\lambda_i \sum_{j=0}^{M-1} \lambda_j k'_{jj}(n) + 4\mu^2 J_{\min}\lambda_i\,, \end{aligned} \tag{29}$$

which is equation 7.4.35. To derive a matrix recursive relationship for these components k'_{ii} we place their values for $i = 0, 1, \cdots, M-1$ in a *vector* (denoted as $k'(n)$ with no subscripts) and from the recursive scalar expression just considered we obtain a vector update equation as

$$\begin{aligned} \begin{bmatrix} k'_{00}(n+1) \\ k'_{11}(n+1) \\ \vdots \\ k'_{M-1,M-1}(n+1) \end{bmatrix} &= F \begin{bmatrix} k'_{00}(n) \\ k'_{11}(n) \\ \vdots \\ k'_{M-1,M-1}(n) \end{bmatrix} + 4\mu^2 J_{\min} \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{M-1} \end{bmatrix} \\ &+ 4\mu^2 \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{M-1} \end{bmatrix} \begin{bmatrix} \lambda_0 & \lambda_1 & \cdots & \lambda_{M-1} \end{bmatrix} \begin{bmatrix} k'_{00}(n) \\ k'_{11}(n) \\ \vdots \\ k'_{M-1,M-1}(n) \end{bmatrix}. \end{aligned}$$

Where we have defined the matrix F as

$$F = \begin{bmatrix} 1 - 4\mu\lambda_0 + 8\mu^2\lambda_0^2 & 0 & \cdots & 0 \\ 0 & 1 - 4\mu\lambda_1 + 8\mu^2\lambda_1^2 & & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & 0 & & 1 - 4\mu\lambda_{M-1} + 8\mu^2\lambda_{M-1}^2 \end{bmatrix},$$

or a diagonal matrix with diagonal elements given by $1 - 4\mu\lambda_i + 8\mu^2\lambda_i^2$. Defining these elements as f_i we find out matrix update equations becomes

$$k'(n+1) = \left(\text{diag}(f_0, f_1, \cdots, f_{M-1}) + 4\mu\lambda\lambda^T\right) k'(n) + 4\mu^2 J_{\min}\lambda\,, \qquad (30)$$

which is equation 7.4.36 in the book. This completes our analysis of the convergence of the weights in the LMS algorithm. We now consider how the error functional J behaves as $n \to \infty$.

We begin by defining J_{ex} as the difference between the current iterate of our error functional $J(n)$ and the best possible value for this. Thus we have

$$J_{\text{ex}}(\infty) \equiv J(\infty) - J_{\min} = \text{tr}(K(\infty)R_x)\,,$$

where we have used equation 7.4.17 to derive an expression for the excess mean square error $J_{\text{ex}}(\infty)$ in terms of $K(n)$ and the autocovariance matrix R_x. As an aside it may help to discuss the motivation for these algorithmic steps. We recognize that the LMS algorithm is an approximation to the optimal Wiener filter and as an approximation will not be able to produce a filter with a minimum mean square error $J_{\min}$. The filter the LMS algorithm will produce should have an error that larger than the smallest possible. We desire to study how this "excess error" behaves as we use the LMS algorithm. From the eigenvalue-eigenvector decomposition of $R_x = Q\Lambda Q^T$ and the fact that $\text{tr}(AB) = \text{tr}(BA)$ we can show that

$$\text{tr}(K(\infty)R_x) = \text{tr}(K'(\infty)\Lambda)\,.$$

Since we have derived a recursive expression for the vector $k'(n)$ in terms of this vector the above excess mean square error is given by

$$J(\infty) - J_{\min} = \sum_{i=0}^{M-1} \lambda_k k'_{ii}(\infty) = \lambda^T k'(\infty)\,.$$

To derive what the expression $\lambda^T k'(\infty)$ is. Assuming convergence of the vector $k'(n)$ to some limiting vector (say $k'(\infty)$ as $n \to \infty$) then Equation 30 for this steady-state vector requires

$$k'(\infty) = Fk'(\infty) + 4\mu^2 J_{\min}\lambda\,,$$

or solving for $k'(\infty)$

$$k'(\infty) = 4\mu^2 J_{\min}(I-F)^{-1}\lambda\,, \tag{31}$$

which is the books equation 7.4.44. Thus taking the λ^T of this expression we have we have an expression for the excess MSE given by

$$J_{\text{ex}}(\infty) = 4\mu^2 J_{\min}\lambda^T(I-F)^{-1}\lambda\,. \tag{32}$$

Since $J_{\min}$ depends on the problem considered (in regard to such things as the signal-to-noise of the problem) we define a missadjustment factor $\mathcal{M}$ that depends on the other properties of LMS algorithm

$$\mathcal{M} \equiv \frac{J_{\text{ex}}(\infty)}{J_{\min}} = 4\mu^2\lambda^T(I-F)^{-1}\lambda\,. \tag{33}$$

Matrix expressions like $(I-F)^{-1}$ can often be simplified using the Woodbury matrix identity

$$(A+CBC^T)^{-1} = A^{-1} - A^{-1}C(B^{-1}+C^TA^{-1}C)^{-1}C^TA^{-1}\,. \tag{34}$$

In this case our matrix F is given by

$$F = \text{diag}(f_0, f_1, \cdots, f_{M-1}) + 4\mu^2\lambda\lambda^T\,.$$

so that $I-F$ becomes

$$I - F = I - \text{diag}(f_0, f_1, \cdots, f_{M-1}) - 4\mu^2\lambda\lambda^T = F_1 + a\lambda\lambda^T\,.$$

Where in this last expression we have defined the matrix F_1 and the constant $a = -4\mu^2$. If we define a vector v as $v = \sqrt{a}\lambda$ we see that the Woodbury identity applied to the misadjustment factor $\mathcal{M}$ and in terms of F_1 and v becomes

$$\begin{aligned}\mathcal{M} &= -a\lambda^T(F_1+vv^T)^{-1}\lambda \\ &= -a\lambda^T\left[{F_1}^{-1} - {F_1}^{-1}v(I+v^T{F_1}^{-1}v)^{-1}v^T{F_1}^{-1}\right]\lambda\,.\end{aligned}$$

Now the inverse term *inside* the bracketed expression above is actually a scalar expression

$$(I + v^T {F_1}^{-1} v)^{-1} = \frac{1}{1 + v^T {F_1}^{-1} v} = \frac{1}{1 + a\lambda {F_1}^{-1} \lambda^T} .$$

Thus the misadjustment factor becomes since $vv^T = a\lambda\lambda^T$ that

$$\mathcal{M} = -a\lambda^T \left({F_1}^{-1} - \frac{{F_1}^{-1} v v^T {F_1}^{-1}}{1 + a\lambda^T {F_1}^{-1} \lambda} \right) \lambda = -a\lambda^T \left({F_1}^{-1} - \frac{a {F_1}^{-1} \lambda\lambda^T {F_1}^{-1}}{1 + a\lambda^T {F_1}^{-1} \lambda} \right) \lambda .$$

Combining the two terms in the parenthesis into one we find

$${F_1}^{-1} - \frac{a {F_1}^{-1} \lambda\lambda^T {F_1}^{-1}}{1 + a\lambda^T {F_1}^{-1} \lambda} = \frac{{F_1}^{-1} + a(\lambda^T {F_1}^{-1} \lambda) {F_1}^{-1} - a {F_1}^{-1} \lambda\lambda^T {F_1}^{-1}}{1 + a\lambda^T {F_1}^{-1} \lambda} .$$

If we take the product of this with λ^T on the left and λ on the right we see that the numerator simplifies to

$$\lambda^T {F_1}^{-1} \lambda + a(\lambda^T {F_1}^{-1} \lambda)(\lambda^T {F_1}^{-1} \lambda) - a\lambda^T {F_1}^{-1} \lambda \, \lambda^T {F_1}^{-1} \lambda = \lambda^T {F_1}^{-1} \lambda .$$

This term in the numerator, $\lambda^T {F_1}^{-1} \lambda$, since F_1 is a diagonal matrix can be computed as

$$\sum_{i=0}^{M-1} \frac{\lambda_i^2}{1 - f_i} = \sum_{i=0}^{M-1} \frac{\lambda_i^2}{4\mu\lambda_i - 8\mu^2\lambda_i^2} = \frac{1}{4\mu} \sum_{i=0}^{M-1} \frac{\lambda_i}{1 - 2\mu\lambda_i} .$$

While the denominator is given by

$$1 + a\lambda^T {F_1}^{-1} \lambda = 1 - \mu \sum_{i=0}^{M-1} \frac{\lambda_i}{1 - 2\mu\lambda_i}$$

Thus the entire fraction for $\mathcal{M}$ is given by

$$\mathcal{M} = \frac{-a\lambda {F_1}^{-1} \lambda}{1 + a\lambda^T {F_1}^{-1} \lambda} = \frac{\mu \sum_{i=0}^{M-1} \frac{\lambda_i}{1 - 2\mu\lambda_i}}{1 - \mu \sum_{i=0}^{M-1} \frac{\lambda_i}{1 - 2\mu\lambda_i}} , \tag{35}$$

as claimed in the book.

Problem Solutions

Problem 7.2.1 (the LMS algorithm for complex valued signals)

If we allow our process, $x(n)$, to be complex our inner product becomes the Hermitian inner product and we would compute for the autocorrelation matrix

$$R_x = E\{x(n)x^H(n)\}\,,$$

while for the cross-correlation p_{dx} in the complex case we would use

$$p_{dx} = d^*(n)x(n)\,,$$

and finally for the filter output $y(n)$ we would take

$$y(n) = \sum_{k=0}^{M-1} w_k^*(n)x(n-k)\,,$$

Thus setting up the expression for the error functional $J(w(n))$ we find

$$\begin{aligned}
J(w(n)) &= E\{|e(n)|^2\} = E\{e^*(n)e(n)\} \\
&= E\{(d(n) - w^*(n)x(n))^*(d(n) - w^*(n)x(n))\} \\
&= E\{\left(d(n) - \sum_{k=0}^{M-1} w_k^*(n)x(n-k)\right)^* \left(d(n) - \sum_{k=0}^{M-1} w_k^*(n)x^*(n-k)\right)\} \\
&= E\{\left(d^*(n) - \sum_{k=0}^{M-1} w_k(n)x^*(n-k)\right)\left(d(n) - \sum_{k=0}^{M-1} w_k^*(n)x^*(n-k)\right)\} \\
&= E\{d^*(n)d(n)\} \\
&- \sum_{k=0}^{M-1} E\{w_k(n)x^*(n-k)d(n)\} - \sum_{k=0}^{M-1} E\{d^*(n)w_k^*(n)x(n-k)\} \\
&+ \sum_{k=0}^{M-1}\sum_{k'=0}^{M-1} E\{w_k(n){w_{k'}}^*(n)x(n-k')x^*(n-k)\}\,.
\end{aligned}$$

With this we see that taking the derivative of this expression with respect to the filter coefficient w_k gives

$$\frac{\partial J(w(n))}{\partial w_k} = -E\{x^*(n-k)d(n)\} - E\{d^*(n)x(n-k)\}^*$$

$$
\begin{aligned}
&+ \sum_{k'=0}^{M-1} E\{w_{k'}^*(n)x(n-k')x^*(n-k)\} + \sum_{k=0}^{M-1} E\{w_k(n)x(n-k')x^*(n-k)\}^* \\
&= -2E\{x^*(n-k)d(n)\} \\
&+ \sum_{k'=0}^{M-1} E\{w_{k'}^*(n)x(n-k')x^*(n-k)\} + \sum_{k=0}^{M-1} E\{w_k^*(n)x(n-k)x^*(n-k')\} \\
&= -2E\{x^*(n-k)d(n)\} + 2E\left\{x^*(n-k)\sum_{k'=0}^{M-1} w_{k'}^*(n)x(n-k')\right\} \\
&= -2E\left\{x^*(n-k)\left(d(n) - \sum_{k'=0}^{M-1} w_{k'}^*(n)x(n-k')\right)\right\} \\
&= -2E\{x^*(n-k)e(n)\}\,.
\end{aligned}
$$

From this derivative the filter update equations, using the gradient decent algorithm are given by

$$
\begin{aligned}
w_k(n+1) &= w_k(n) - \mu\frac{\partial J}{\partial w_k} \\
&= w_k(n) + 2\mu E\{x^*(n-k)e(n)\}\,.
\end{aligned}
$$

If we use a point estimate to approximate the expectation in the above scheme we then take $E\{x^*(n-k)e(n)\} \approx x^*(n-k)e(n)$ and we arrive at the following filter update equations

$$
w_k(n+1) = w_k(n) + 2\mu x^*(n-k)e(n)\,.
$$

In vector form for the entire set of weights $\mathbf{w}$ this becomes

$$
\mathbf{w}(n+1) = \mathbf{w}(n) + 2\mu e(n)\mathbf{x}^*(n)\,.
$$

Here the notation $\mathbf{x}^*(n)$ means that we take (conjugated) the last M elements of the signal $\mathbf{x}$ starting at position n i.e. the vector

$$
(x^*(n), x^*(n-1), x^*(n-2), \cdots, x^*(n-M+1))^T\,.
$$

Problem 7.2.2 (the discrete-time representation of an AR system)

We are told that our AR system has poles at $0.85e^{\pm j\frac{\pi}{4}}$ and is driven by an input $v(n)$ given by discrete white noise. Then the z-transform of the *output*

of this system is given by the product of the system transfer function $H(z)$ and the z-transform of the input to the system. Since our AR system has poles at the given two points it has a system transfer function, $H(z)$ given by

$$\begin{aligned} H(z) &= \frac{1}{\left(1-0.85e^{j\frac{\pi}{4}}z^{-1}\right)\left(1-0.85e^{-j\frac{\pi}{4}}z^{-1}\right)} \\ &= \frac{1}{1-0.85\sqrt{2}z^{-1}+0.85^2z^{-2}} = \frac{1}{1-1.2021z^{-1}+0.7225z^{-2}}\,. \end{aligned}$$

So that the z-transform of our system output $X(z)$, assuming a z-transform of the system input given by $\sigma_v^2 V(z)$ is given by $X(z) = H(z)\sigma_v^2 V(z)$ or the input in terms of the output as

$$\sigma_v^2 V(z) = H(z)^{-1}X(x) = X(z) - 1.2021z^{-1}X(z) + 0.7225z^{-2}X(z)\,.$$

Taking the inverse z-transform of this expression and solving for $x(n)$ gives

$$x(n) = 1.2021x(n-1) - 0.7225x(n-2) + \sigma_v^2 v(n)\,,$$

for the AR(2) process the discrete output signal $x(n)$ satisfies.

Problem 7.4.1-7.4.5 (performance analysis of the LMS algorithm)

See the derivations in the section on the performance analysis of the LMS algorithm which are presented in the notes above.

Problem 7.4.6 (the expressions for L and $\mathcal{M}$)

Equation 7.4.54 in the book is given by

$$L = \sum_{i=0}^{M-1} \frac{\mu\lambda_i}{1-2\mu\lambda_i}\,.$$

Taking the derivative of L with respect to μ we obtain

$$\begin{aligned} \frac{\partial L}{\partial \mu} &= \sum_{i=0}^{M-1}\left(\frac{\lambda_i}{1-2\mu\lambda_i} - \frac{\mu\lambda_i}{(1-2\mu\lambda_i)^2}(-2\lambda_i)\right) \\ &= \sum_{i=0}^{M-1}\frac{\lambda_i}{(1-2\mu\lambda_i)^2} > 0\,. \end{aligned}$$

so L is an increasing function of μ. Now equation 7.4.55 is $\mathcal{M} = \frac{L}{1-L}$ so

$$\frac{\partial \mathcal{M}}{\partial L} = \frac{1}{1-L} - \frac{L(-1)}{(1-L)^2} = \frac{1}{(1-L)^2} > 0\,,$$

so $\mathcal{M}$ is an increasing function of L.

Problem 7.4.8 (the effect of the initial guess at the weights)

To study the transient behavior means to study how $J(n)$ trends to $J_{\min}$ as $n \to \infty$. Recalling the decomposition of J in terms of the eigenvalues of R_x we have

$$J(n) = J_{\min} + \sum_{k=0}^{M-1} \lambda_k (1 - \mu' \lambda_k)^{2n} \xi'^2_k(0)\,,$$

with $\xi'(n) = Q^T \xi(n) = Q^T(w(n) - w^o)$. With these expression we see that the initial scaled coordinate ξ is given by $\xi'(0) = Q^T(w(0) - w^o)$ so if we take an initial guess at our filter coefficients of zero i.e. $w(0) = 0$ then we have

$$\xi'(0) = -Q^T w^o\,,$$

and the equation for $J(n)$ then becomes

$$\begin{aligned} J(n) &= J_{\min} + \sum_{k=0}^{M-1} \lambda_k (1 - \mu' \lambda_k)^{2n} (Q^T w^o)_k{}^2 \\ &= J_{\min} + \sum_{k=0}^{M-1} \lambda_k (1 - \mu' \lambda_k)^{2n} w'^o{}_k{}^2 \end{aligned}$$

with w'^o defined as $w'^o = Q^T w^o$. Now how fast $J(n)$ will converge to $J_{\min}$ will depend on how large the products $\lambda_k (w'^o)_k^2$ are. We know that for convergence we must have

$$|1 - \mu' \lambda_k| < 1\,,$$

so if we "tilt" the components of w'^o to be close to zero for small eigenvalues the convergence will then depend primarily on the large eigenvalues and should be fast since the corresponding time constants are small. If on the other hand the values of w'^o happen to be such that they are near zero for the large eigenvalues, the convergence will be dominated by the small eigenvalues and result in slow convergence.

Problem 7.4.9 (the LMS algorithm using $E\{e^2(n)\} \approx e^2(n)$)

For this problem we consider the cost functional $J(n)$ defined as

$$J(n) = e^2(n) = (d(n) - w^T(n)x(n))^2 .$$

Then the gradient decent method applied to $J(n)$ results in updating $w(n)$ with

$$w(n+1) = w(n) - \mu \frac{\partial J(w)}{\partial w} ,$$

We find for the first derivative of $J(n)$ the following

$$\begin{aligned} \frac{\partial J}{\partial w} &= 2(d(n) - w^T(n)x(n))(-x(n)) \\ &= -2e(n)x(n) , \end{aligned}$$

so the gradient decent algorithm for $w(n)$ then becomes

$$w(n+1) = w(n) + 2\mu e(n)x(n) .$$

Problem 7.4.10 (a linear system driven by a Bernoulli sequence)

Part (a): For the given specification of the filter coefficient weights $h(k)$ we have a discrete time representation of the system output given by

$$x(n) = \sum_{k=1}^{3} h(k)s(n-k) + v(n) .$$

Using this explicit representation of $x(n)$ in terms of $h(n)$ and $s(n)$ we can compute the autocorrelation for $x(n)$ using its definition. We find

$$\begin{aligned} r_x(m) &= E\{x(n)x(n-m)\} \\ &= E\{\left(\sum_{k=1}^{3} h(k)s(n-k) + v(n)\right)\left(\sum_{k'=1}^{3} h(k')s(n-m-k') + v(n-m)\right)\} \\ &= \sum_{k=1}^{3}\sum_{k'=1}^{3} h(k)h(k')E\{s(n-k)s(n-m-k')\} + E\{v(n)v(n-m)\} . \end{aligned}$$

Where we have used the fact that $s(n)$ and $v(n)$ are uncorrelated and have zero mean. The expectation of $s(n)$ and $v(n)$ against themselves can be

computed as

$$\begin{aligned} E\{s(n-k)s(n-m-k')\} &= \sigma_s^2\delta(-k+k'+m) \\ E\{v(n)v(n-m)\} &= \sigma_v^2\delta(m)\,. \end{aligned}$$

Using these two results the above expression for $r_x(m)$ becomes

$$\begin{aligned} r_x(m) &= \sigma_s^2 \sum_{k=1}^{3}\sum_{k'=1}^{3} h(k)h(k')\delta(-k+k'+m) + \sigma_v^2\delta(m) \\ &= \sigma_s^2 \sum_{k=1}^{3} h(k)h(k-m) + \sigma_v^2\delta(m) \\ &= \sigma_s^2 r_h(m) + \sigma_v^2\delta(m)\,. \end{aligned}$$

Now computing each term $r_x(m)$ assuming a length five ($M = 5$ filter) requires us to evaluate $r_x(m)$ for $m = 1, 2, 3, 4, 5$. We find

$$\begin{aligned} r_x(0) &= 1r_h(0) + \sigma_v^2 = 1\sum_{k=1}^{3} h(k)^2 + 0.01 \\ &= 0.22^2 + 1^2 + 0.22^2 + 0.01 = 1.1068 \\ r_x(1) &= 1\sum_{k=1}^{3} h(k)h(k-1) = h(1)h(0) + h(2)h(1) + h(3)h(2) \\ &= 0 + 1(0.22) + (0.22)1 = 0.44 \\ r_x(2) &= \sum_{k=1}^{3} h(k)h(k-2) = h(3)h(1) = 0.22^2 = 0.0484\,, \end{aligned}$$

and $r_x(m) = 0$ for $m \geq 3$. Thus we get a matrix R_x given by

$$R_x = \begin{bmatrix} 1.1068 & 0.4400 & 0.0484 & 0 & 0 \\ 0.4400 & 1.1068 & 0.4400 & 0.0484 & 0 \\ 0.0484 & 0.4400 & 1.1068 & 0.4400 & 0.0484 \\ 0 & 0.0484 & 0.4400 & 1.1068 & 0.4400 \\ 0 & 0 & 0.0484 & 0.4400 & 1.1068 \end{bmatrix}$$

From which we find that $\frac{\lambda_{\max}}{\lambda_{\min}} = 4.8056$ so that from stability considerations we require

$$\mu < \frac{1}{3\mathrm{tr}(R_x)} = 0.0602\,.$$

Chapter 8 (Variations of LMS algorithms)

Problem 8.1.1 (the step-size in the error sign algorithm)

The error sign algorithm is given by

$$\begin{aligned} w(n+1) &= w(n) + 2\mu \text{sign}(e(n))x(n) \\ &= w(n) + \frac{2\mu}{|e(n)|}\text{sign}(e(n))|e(n)|x(n) \\ &= w(n) + \frac{2\mu}{|e(n)|}e(n)x(n)\,, \end{aligned}$$

which equals the normal LMS algorithm with a variable (dependent on n) step size parameter $\mu'(n) = \frac{\mu}{|e(n)|}$. Since $|e(n)| \to 0$ as $n \to +\infty$ if we have convergence we see that $\mu'(n) \to +\infty$, so μ must be very small in the error sign algorithm for convergence.

Problem 8.2.1 (a derivation of the normalized LMS algorithm)

We want to pick $\mu(n)$ in the generalized LMS recursion algorithm

$$w(n+1) = w(n) + 2\mu(n)e(n)x(n)\,.$$

to minimize the a posteriori error

$$e_{\text{ps}}(n) = d(n) - w(n+1)^T x(n)\,.$$

when we put $w(n+1)$ into the above we get

$$\begin{aligned} e_{\text{ps}}(n) &= d(n) - w(n)^T x(n) - 2\mu(n)e(n)x^T(n)x(n) \\ &= (1 - 2\mu(n)x^T(n)x(n))e(n)\,. \end{aligned}$$

Then

$$\begin{aligned} \frac{\partial e_{\text{ps}}(n)}{\partial \mu(n)} &= 2e_{\text{ps}}(n)\frac{\partial e_{\text{ps}}(n)}{\partial \mu(n)} \\ &= 2e_{\text{ps}}(n)(-2x^T(n)x(n))\,, \end{aligned}$$

or when we set this equal to zero we have the equation

$$(1 - 2\mu(n)x^T(n)x(n)) = 0 \quad \text{or} \quad \mu(n) = \frac{1}{2x^T(n)x(n)}\,.$$

so the LMS algorithm becomes

$$w(n+1) = w(n) + \frac{e(n)x(n)}{x^T(n)x(n)}\,,$$

which is equation 8.2.3 in the book.

Problem 8.4.1 (a derivation of the leaky LMS algorithm)

If we take an error functional $J(n)$ given by

$$\begin{aligned} J(n) &= e^2(n) + \gamma w^T(n)w(n) \\ &= (d(n) - w^T(n)x(n))^2 + \gamma w^T(n)w(n)\,, \end{aligned}$$

then recalling that the LMS algorithm is given by the gradient decent algorithm applied to $J(n)$ as

$$w(n+1) = w(n) - \mu\frac{\partial J(w(n))}{\partial w}\,. \tag{36}$$

Computing the w derivative we find

$$\begin{aligned} \frac{\partial J(w(n))}{\partial w} &= -2e(n)x(n) + \gamma w(n) + \gamma w(n) \\ &= 2(-e(n)x(n) + \gamma w(n))\,. \end{aligned}$$

When we put this into Equation 36 above we obtain

$$\begin{aligned} w(n+1) &= w(n) + 2\mu(e(n)x(n) - \gamma w(n)) \\ &= (1 - 2\mu\gamma)w(n) + 2\mu e(n)x(n)\,, \end{aligned}$$

which is equation 8.4.9 or the leaky LMS algorithm.

Problem 8.5.1 (J for the linearly constrained LMS)

We desire to derive equation 8.5.5 a representation of the error functional $J_c(n)$ in terms of centered coordinates $\xi(n)$. Now in the linearly constrained LMS algorithm we have a constraint functional J_c defined as

$$J_c = E\{e^2(n)\} + \lambda(c^T w - a)\,,$$

with the error $e(n)$ given by $e(n) = d(n) - w^T(n)x(n)$. From the discussion earlier and problem 5.2.1 on page 21 we know that

$$E\{e^2(n)\} = J_{\min} + \xi^T R_x \xi \,.$$

with $\xi = w(n) - w^o$ and $w^o = {R_x}^{-1} p_{dx}$ the optimal Wiener filtering weights. Then to transform the additional Lagrange multiplier term in our cost functional, $\lambda(c^T w - a)$, into the same form we write it as

$$\begin{aligned} \lambda(c^T(w - w^o + w^o) - a) &= \lambda(c^T\xi + c^T w^o - a) \\ &= \lambda(c^T\xi + a') \,, \end{aligned}$$

where we have defined a' as $a' = c^T w^o - a$ and we then obtain the following centered representation of the error criterion

$$J_c = J_{\min} + \xi^T R_x \xi + \lambda(c^T\xi - a') \,,$$

the same expression as requested.

Problem 8.7.1 (the DFT of the filter coefficients $w(n)$)

We have the discrete Fourier transform of the filter output given by $Y_i(k) = W_{i,k} X_i(k)$ for $k = 0, 1, 2, \cdots, M-1$ and we measure how well our filtered result matches the desired signal as the difference between the Fourier transform of the desired signal $D_i(k)$ and the filtered output $Y_i(k)$. That is

$$\begin{aligned} E_i(k) &= D_i(k) - Y_i(k) \\ &= D_i(k) - W_{i,k} X_i(k) \,. \end{aligned}$$

Then with the LMS iteration scheme for the filter coefficients given by equation 8.7.7 we find the following recursion equation for $W_{i+1,k}$

$$\begin{aligned} W_{i+1,k} &= W_{i,k} + 2\mu X_i(k)^* E_i(k) \\ &= W_{i,k} + 2\mu X_i(k)^* (D_i(k) - W_{i,k} X_i(k)) \\ &= W_{i,k} + 2\mu X_i(k)^* D_i(k) - 2\mu W_{i,k} |X_i(k)|^2 \\ &= (1 - 2\mu |X_i(k)|^2) W_{i,k} + 2\mu D_i(k) X_i(k)^* \,, \end{aligned}$$

which is equation 8.7.10.

Problem 8.7.2 (the steady-state value for $W_{i,k}$)

We desire to verify the recursive update equation 8.7.12 for the expected Fourier transform filter coefficients. The equation that these filter coefficients $W_{i,k}$ satisfy is given by (repeated here for convenience)

$$E\{W_{i+1,k}\} = (1 - 2\mu E\{|X_i(k)|^2\})E\{W_{i,k}\} + 2\mu E\{D_i(k)X_i^*(k)\}\,. \quad (37)$$

We will take the z-transform of the above equation with respect to the index i, under the assumption that that the two expectations $E\{|X_i(k)|^2\}$ and $E\{D_i(k)X_i^*(k)\}$ are *independent* of i. Recalling the z-transform identities

$$\begin{aligned} \mathcal{Z}\{1\} &= \frac{z}{z-1} \\ \mathcal{Z}\{x(n+1)\} &= z\left(\mathcal{Z}\{x(n)\} - x(0)\right)\,, \end{aligned}$$

we can take the z-transform of Equation 37 to get

$$z(W_k(z) - W_{0,k}) = (1 - 2\mu E\{|X_i(k)|^2\})W_k(z) + \frac{2\mu z E\{D_i(k)X_i^*(k)\}}{z-1}\,.$$

Solving this for $W_k(z)$ we find

$$W_k(z) = \frac{2\mu z E\{D_i(k)X_i^*(k)\}}{(z-1)(z-1+2\mu E\{|X_i(k)|^2\})}\,.$$

Using this expression we can call on the final value theorem requires to find $E\{W_k^\infty\}$ where we see that

$$\begin{aligned} E\{W_k^\infty\} &= \lim_{z\to+1}((z-1)W_k(z)) = \lim_{z\to+1}\frac{2\mu z E\{D_i(k)X_i^*(k)\}}{(z-1+2\mu E\{|X_i(k)|^2\})} \\ &= \frac{E\{D_i(k)X_i^*(k)\}}{E\{|X_i(k)|^2\}}\,, \end{aligned}$$

the desired expression.

Problem 8.7.3 (the solution for $E_i(k)$)

Recalling the steady state Fourier error given by $E_i(k) = E\{W_{i,k}\} - E\{W_k^\infty\}$. When we subtract $E\{W_k^\infty\}$, which is equivalent to $\frac{E\{D_i(k)X_i^*(k)\}}{E\{|X_i(k)|^2\}}$, from equation 8.7.11 the recursive expression for the error term we find

$$E_{i+1}(k) = (1 - 2\mu E\{|X_i(k)|^2\})E\{W_{i,k}\}$$

$$
\begin{aligned}
&+ \quad 2\mu E\{D_i(k)X_i^*(k)\} - \frac{E\{D_i(k)X_i^*(k)\}}{E\{|X_i(k)|^2\}} \\
&= \quad (1 - 2\mu E\{|X_i(k)|^2\})E\{W_{i,k}\} \\
&- \quad (1 - 2\mu E\{|X_i(k)|^2\})\frac{E\{D_i(k)X_i^*(k)\}}{E\{|X_i(k)|^2\}} \\
&= \quad (1 - 2\mu E\{|X_i(k)|^2\})(E\{W_{i,k}\} - E\{W_k^\infty\}) \\
&= \quad (1 - 2\mu E\{|X_i(k)|^2\})E_i(k)\,,
\end{aligned}
$$

which is equation 8.7.15.

Chapter 9 (Least squares and recursive least-squares signal processing)

Additional Notes

For the weighted cost function J_G given by

$$J_G = e^T G e = (y - Xw)^T G (y - X)\,,$$

we have on expanding this quadratic that

$$J_G = y^T G y - y^T G X w - w^T X^T G y + w^T X^T G X w\,.$$

Then the derivative of this with respect to w is given by

$$\begin{aligned}\frac{\partial J_W}{\partial w} &= -(y^T G X)^T - X^T G y + (X^T G X + (X^T G X)^T) w \\ &= -2X^T G y + 2X^T G X w\,.\end{aligned}$$

Setting this expression equal to zero and solving for w we find

$$w = (X^T G X)^{-1} X^T G y\,. \tag{38}$$

which is equation 9.2.33.

Exercise Solutions

Problem 9.2.1 (a derivation of the least-squares solution)

From equation 9.2.15 in the book we have

$$\begin{aligned}J(\mathbf{w}) &= (\mathbf{d} - \mathbf{X}\mathbf{w})^T (\mathbf{d} - \mathbf{X}\mathbf{w}) \\ &= \mathbf{d}^T\mathbf{d} - 2\mathbf{p}^T\mathbf{w} + \mathbf{w}^T\mathbf{R}\mathbf{w}\,.\end{aligned}$$

Now we have the following identities of matrix derivatives so some common scalar forms

$$\frac{\partial(\mathbf{p}^T\mathbf{w})}{\partial\mathbf{w}} = \mathbf{p}$$

$$\frac{\partial(\mathbf{w}^T R\mathbf{w})}{\partial\mathbf{w}} = (\mathbf{R} + \mathbf{R}^T)\mathbf{w}\,.$$

Thus

$$\nabla J(\mathbf{w}^*) = -2\mathbf{p} + 2\mathbf{R}\mathbf{w}^* = 0 \quad \text{or} \quad \mathbf{w}^* = \mathbf{R}^{-1}\mathbf{p},,$$

With this expression the minimum sum of square errors is given by

$$\begin{aligned} J_{\min} = J(\mathbf{w}^*) &= \mathbf{d}^T\mathbf{d} - 2\mathbf{p}^T\mathbf{R}^{-1}\mathbf{p} + \mathbf{p}^T\mathbf{R}^{-1}\mathbf{R}\mathbf{R}^{-1}\mathbf{p} \\ &= \mathbf{d}^T\mathbf{d} - \mathbf{p}^T\mathbf{R}^{-1}\mathbf{p}\,. \end{aligned}$$