Contents

Chapter 9

Time-Varying Channels and Adaptive Transmission Systems

Time-variation in transmission corresponds to time-dependent change in either or both of the basic channel impulse response h and the noise probability distribution p_n. The criteria for best performance then also can be time-varying, and design for best (or even just good) performance may need alteration with respect to best stationary-channel design. This chapter augments earlier chapters' basic transmission theory to consider time-variation.

There are essentially two approaches to time-varying channels. Where time-variation is sufficiently slow, the existing theories of Chapters 1-6 (and also Chapters 8 to 15 of Volume II) apply well over continuous blocks of time where the time-variation is minimal. From time block to time block, the design will require either or both of the transmitter and/or the receiver to correspondingly vary in time, or equivalently to **track** *the time variation*. However, within each block of time, the design proceeds as in the stationary case. This type of situation sometimes occurs in "fixed-line" channels where the medium is copper or fiber and essentially a "wire," where at least the channel impulse response is somewhat constant. These kinds of designs are often called **adaptive.** There may be recursive ways that incrementally adapt the receiver structure to track the channel changes from block to block (or even within the block). Sections 9.1 and 9.2 address methods that correspond to such adaptive channel learning and tracking. The initial learning phase is often called **training** in adaptive transmission systems. Training and tracking for multi-carrier systems usually correspond to DMT's use. While wire-line systems are often perceived as time-invariant, they may not have stationary noise. Electrical or optical transmission media themselves often vary very slowly with time, but the noises they experience rarely are stationary (even if often Gaussian, the variance of this Gaussian noise changes). Such "intermittent noise" can be induced by various man-made external noise sources (including sometimes, but certainly not limited to, other communication systems trying to use the same link or neighboring cross-talking links).

Where **tracking** would not be helpful because the channel varies too quickly, good transmission designs instead use **statistical** approaches that characterize the range of possible channel conditions and their corresponding probabilities of occurrence. Such statistical characterization and the corresponding designs can apply to both wireless transmission and wireline channels. As noted above, wireline systems can have intermittent noise. Wireless channels can see significant change in both the channel impulse response and the noise spectrum. The statistical design must instead be **robust** to the range of variation expected instead of attempting (only) to track. These robust designs often employ Chapter 3's diversity (and the related Chapter 10 and 11's coding, which is a form of diversity) to mitigate the impact of the worst-case situations. Many practical systems today combine some degree of adaptive design with robust statistical design, depending on the different rates of variation of the channel impulse response and/or noise distribution. Section **??** will address statistical modeling and the corresponding designs. The multi-carrier system that corresponds to robust design for statistical channels is Coded-OFDM. These Coded-OFDM systems are a special case of DMT in that they use group loading (see Section 4.3) as opposed to individual dimension-by-dimension loading. Section **??** will use and extend Chapter 5's CDEF and GDFE interpretations of Chapter 5 to show that Coded-OFDM systems will perform

at near-canonical levels **for the given fixed bands they use** as long as the applied code spreads sufficient redundancy and gain to that given band, an effect shown to be increasingly true as $\Gamma \to 0$ dB. Indeed DMT's loading adds nothing but simplification of the best detectors when the usable band or tones cannot be adapted. Most wireless systems today experience performance levels considerably below static design's theoretical limits (unlike fixed wireline channels), largely because the time-variation creates optimum bandwidths that may differ significantly from those fixed in wireless. However, changing the spectrum dynamically may simply be impossible if the channel is varying too quickly, so a robust statistical design is the best option.

In both cases, whether adaptively tracking slow changes or statistically accommodating a range of channel possibilities, there is a certain probability that the system cannot sustain an attempted data rate at the desired probability of symbol error, which is known as the "outage probability" for that data rate. This outage probability[1] is often a better indication of the users' satisfaction than is a symbol-error or bit-error probability alone, and is discussed in Section **??**.

Section 9.3 focuses more on nonlinear channels and the use of neural nets to both model channels, and possibly also to model the receivers and even possibly the transmitters. Some fundamental issues on the amount of training data and the depth and number of to-be-learned parameters are also discussed. Learning of neural-network models as well as the related Hidden Markov Models are also addressed.

Section 9.4 provides adaptive algorithms for single-carrier finite-impulse-response systems that may still be used and require adaptive "equalization" or other FIR functions, wherever they may still be in use. The multi-dimensional transmission methods often avoid many of the issues in single-carrier adaptive filtering because each dimension is more or less adapted independently. However, Section 9.4 summarizes algorithms that do exist for direct adaptation of an ensemble of dimensions.

[1] Many early 21st-century cell-phone-using outages occur when the transmitter and/or receiver design cannot sustain the data rate attempted at the desired probability of symbol error. An example of an outage would be the "dropped call" by a cellphone or perhaps "jerkiness" or "pixelation" in a video image on a tablet.

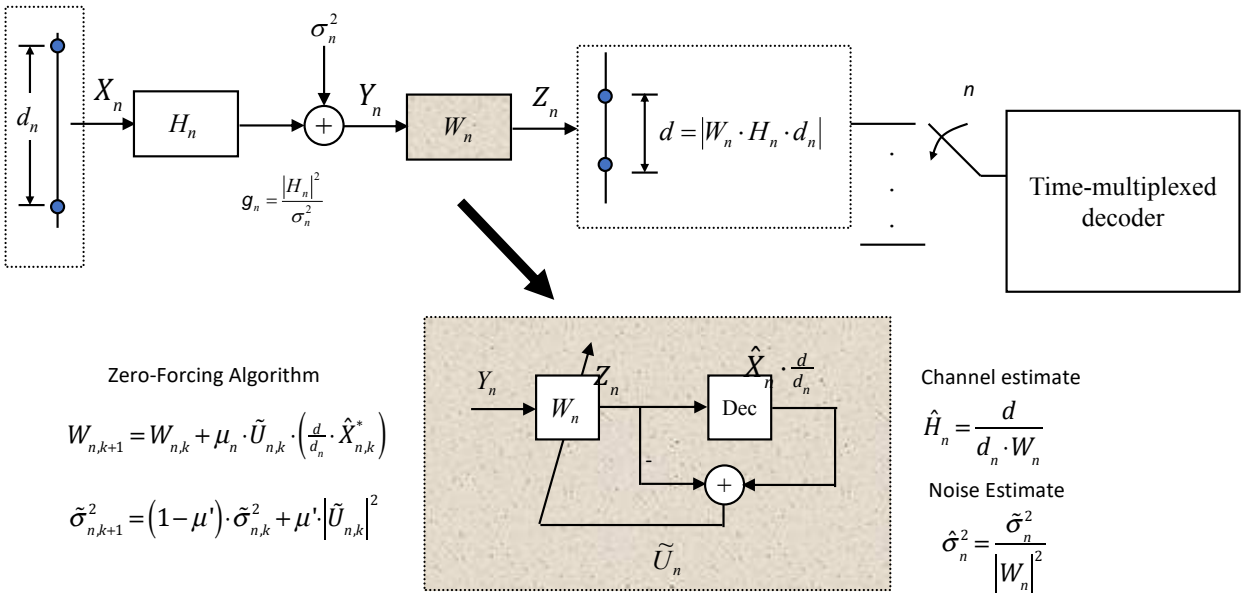

Figure 9.1: Gain Normalizer Revisted.

9.1 Per-Dimensional Channel Identification and Tracking

Generally, channel-identification methods measure the channel pulse response and noise power spectral density, or their equivalents (that is direct measurement of SNR's without measuring pulse response and noise separately). The channel-dependent designs of Chapters 3-5 may then be applied to the currently identified channel. Per-dimensional channel identification directly estimates signal and noise parameters for each of the subchannels/dimensions. A DMT/OFDM-specific method for channel identification appears in this section. Training usually occurs for a known transmitted sequence. DMT training can separately identify first the pulse-response DFT values at the subchannel center frequencies used by a DMT/OFDM system; then DMT design can estimate the noise-sample variances at these same frequencies. The presented training method concludes by computing the SNR's from the results of gain and noise estimation. The measurement of the pulse-response DFT values is called **gain estimation**[2], while the measurement of the noise variances is called **noise estimation**. This section's particular method is in common use because it reuses the same hardware/software as steady-state transmission with DMT; however, it is neither optimum nor necessarily the best choice in terms of performance for limited data observation. It is easy to implement. Fortunately, a lengthy training interval is often permitted for DMT-transmission applications, and this method will become very accurate with sufficiently long training. The presented method is thus usually not as appropriate for wireless lower-performing C-OFDM systems because it takes too long, and OFDM does not require the accuracies in SNR estimation that DMT requires.

C-OFDM sometimes requires only gross (even as "gross" as just good or bad) estimation of SNR's. Bad SNR dimensions are de-emphasized in decoding while decoding instead relies more heavily on good-SNR dimensions. Coded-OFDM has reduced performance because it uses all subchannels even if optimized loading would not. Tracking is usually sufficient for Coded-OFDM, although some gross training can also be implemented by this section's methods, but simply with much shorter training intervals and far less accuracy. These issues are addressed in Section **??**.

The channel noise is an undesired disturbance in gain estimation, and this noise needs to be averaged in determining subchannel gains as in illustrated in the gain normalizer of Section 4.3 that is revisited in Figure 9.1. The Gain Normalizer (or FEQ as it is sometimes called) directly computes a quantity

[2]This gain is usually a complex gain meaning that both subchannel gain and phase information are estimated.

proportional to the inverse channel SNR g_n for each dimension (real or complex). As such it is used for tracking. The algorithm listed in Figure 9.1 is a scalar version of Section 3.11's zero-forcing equalization algorithm. If decisions are usually correct, this system will cause all dimensions' equalized decision regions to appear the same (allowing a common scaling to be used in subsequent decoding) and as well to provide a critical quantity for Section 4.3's dynamic channel loading. Because this receiver component equalizes constellation-point spacing to each tone's decoder, it is usually called a **Frequency Equalizer (FEQ)**.

The FEQ may also be used for initial training, but requires Subsection 9.1.1's "gear-shifting." Initially in Subsections 9.1.1 and 9.1.2, both gain and noise estimation are respectively addressed. Usually, gain training occurs first. When gain estimation is complete, the estimated (noiseless) channel output can be subtracted from the actual channel output to form a channel-noise estimate. This noise's spectrum can then be estimated as in Subsection 9.1.2. Good gain estimation ensures that any residual gain-estimation error is sufficiently small that it does not significantly corrupt the subsequent noise-spectral-estimation process. Such small error can require large dynamic range in estimating the subchannel gains.

The channel-SNR g_n can be computed from the known subchannel input distance, d_n, the known multichannel normalized output distance d, and the noise estimate $\tilde{\sigma}_n^2$ as described in Subsection 9.1.2. During training, or in C-OFDM systems, it is possible to send the same constellation on all subchannels, thus essentially forcing $d_n = d$, and thus simplifying the g_n calculation to

$$g_n = \frac{1}{\tilde{\sigma}_n^2} \quad . \tag{9.1}$$

Section 4.3 did not address selection of the gain constants μ and μ' for the FEQ's updating loops. These values will be provided shortly, and a more direct consideration of computation of g_n will be considered for the special case of initialization. The choice of μ then follows from the general problem of gain estimation in Subsection 9.1.1, which is the estimation of the subchannel signal gain. The choice of the second gain constant μ' similarly follows from the general problem of noise estimation in Subsection 9.1.2, which is the estimation of the unscaled subchannel noise.

The presented methods will allow any imperfections (ISI, colored noise correlation, etc) to be simply included in the resultant measured g_n's. "These g_n will be what the receiver measured" as far as loading is concerned.

9.1.1 Gain-Estimation Training

Figure 9.2 illustrates gain estimation during initialization. An initial known training sequence, x_k, is periodic with period $\bar{N}$ equal to[3] or greater than the number of significant coefficients in the unknown channel pulse response p_k ("periodic" can mean use of the cyclic prefix of Section 4.6). The channel output sequence is

$$y_k = x_k * p_k + u_k \tag{9.2}$$

where u_k is an additive noise signal assumed to be uncorrelated with x_k. The channel distortion sequence is denoted by u_k, instead of n_k, to avoid confusion of the frequency index n with the noise sequence and because practical implementations may have some residual signal elements in the distortion sequence u_k (even though independence from x_k is assumed for simplification of mathematics).

Gain estimation constructs an estimate of the channel pulse response, $\hat{p}_k$, by minimizing the mean square of the error

$$e_k = y_k - \hat{p}_k * x_k \quad . \tag{9.3}$$

Ideally, $\hat{p}_k = p_k$.

[3] Recall from Chapter 5 that to avoid confusion

$$\bar{N} \triangleq \begin{cases} N & \text{real basedband systems} \\ \frac{N}{2} & \text{complex baseband-equivalent systems} \end{cases}$$

and N remains the number of real dimensions.

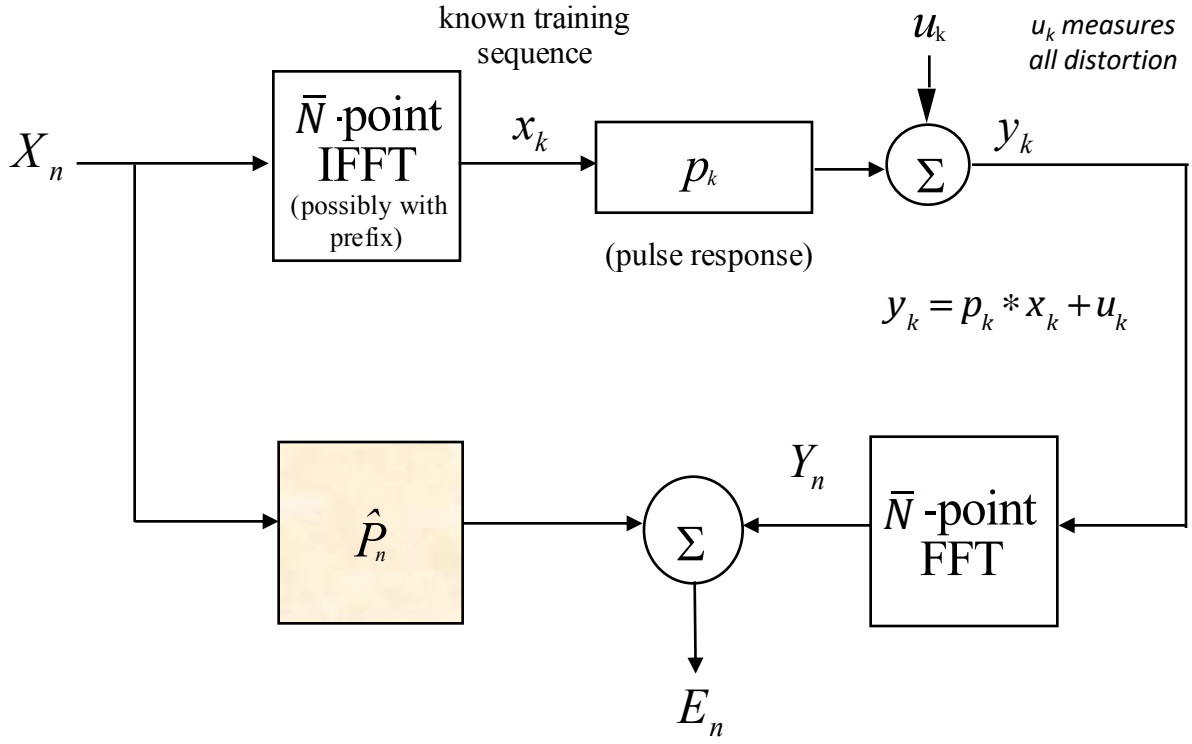

Figure 9.2: Channel-identification basics and terminology.

The Discrete Fourier Transform, DFT, of x_k is again

$$X_n = \frac{1}{\sqrt{\bar{N}}} \cdot \sum_{k=0}^{N-1} x_k \cdot e^{-\jmath \frac{2\pi}{\bar{N}} kn} \quad . \tag{9.4}$$

When x_k is periodic[4] with period $\bar{N}$ samples, the DFT samples X_n are also periodic with period $\bar{N}$ and constitute a complete representation for the periodic time sequence x_k over all time. The N-point DFT's of y_k, u_k, $\hat{p}_k$, p_k, and e_k are Y_n, U_n, $\hat{P}_n$, P_n, and E_n, respectively.

Since x_k is periodic, then

$$Y_n = P_n \cdot X_n + U_n \quad . \tag{9.5}$$

The corresponding frequency-domain error is then

$$E_n = Y_n - \hat{P}_n \cdot X_n \quad n = 0, ..., \bar{N} \quad . \tag{9.6}$$

(Recall that subchannels 0 and $N/2$ are one-dimensional in a real baseband system, while complex – 2 real-dimensions – in a complex baseband system with FFT size $\bar{N}$.) A vector of samples will contain one period of samples of any of the periodic sequences in question, for instance

$$\boldsymbol{x} = \begin{bmatrix} x_{N-1} \\ \vdots \\ x_1 \\ x_0 \end{bmatrix} \quad . \tag{9.7}$$

Then the FFT relations

$$\boldsymbol{X} = \mathcal{Q}\boldsymbol{x} \quad \boldsymbol{x} = \mathcal{Q}^* \boldsymbol{X} \tag{9.8}$$

[4]The periodicity can also be emulated by a cyclic prefix of length ν samples at the beginning of each transmitted block.

follow easily, where Q is the matrix describing the DFT ($q_{\bar{N}-1-i,\bar{N}-1-k} = e^{-j\frac{2\pi}{\bar{N}}ki}$) and $QQ^* = Q^*Q = I$. Similarly, $\boldsymbol{y}$, $\boldsymbol{Y}$, $\boldsymbol{e}$, $\boldsymbol{E}$, $\boldsymbol{p}$, $\boldsymbol{P}$, $\hat{\boldsymbol{p}}$, $\hat{\boldsymbol{P}}$, $\boldsymbol{u}$ and $\boldsymbol{U}$ as corresponding time-domain (lower case) and frequency-domain (upper case) vectors. For aperiodic sequences u_k, and thus aperiodic y_k and e_k, a block-time index l differentiates between different channel-output blocks (symbols) of $\bar{N}$ samples, i.e. $\boldsymbol{y}_l$.

The MSE for any estimate $\boldsymbol{p}$ is then

$$\begin{aligned} \text{MSE} &= E\left\{|e_k|^2\right\} && (9.9)\\ &= \frac{1}{\bar{N}} E\left\{\|\boldsymbol{e}\|^2\right\} = \frac{1}{\bar{N}} \sum_{k=0}^{\bar{N}-1} E\left\{|e_k|^2\right\} && (9.10)\\ &= \frac{1}{\bar{N}} E\left\{\|\boldsymbol{E}\|^2\right\} = \frac{1}{\bar{N}} \sum_{n=0}^{\bar{N}} E\left\{|E_n|^2\right\} \quad . && (9.11) \end{aligned}$$

The tap error is $\delta_k \triangleq p_k - \hat{p}_k$ in the time domain and $\Delta_n \triangleq P_n - \hat{P}_n$ in the frequency domain. The autocorrelation sequence for any discrete-time sequence x_k is defined by

$$r_{xx,k} = E\left\{x_m x^*_{m-k}\right\} \tag{9.12}$$

with corresponding discrete power spectrum

$$R_{xx,n} = \mathcal{Q} \cdot r_{xx,k} \quad . \tag{9.13}$$

The norm tap error (NTE) is defined as

$$\begin{aligned} \text{NTE} &\triangleq E\left\{\|\boldsymbol{p} - \hat{\boldsymbol{p}}\|^2\right\} = E\left\{\|\boldsymbol{\delta}\|^2\right\} && (9.14)\\ &= E\left\{\|\boldsymbol{P} - \hat{\boldsymbol{P}}\|^2\right\} = E\left\{\|\boldsymbol{\Delta}\|^2\right\} && (9.15)\\ &= \sum_{n=0}^{\bar{N}-1} |P_n - \hat{P}_n|^2 = \sum_{k=0}^{\bar{N}-1} |p_k - \hat{p}_k|^2 \quad . && (9.16) \end{aligned}$$

When the estimate $\hat{\boldsymbol{p}}$ equals the channel pulse response $\boldsymbol{p}$, then

$$e_k = u_k \tag{9.17}$$

and

$$r_{ee,k} \triangleq E\left[e_m \cdot e^*_{m-k}\right] = r_{uu,k} \;\; \forall\, k = 0, ..., \bar{N} - 1 \quad , \tag{9.18}$$

where $r_{ee,k}$ is the autocorrelation function of e_k, and $r_{uu,k}$ is the autocorrelation function of u_k. The MMSE is then

$$MMSE = r_{ee,0} = \sigma_u^2 \quad , \tag{9.19}$$

the mean-square value of the channel noise samples, or $\sigma_u^2 = \frac{\mathcal{N}_0}{2}$ per dimension for the AWGN channel. In practice, $\hat{p}_k \neq p_k$, MSE > MMSE, and the excess MSE[5] is

$$\text{EMSE} \triangleq \text{MSE} - \sigma_u^2 \quad . \tag{9.20}$$

The overall channel SNR is

$$\text{SNR} = \frac{r_{xx,0} \cdot \|\boldsymbol{p}\|^2}{r_{uu,0}} \tag{9.21}$$

and the SNR of the n^{th} channel is

$$\text{SNR}_n = \frac{R_{xx,n} \cdot |P_n|^2}{R_{uu,n}} \quad . \tag{9.22}$$

[5]Toeplitz distribution applies if $\bar{N}$ is large or the input and noise are periodic, and thus the error E_n and noise/distribution U_n are uncorrelated so EMSE> 0. Aberrations lead to EMSE < 0 would occur only if N is too small and the noise is highly correlated.

Definition 9.1.1 (Gain-Estimation SNR) The gain-estimation SNR is a measure of the true channel-output signal power on any subchannel to the excess MSE in that same channel (which is caused by $\hat{\boldsymbol{p}} \neq \boldsymbol{p}$), and is

$$SNR_{\hat{P},n} \stackrel{\Delta}{=} \frac{R_{xx,n} \cdot |P_n|^2}{R_{ee,n} - R_{uu,n}} \quad . \tag{9.23}$$

EXAMPLE 9.1.1 (Accuracy) $SNR_{\hat{P},n}$ should be 30-60 dB for good gain estimation. To understand this requirement on $SNR_{\hat{P},n}$, consider the case on a subchannel in DMT with $\mathrm{SNR}_n = \frac{R_{xx,n}|P_n|^2}{R_{uu,n}} = 44$ dB, permitting uncoded 2048 QAM to be transmitted at $P_e = 10^{-6}$ on this subchannel. The noise is then 44 dB below the signal on this subchannel. To estimate this noise power to within .1 dB of its true value, any residual gain estimation error on this subchannel, $R_{ee,n} - R_{uu,n}$, should be well below this mean-square noise level. That is

$$R_{ee,n} < 10^{.1/10} \cdot R_{uu,n} \approx (1 + 1/40) \cdot R_{uu,n} \tag{9.24}$$

Then,

$$SNR_{\hat{P},n} = \frac{R_{xx,n}|P_n|^2}{R_{ee,n} - R_{uu,n}} = \frac{\mathrm{SNR}_n}{(1/40)} = 40 \cdot \mathrm{SNR} \tag{9.25}$$

is then 60 dB (= 16 + 44 dB). Similarly, on a comparatively weak subchannel with SNR_i=14 dB for 4-point QAM, the $SNR_{\hat{P},n}$ should be at least 30 dB (= 14+16).

In general, for x dB accuracy

$$SNR_{\hat{P},n} \geq \left(10^{x/10} - 1\right)^{-1} \cdot \mathrm{SNR}_n \quad . \tag{9.26}$$

An Accurate Gain Estimation method for Training

A particularly accurate gain-estimation method uses a cyclically prefixed training sequence x_k with number of symbol dimensions, N, equal to or slightly longer than ν, the length of the equalized channel pulse response. Typically, N is the same as in the DMT transmission system. The receiver measures (and possibly averages over the last L of L+1 cycles) the corresponding channel output, and then divides the DFT of the channel output by the DFT of the known training sequence.

The channel estimate in the frequency domain is

$$\hat{P}_n = \frac{1}{L} \sum_{l=1}^{L} \frac{Y_{l,n}}{X_{l,n}} \quad . \tag{9.27}$$

Training Performance Analysis

The output of the n^{th} dimension on the l^{th} symbol produced by the receiver DFT is $Y_{l,n} = P_n \cdot X_{l,n} + U_{l,n}$. The estimated value for $\hat{P}_n$ is

$$\hat{P}_n = P_n + \sum_{l=1}^{L} \frac{U_{l,n}}{L \cdot X_{l,n}} \quad , \tag{9.28}$$

so

$$\Delta_n = -\sum_{l=1}^{L} \frac{U_{l,n}}{L \cdot X_{l,n}} \quad . \tag{9.29}$$

By selecting the training sequence, $X_{l,n}$, with constant magnitude, the training sequence becomes $X_n = |X|e^{\jmath\theta_{l,n}}$.

The error signal on this dimension after $\hat{P}_n$ has been computed is

$$E_n = Y_n - \hat{P}_n \cdot X_n = \Delta_n \cdot X_n + U_n \tag{9.30}$$

$$= U_n + \frac{1}{L} \cdot \sum_{l=1}^{L} U_{l,n} \cdot e^{\jmath(\theta_n - \theta_{l,n})} \quad . \tag{9.31}$$

The phase term on the noise will not contribute to its mean-square value. U_n corresponds to a block that is not used in training, so that U_n and each of $U_{l,n}$ are independent, and

$$E\left\{|E_n|^2\right\} = R_{ee,n} = R_{uu,n} + \frac{1}{L} \cdot R_{uu,n} = (1 + \frac{1}{L}) \cdot R_{uu,n} \quad . \tag{9.32}$$

The excess MSE on the i^{th} dimension then has variance $(1/\cdot L) \cdot R_{uu,n}$ that is reduced by a factor equal to the the number of symbol lengths that fit into the entire training interval length, with respect to the mean-square noise in that dimension. Thus the gain-estimation SNR is

$$SNR_{\hat{P},n} = L \cdot \frac{R_{xx,n} \cdot |P_n|^2}{R_{uu,n}} = L \cdot \text{SNR}_n \quad , \tag{9.33}$$

which means that the performance of the recommended gain estimation method improves linearly with L on all dimensions. Thus, while noise may be relatively higher (low SNR_n) on some subchannels than on others, the extra noise caused by miss-estimation is a constant factor below the relative signal power on that same dimension. In general then for x dB of accuracy

$$L \geq \left(10^{x/10} - 1\right)^{-1} \quad . \tag{9.34}$$

For an improvement in $SNR_{\hat{P},n}$ of 16 dB, Example 9.1.1 showed that SNR_n degrades by 0.1 dB. This 0.1 dB does not depend on the dimension's SNR_n; gain estimation requires $L = 40$ symbols of training on any dimension for 0.1 dB error.

For instance, the previous ADSL and VDSL examples of Section 4.6 used symbol periods of 250μs. Thus, gain estimation to within .1 dB accuracy is possible (on average) within 10 ms using this simple method.

Windowing When the FFT size is much larger than the number of significant channel taps, as is usually the case, it is possible to significantly shorten the training period. This is accomplished by transforming (DFT) the final channel estimate to the time domain, windowing to $\nu+1$ samples and then returning to the frequency domain. Essentially, all the noise on sample times $\nu + 2...N$ is thus removed without disturbing the channel estimate. The noise on all estimates, or equivalently the residual error in the SNR, improves by the factor N/ν with windowing. For the ADSL/VDSL Examples in Section 4.6, if the channel were limited to 32 nonzero successive samples in duration, then windowing would provide a factor of (32/512) reduction or 12 dB of improvement, possibly allowing just a few DMT symbols to estimate the channel to within an (on average) accuracy of .1 dB, and accurate gain estimation in a few ms.

Suggested Training Sequences

One training sequence for the gain estimation phase is a random or white sequence.[6] L blocks are used to create $\hat{P}_n$, which is computed for each dimension, one at a time. The L blocks can be periodic, or a cyclic prefix can be used to make the training sequence appear periodic. For reasons of small unidentified channel aberrations, like small nonlinearities caused by drivers, converters, amplifiers, it is better to use a cyclic prefix with non-periodic training because any such aberrations will appear as noise in the noise estimation to be subsequently described. (A purely periodic training signal might otherwise disguise these abberations.)

In complex baseband channels a "chirp" training sequence can also be used.

$$x_k = e^{j\frac{2\pi}{N}k^2} \quad . \tag{9.35}$$

[6] Say the PRBS of Chapter 10 with polynomial $(1 + D + D^{16})$ with $b_n = 2$ on channels $n = 1, ..., N-1$ and $b_n = 1$ on the DC and Nyquist dimensions. The designer should ensure that the seed for the PRBS is such that no peaks occur that would exceed the dynamic range of channel elements during gain estimation. Such peaks are usually compensated by a variety of other mechanisms when they occur in normal transmission (rarely) and would cause unnecessarily conservative loading if their effects were allowed to dominate measured SNRs.

This chirp has a theoretical minimum peak-to-average (one-dimensional) power ratio of 2 and is a "white sequence," with $r_{xx,k} = \delta_k$, where δ_k is the Kronecker delta (not the norm tap error). While "white" is good for identifying an unknown channel, a chirp's low PAR can leave certain small nonlinear "noise" distortion unidentified, which ultimately might cause the identified SNR_n's to be too optimistic.

Gain Estimation in Tracking

The value for μ in the FEQ The EMSE for the first-order zero-forcing update algorithm in Section 4.3 can be found with some algebra and approximations to be

$$\text{EMSE} \approx \frac{\mu_n \cdot \mathcal{E}_n \cdot |P_n|^2}{2 - \mu_n \cdot \mathcal{E}_n \cdot |P_n|^2} \cdot R_{\tilde{u}\tilde{u},n} \quad , \tag{9.36}$$

where $\tilde{U}_n$ is the noise scaled by the inverse of the channel. This scaling is part of both gain and noise and does not affect the achievable SNR. Thus, for the multichannel normalizer to maintain the same accuracy as the initial gain estimation in Equation (9.32)

$$\frac{1}{L} = \frac{\mu_n \cdot \mathcal{E}_n \cdot |P_n|^2}{2 - \mu_n \cdot \mathcal{E}_n \cdot |P_n|^2} \tag{9.37}$$

or

$$\mu_n = \frac{2}{(L+1) \cdot \mathcal{E}_n \cdot |P_n|^2} \quad , \tag{9.38}$$

so for $L = 40$, $|P_n|^2 = 1$, and $\mathcal{E}_n = 1$, the value would be $\mu_n = 2/41$.

Gear shifting occurs when μ_n is set at a large value and then reduced with time as the FEQ nears convergence, allowing smaller excess mean square error with each shift, but faster convergence with each shift compared to using the final μ_n value in (9.38) throughout training.

9.1.2 Noise Spectral Estimation in Training

Training also estimates the discrete-time noise power spectrum[7], or equivalently the mean-square noise at each of the demodulator DFT outputs for SNR_n computation. Noise estimation computes a spectral estimate of the error sequence, E_n, that remains after the channel response has been removed. The training sequence used for gain estimation is usually continued for noise estimation.

Analysis of Variance

The variance of L noise samples on the n^{th} dimension is:

$$\hat{\sigma_n^2} = \frac{1}{L} \cdot \sum_{l=1}^{L} |E_{l,n}|^2 \quad . \tag{9.39}$$

For stationary noise on any tone, the mean of this expression is the variance of the (zero-mean) noise. The "variance of this variance estimate" is computed as

$$\text{var}\left(\hat{\sigma_n^2}\right) = \frac{1}{L^2}\left(3 \cdot L \cdot \sigma_n^4 - L \cdot (\sigma_n^2)^2\right) = \frac{2}{L}\sigma_n^4 \tag{9.40}$$

for Gaussian noise.[8] Thus Equation (9.39)'s excess noise estimate is $\sqrt{2/L} \cdot \sigma_n^2$ and this excess noise decreases with the square-root of the number of averaged noise terms. To make this extra noise .1 dB or less, $L = 3200$. Recalling that the actual computed value for the noise estimate will have a standard deviation on average that leads to less than .1 dB error, there is still a 10% chance that the statistics of the noise will lead to deviation greater than 1 standard deviation (assuming Gaussian distribution, or approximating the more exact Chi-square with a Gaussian). Thus, sometimes a yet larger value of L is used. 98% confidence occurs at $L = 12,800$ and 99.9% confidence at $L = 28,800$. If the equivalent noise autocorrelation response is short, windowing could again be used to reduce this training time.

[7] Clearly the noise can not have been whitened if it is unknown and so this step precedes any noise whitening that might occur with later data transmission.

[8] A Gaussian has fourth moment $E(n^4) = 3\sigma^2$.

EXAMPLE 9.1.2 (ADSL) The ADSL example with symbol rate of 4000 Hz has been discussed previously. The training segment for SNR estimation in G.992.1 initially is called "Medley" and consists of 16,000 symbols (with QPSK on all subchannels) for 4 seconds of training. This extra time is provided with the cognizance that the standard deviation of the measured noise and channel should lead to no more than .1 dB error with a smaller training interval, but that there are still statistical deviations with real channels. Essentially, with this increased value, about 99% of the trainings will result in the computed value deviating by 0.1 dB or less. The gain estimation error clearly can be driven to a negligible value by picking the gain-estimation L to be a few hundred without significantly affecting the overall training time. (The long training interval also provides run time for SNR computation and loading algorithms after gains and noises are estimated.) Windowing may not be helpful in DSL noise estimation because the equivalent noise response can be long and noise can vary significantly with frequency.

Noise Variance Estimate Tracking

The value for μ' in the FEQ The standard deviation for the first-order noise update in the FEQ can be found to be

$$\text{std}(\tilde{\hat{\sigma}}_n^2) = \sqrt{\frac{\mu'}{2-\mu'}} \cdot \tilde{\sigma}_n^2 \quad . \tag{9.41}$$

Thus, to maintain a constant noise accuracy, the step size should satisfy

$$\frac{2}{L} = \frac{\mu'}{2-\mu'} \tag{9.42}$$

or

$$\mu' = \frac{4}{L+2} \tag{9.43}$$

so when $L = 6400$, then $\mu' \approx 6.25 \times 10^{-4}$. In reality, the designer might want high confidence (not just an average confidence) that the accuracy is .1 dB, in which case a μ of perhaps 10^{-4} would suggest the chance of missing would be the same as noise occuring with 4 times the standard deviation (pretty small in Gaussian and Xi-square distributions).

From Section 4.6, gain normalizers in general, – and thus the FEQ – update the noise in the proportion:

$$\frac{1}{g_n} \propto \tilde{\sigma}_n^2 \quad . \tag{9.44}$$

Effectively, the normalizer then tracks the ratio

$$\text{gain} = \frac{\tilde{\sigma}_{n,current}^2}{\tilde{\sigma}_{init,n}^2} \quad . \tag{9.45}$$

While it may take a long averaging time to introduce .1 dB accuracy, clearly an abrupt change in noise of several dB or more on any (or many) single tones can be detected rapidly.

A simple gear-shifting mechanism would be to introduce a second noise loop as in Section 4.3 with a μ'' replacing μ' where $\mu'' >> \mu'$. This faster update the noise variance estimate would be much less accurate. However, to discern whether the noise has changed rapidly, it provides an early indication. Basically, this faster tracking exploits the fact that the bit distribution is near converged in steady-state operation just before a large noise change on some tones might occur. Thus, the bit-swap controller is looking only for a gross change to initiate more rapid swapping. The following example illustrates.

Figure 9.3 illustrates the basic concept of faster swapping. It is possible to use the FEQ initially with the known training sequence and thereby avoid separate estimation of first gain and then noise, and simply directly compute the g_n values. This is feasible with gear-shifting allowing successive refinements of the g_n estimates, again as implied in Figure 9.3.

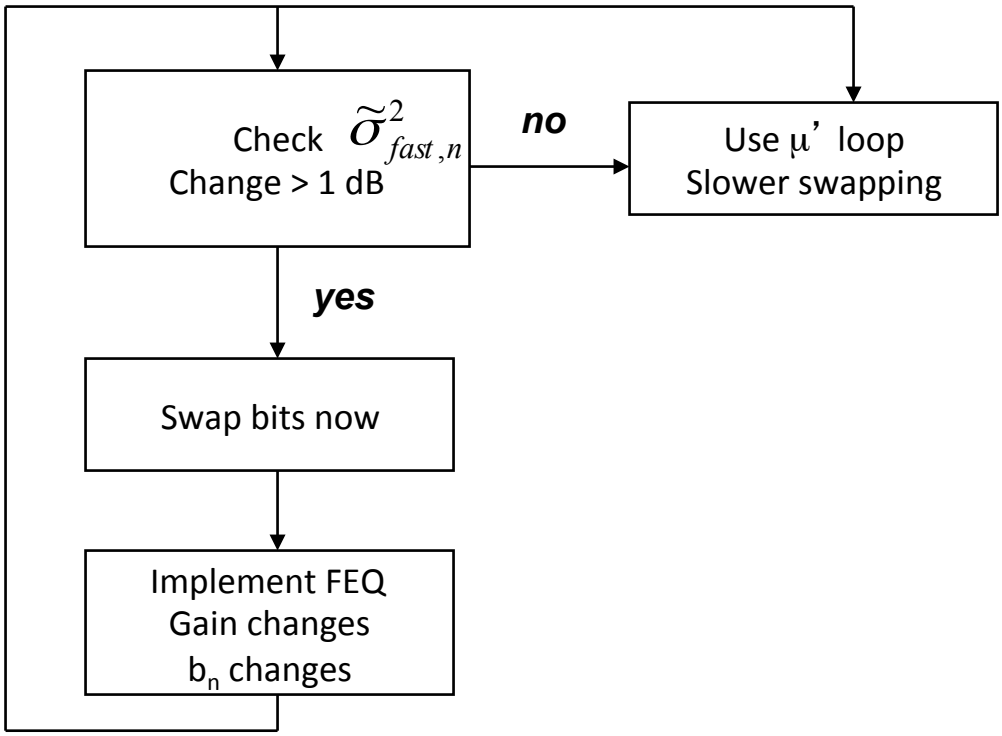

Figure 9.3: Flow chart for fast-swap augmentation of noise calculation.

EXAMPLE 9.1.3 (ADSL swap tracking speed) As an example, suppose 1 dB error (so a factor of 1+1/10) of accuracy is sufficient to detect whether a noise change of several dB has occurred. In this case

$$\frac{2}{\sqrt{L}} < \frac{1}{10} \tag{9.46}$$

which means an interval of 200 symbols would be sufficient. With ADSL's symbol rate of 4000 Hz, this corresponds to 50 ms. This would correspond to 20 bits swapped per second if a large noise event were detected, much faster than the usual situation of perhaps a swapped bit per second or less for channels with only gain change. As long as gains and bit tables are maintained (scaled as discussed in Section 4.3), a higher accuracy shift corresponding to .1 dB then replaces the faster loop to refine the noise estimate until another large noise change is observed.

9.1.3 SNR Computation in Training

The signal to noise ratio is then estimated for each dimension independently as

$$\mathrm{SNR}_n = \mathcal{E}_n \frac{|\hat{P}_n|^2}{\hat{\sigma}_n^2} \quad . \tag{9.47}$$

Loading can then be executed on the g_n to determined the $\mathcal{E}_n$ and b_n at the receiver and sent to the transmitter through a secure feedback channel during training. This step is often called "the exchange" in DMT modems and occurs as a last significant step prior to live transmission of data, often called "showtime!"

Using the FEQ, the SNR is given by (9.45). Loading can be executed in the receiver (or after passing SNR values, in the transmitter) and then the resulting b_n and $\mathcal{E}_n$ (or relatively increase/decrease in $\mathcal{E}_n$ with respect to what was used last) are sent to the transmitter via the reverse link always necessary in DMT. This was discussed at length in Section 4.3 on bit swapping.

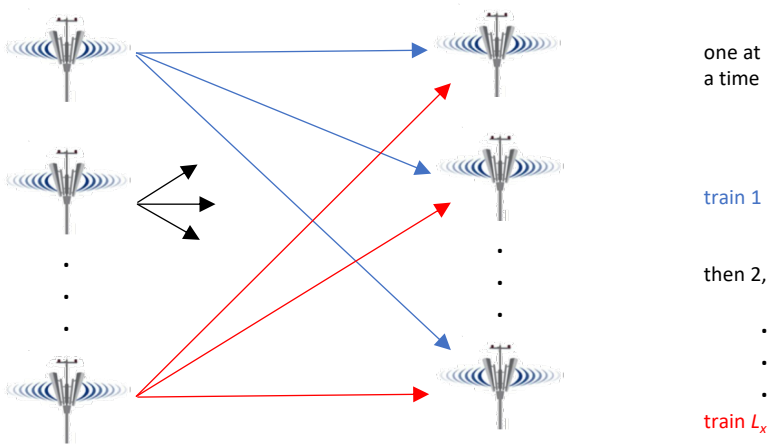

Figure 9.4: Simple training by activating each transmitter individually.

9.2 MIMO Channel Identification

The MIMO generalization of scalar channel identification begins here with a simple extension of Section 9.1's SISO case. Subsection 9.2.2 then generalizes to more efficient methods in terms of training time, which then leads to a need for singular-value decomposition of the learned MIMO result. Subsection 9.2.3 alternately describes a method that directly computes the SVD from the training sequence and the channel output using two inter-related but simple adaptive filters that avoid the intermediate calculations. This computation directly results in the per-tone matrices needed in a Vector DMT or Vector OFDM system, and will become the matrix version of the FEQ (gain normalizer) as useful also for tracking in Subsection 9.2.4. Good choices of training sequence also appear in Subsection 9.2.2.

9.2.1 Simple scalar calculation of each crosstalking gain

MIMO channel identification generalizes to the estimation of all entries in the $L_y \times L_x$ MIMO matrix $\boldsymbol{P}_n$, with these entries given by $\boldsymbol{P}_n(i,j)$, or $\hat{\boldsymbol{P}}_n(i,j)$ for $i = 1, ..., L_y$ and $j = 1, ..., L_x$. This is the transfer gain from the j^{th} input to the i^{th} output on tone n. This one-dimensional estimate becomes

$$\hat{\boldsymbol{P}}_n(i,j) = \frac{1}{L} \cdot \sum_{l=1}^{L} \frac{\boldsymbol{Y}_{n,l}(i)}{\boldsymbol{X}_{n,l}(j)} \text{ for } \boldsymbol{X}_{j' \neq j,n} = 0 \quad . \tag{9.48}$$

For this estimate, all inputs but one are zeroed in each of the L_x phases of training as in Figure 9.4, there indicated by different colors, one for each transmit antenna. The $L_y \times L_x$ matrix $\hat{\boldsymbol{P}}_n$ is then constructed from the $L_y \cdot L_x$ such computed coefficients, $i = 1, ..., L_y$, $j = 1, ..., L_x$. For each tone, vector coding in space-time would then use the SVD of $\hat{\boldsymbol{P}}_n$ so

$$\hat{F}_n \cdot \hat{\Lambda}_n \cdot \hat{M}_n^* = \hat{\boldsymbol{P}}_n \quad . \tag{9.49}$$

The total training time L_{train} should allow L_x calculations of (9.48), once for each of the L_x possible input antennas individually (in isolation) exciting the MIMO channel. The total training time L_{train} then is effectively $L_{train} = L \cdot L_x$ in terms of vector samples to estimate $\hat{\boldsymbol{P}}_n(i,j)$. (More sophisticated forms will estimate all coefficients simultaneoulsy as in Subsections 9.2.2 and 9.2.3.) The error vector for tone n at time l is formed as

$$\boldsymbol{E}_{n,l} = \boldsymbol{Y}_{n,l} - \hat{\boldsymbol{P}}_n \cdot \boldsymbol{X}_{n,l} \quad , \tag{9.50}$$

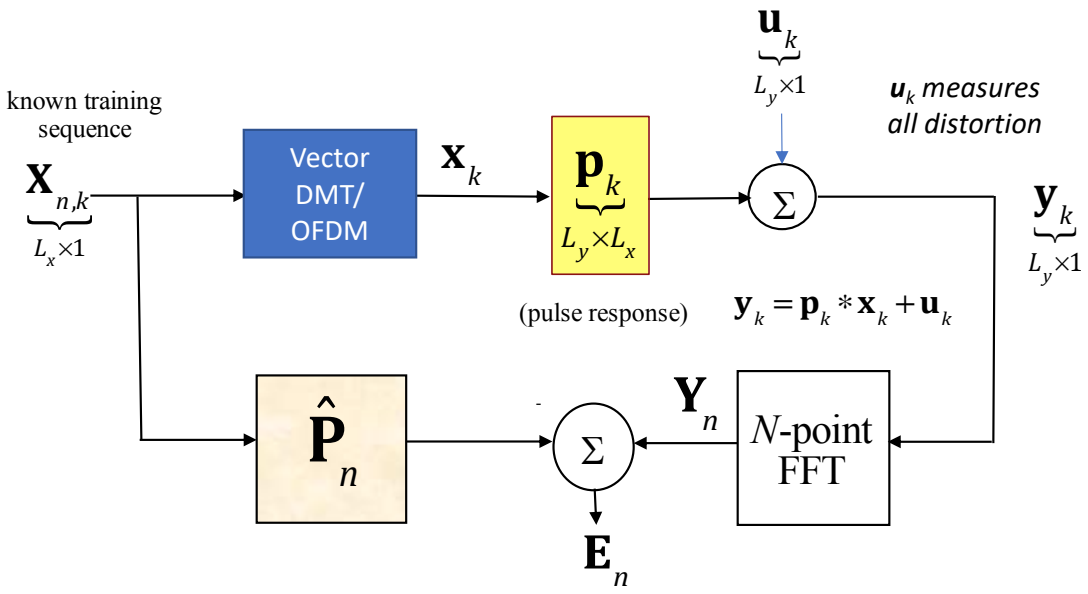

Figure 9.5: Matrix Channel/Crosstalk Identification.

and the estimated noise autocorrelation matrix is subsequently computed as

$$\hat{R}_{\boldsymbol{n},\boldsymbol{n}}(n) = \frac{1}{L_{train}} \cdot \sum_{l=1}^{L_{train}} \boldsymbol{E}_{l,n} \cdot \boldsymbol{E}_{l,n}^* \quad . \tag{9.51}$$

If this estimated noise autocorrelation matrix in (9.51) is not constant diagonal, the SVD for the channel is then computed as $\hat{R}_{\boldsymbol{n},\boldsymbol{n}}^{-1/2} \cdot \hat{\boldsymbol{P}}_n$. A more direct method for estimating the noise-equivalent MIMO channel appears later in Subsection 9.2.3.

9.2.2 Matrix simultaneous calculation of the crosstalking matrices

Figure 9.5 illustrates the random vector-process generalization of channel identification during training where each tone's $L_y \times 1$ error vector is formed as

$$\boldsymbol{E}_n = \boldsymbol{Y}_n - \hat{\boldsymbol{P}}_n \cdot \boldsymbol{X}_n \quad , \tag{9.52}$$

where the dimensions are all assumed to use the same symbol and sampling clocks. The MMSE solution for the error in (9.52) is known through Appendix D's orthogonality principal as

$$\hat{\boldsymbol{P}}_n = \underbrace{E\left[\boldsymbol{Y}_n \cdot \boldsymbol{X}_n^*\right]}_{L_y \times L_x} \cdot \left\{ \underbrace{E\left[\boldsymbol{X}_n \cdot \boldsymbol{X}_n^*\right]}_{L_x \times L_x} \right\}^{-1} \quad , \tag{9.53}$$

which minimizes actually the trace, or equivalently the determinant, of the matrix $E\left[\boldsymbol{E}_n \boldsymbol{E}_n^*\right]$, so

$$\text{MMSE} = \min_{\hat{\boldsymbol{P}}_n} \left| E\left[\boldsymbol{E}_n \cdot \boldsymbol{E}_n^*\right] \right| \ . \tag{9.54}$$

The expectation can be approximated by a sum of squared-error matrices as: The solution

$$\widehat{\text{MMSE}} = \min_{\hat{\boldsymbol{P}}_n} \left| \frac{1}{L_{train}} \cdot \sum_{k=1}^{L_{train}} \boldsymbol{E}_{n,k} \boldsymbol{E}_{n,k}^* \right| \quad . \tag{9.55}$$

(9.55)'s solution can also be found using Chapter 3's orthogonality principle, now operating on the deterministic vector space rather than statistical Hilbert Space, as

$$\hat{\boldsymbol{P}}_n = \left[\sum_{k=1}^{L_{train}} \boldsymbol{Y}_{n,k} \cdot \boldsymbol{X}_{n,k}^* \right] \cdot \left\{ \left[\sum_{k=1}^{L_{train}} \boldsymbol{X}_{n,k} \cdot \boldsymbol{X}_{k,n}^* \right] \right\}^{-1} \quad . \tag{9.56}$$

Equation (9.56) simultaneously calculates all coefficients' estimates and basically reduces the training time by L_x with respect to Subsection 9.2.1's one-scalar-coefficient-at-a-time method for the same accuracy.

The matrix inversion would nominally add significant complexity as L_x grows, but since the training sequence can be selected, certain selections will trivialize this inverted matrix to an identity. One particularly effective choice is to repeat the sequence of L_x vectors (L times), corresponding to L_x successive $L_x \times 1$ input vector samples as

$$\boldsymbol{X}_{n,k+l} = \boldsymbol{q}_{l,k} = \frac{1}{\sqrt{L_x}} \cdot \begin{bmatrix} 1 \\ e^{-\jmath 2\pi kl/L_x} \\ e^{-\jmath 2 \cdot 2\pi kl/L_x} \\ \vdots \\ e^{-\jmath l \cdot 2\pi kl/L_x} \\ \vdots \\ e^{-\jmath 2(L_x-1)\pi kl/L_x} \end{bmatrix} \quad l = 0, ..., L_x - 1 \quad . \tag{9.57}$$

(k can be set to zero above by reference time to the beginning of a block of L_x symbols that begin at sampling instant kT'.) This same vector training sequence is used on all tones. For this vector sequence, (recalling $L_{train} = L \cdot L_x$)

$$\sum_{l=1}^{L_{train}} \boldsymbol{X}_{n,l} \cdot \boldsymbol{X}_{n,l}^* = L \cdot I \quad , \tag{9.58}$$

and the matrix inversion in (9.56) is eliminated. Further this vector sequence's elements are the columns of a DFT matrix of size L_x, so its correlation with the successive elements of the channel output sequences (for each successive-time vector dimension or antenna) can be computed with a size-L_x FFT algorithm. Thus Equation (9.56)'s remaining matrix cross-correlation calculation can simplify when L_x is a large number (such as in "Massive MIMO" systems). The DFT columns $\boldsymbol{q}_{l,k}$ are not the only input training sequences that will simplify Equation (9.56)'s inverted matrix to diagonal. Other such sequences are known as complex (or real) Walsh sequences, chirp sequences, Golay Sequences, Kasami Sequences, Gold Sequences, and more generally complimentary sequences. This variety of sequences often evenly partition a constant-amplitude's circle in phase. Walsh sequences are sometimes used[9] to multiply the complex training sequence over many successive vector samples with different specific sequences used on different dimensions. This associates a unique identifier (the specific Walsh sequence) with each input antenna/dimension.

MIMO matrix channel identification is fairly direct and readily implemented. The challenge will be singular value decomposition of $\hat{\boldsymbol{P}}_n$ (without Matlab to help).

[9] Walsh sequences are binary ± 1 only. The base length two Walsh sequence are the columns/rows of

$$Q(2^1) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

and then recursively computed as the 2^k columns/rows of

$$Q(2^k) = \begin{bmatrix} Q(2^{k-1}) & Q(2^{k-1}) \\ Q(2^{k-1}) & -Q(2^{k-1}) \end{bmatrix} \quad ,$$

for which $QQ^* = 2^k \cdot I$.

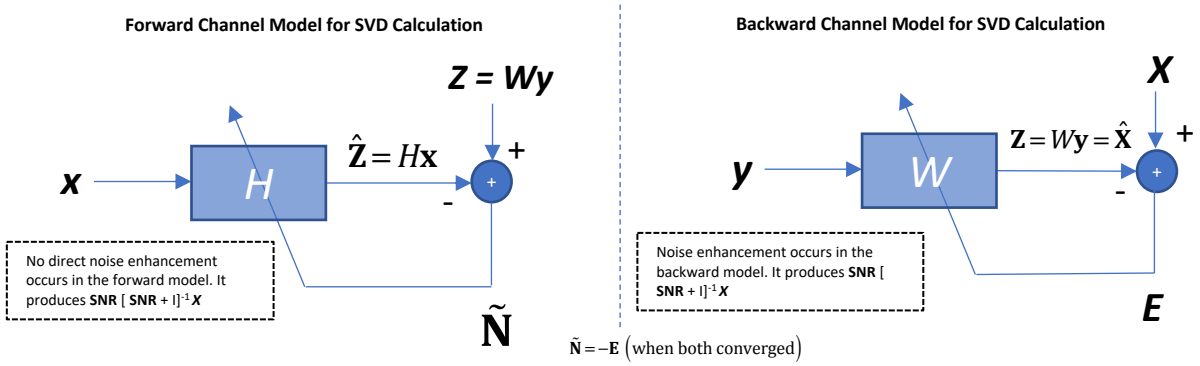

Figure 9.6: Forward and Backward Models with MMSE.

9.2.3 Adaptive Singular Value Decomposition

This text has to date leveraged the use of Matlab for analysis and design. It is unlikely such a computer tool would be available to a receiver (or transmitter) to compute SVD for each tone of a Vector-DMT or Vector-OFDM system. This subsection will first investigate some MMSE relationships that are interesting and useful for direct adaptive computation of directly SVD. This subsection simplifies notation by removing the tone, and initially time indices, as all results will apply in the same way to each tone individually. **Further, channel tonal inputs will now each individually be thought of as time-vector-sample inputs to an $L_y \times L_x$ vector-coding channel and thus LOWER case x while the corresponding pre-M and post-F^* vector-coding quantities like $\boldsymbol{X}$ and $\boldsymbol{Y}$ will be UPPER case.**

It will be convenient to define a block diagonal SNR matrix (for each tone) as

$$\boldsymbol{SNR} = \begin{bmatrix} SNR_1 & 0 & ... & 0 \\ 0 & SNR_2 & ... & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & ... & 0 & SNR_{L_x} \end{bmatrix} . \tag{9.59}$$

Chapter 5 developed and used the concepts of forward- and backward-canonical channels for the GDFE. Here, those concepts will be slightly modified to view the two estimators in Figure 9.6. These two models estimate the (scaled) SVD-matrices through two MMSE estimation problems closely related to the forward- and backward-channel directions. The matrix quantities H^* and W are viewed as adaptive matrices that are to be learned. Since W tries to form an estimate of $\boldsymbol{X}$ (the vector-coding input) from the channel output $\boldsymbol{y}$, it essentially models the SVD matrix F^*. Similarly as H estimates $\hat{\boldsymbol{X}} = W\boldsymbol{y}$ from the channel input, it essentially models the SVD matrix M^*.

The two adaptive problems when jointly solved will compute the equivalent of SVD. To see this, the backward channel problem under MMSE optimization is[10]

$$\underbrace{W}_{L_x \times L_y} = R_{\boldsymbol{Xy}} R_{\boldsymbol{yy}}^{-1} \tag{9.60}$$

$$= E\left[\boldsymbol{X}\left(\boldsymbol{Px}+\boldsymbol{n}\right)^*\right] \cdot \left[\boldsymbol{P}R_{\boldsymbol{xx}}\boldsymbol{P}^* + \frac{\mathcal{N}_0}{2}\cdot I\right]^{-1} \tag{9.61}$$

$$= M^* R_{\boldsymbol{xx}} \boldsymbol{P}^* \left[\boldsymbol{P}R_{\boldsymbol{xx}}\boldsymbol{P}^* + \frac{\mathcal{N}_0}{2}\cdot I\right]^{-1} \tag{9.62}$$

$$= M^* M R_{\boldsymbol{XX}} M \boldsymbol{P}^* F F^* \left[F\Lambda R_{\boldsymbol{XX}} \Lambda F^* + \frac{\mathcal{N}_0}{2}\cdot I\right]^{-1} F F^* \tag{9.63}$$

[10]The noise floor $\frac{\mathcal{N}_0}{2} \neq 0$ will ensure that the matrix $R_{\boldsymbol{yy}}$ is invertible.

$$= R_{\boldsymbol{XX}}\Lambda\left[\Lambda R_{\boldsymbol{XX}}\Lambda + \frac{\mathcal{N}_0}{2}\cdot I\right]^{-1} F^* \tag{9.64}$$

which equivalently says

$$F^* = \left[R_{\boldsymbol{XX}}\Lambda^2 + \frac{\mathcal{N}_0}{2}\cdot I\right]\Lambda^{-1}R_{\boldsymbol{XX}}^{-1}R_{\boldsymbol{Xy}}R_{\boldsymbol{yy}}^{-1} \tag{9.65}$$

$$= \begin{bmatrix} 1+\frac{1}{SNR_1} & 0 & ... & 0 \\ 0 & 1+\frac{1}{SNR_2} & ... & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & ... & 0 & 1+\frac{1}{SNR_{L_x}} \end{bmatrix} \underbrace{\Lambda}_{L_x\times L_x} W \tag{9.66}$$

$$= [\boldsymbol{SNR}+I]\cdot\boldsymbol{SNR}^{-1}\cdot\Lambda\cdot W \quad . \tag{9.67}$$

The diagonal matrix in (9.67) needs some interpretation as it must be $L_x \times L_x$: If $L_y \geq L_x$, then Equation (9.67) is correct as depicted by a square non-singular matrix. However, if $L_y < L_x$, then the matrix will be singular with $L_x - L_y$ zero positions that will correspond to input components that cannot be estimated at the output because they are in the null space of the channel. These components should be eliminated from the input $\boldsymbol{X}$ on this tone. Thus,

$$L_{min} \triangleq \min\{L_x, L_y\}\ , \tag{9.68}$$

and L_{min} becomes the common dimensionality of input and output after such singularity elimination. Such components depend initially on the assumption that M is known and $\boldsymbol{x} = M\boldsymbol{X}$ and that Λ is known. So if M and Λ are not yet known, the above system when adapted or learned will indeed product an $L_x \times L_y$ filter W, but some rows may have large or infinite gain. This singularity issue will be addressed shortly.

Equation 9.67 essentially says that if M is known, then Vector Coding's F^* matrix is the (diagonally scaled) solution of the backward equalization problem in Figure 9.6. That adaptive-filtering problem can be solved by a number of well-known algorithms as in Appendices A and B. The MMSE vector $\boldsymbol{E}$ has autocorrelation (uses Pythagorean Theorem for the orthogonality principle) matrix that satisfies

$$R_{\boldsymbol{ZZ}} + R_{\boldsymbol{EE}} = R_{\boldsymbol{XX}} \quad . \tag{9.69}$$

As is the case for matrix MMSE problems, $R_{\boldsymbol{EE}}$ must be diagonal, which implies that $R_{\boldsymbol{ZZ}}$ is also diagonal (and that will be seen below) since the input $R_{\boldsymbol{XX}}$ is diagonal. For any dimension in which this problem has a solution with a squared error that is very large, that dimension should be zeroed because it corresponds to a channel singularity (and a now-zeroed input value within the set of L_x input dimensions for $\boldsymbol{X}$, now reduced to $L_x - 1$).

The forward channel from $\boldsymbol{x}$ to $\boldsymbol{Z}$ (note that $\boldsymbol{Z}$ is a diagonally scaled version of Vector Coding's $\boldsymbol{Y}$ and is also associated with the MMSE linear matrix filter) as

$$\boldsymbol{Z} = W\boldsymbol{y} = \boldsymbol{X} - \boldsymbol{E} \ . \tag{9.70}$$

The channel-input vector $\boldsymbol{x}$ is repeated here as

$$\boldsymbol{x} = M\boldsymbol{X} = M\left[\boldsymbol{Z} + \boldsymbol{E}\right] = M\boldsymbol{Z} + \tilde{\boldsymbol{e}}\ , \tag{9.71}$$

where $\tilde{\boldsymbol{e}}$ is simply the rotated backward-channel MMSE error vector. Since the orthogonality principle ensures that $\boldsymbol{E}$ is uncorrelated with $\boldsymbol{y}$, then

$$E[\boldsymbol{Z}\tilde{\boldsymbol{e}}^*] = WE[\boldsymbol{y}\boldsymbol{E}^*]M^* = 0 \ . \tag{9.72}$$

Using (9.71) and (9.72) the MMSE linear combiner for the "forward channel" H is:

$$H = R_{\boldsymbol{Zx}}R_{\boldsymbol{xx}}^{-1} = R_{\boldsymbol{ZZ}}M^*MR_{\boldsymbol{XX}}^{-1}M^* \quad , \tag{9.73}$$

where

$$
\begin{aligned}
R_{\boldsymbol{ZZ}} &= E\left\{\left[(\boldsymbol{n}+WPM\boldsymbol{X})\right]\left[(\boldsymbol{n}^*+\boldsymbol{X}^*M^*P^*)W^*\right]\right\} && (9.74)\\
&= W\cdot\frac{\mathcal{N}_0}{2}I\cdot W^*+W\left(PM\cdot R_{\boldsymbol{XX}}\cdot M^*P^*\right)W^* && (9.75)\\
&= W\left[R_{\boldsymbol{nn}}+F\begin{bmatrix}\mathcal{E}_1\left|\lambda_1\right|^2 & 0 & ... & 0 & 0\\ 0 & \mathcal{E}_2\left|\lambda_2\right|^2 & 0 & ... & 0\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & ... & 0 & \mathcal{E}_{L_{min}}\left|\lambda_{L_{min}}\right|^2\end{bmatrix}F^*\right]W^* && (9.76)\\
&= WF\begin{bmatrix}\mathcal{E}_1\left|\lambda_1\right|^2+\frac{N_0}{2} & 0 & ... & 0 & 0\\ 0 & \mathcal{E}_2\left|\lambda_2\right|^2+\frac{N_0}{2} & 0 & ... & 0\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & ... & 0 & \mathcal{E}_{L_{min}}\left|\lambda_{L_{min}}\right|^2+\frac{N_0}{2}\end{bmatrix}F^*W^* && (9.77)\\
&= \begin{bmatrix}\frac{\mathcal{E}_1^2|\lambda_1|^2}{\mathcal{E}_1|\lambda_1|^2+\frac{N_0}{2}} & 0 & ... & 0 & 0\\ 0 & \frac{\mathcal{E}_2^2|\lambda_2|^2}{\mathcal{E}_2|\lambda_2|^2+\frac{N_0}{2}} & 0 & ... & 0\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & ... & 0 & \frac{\mathcal{E}_{L_{min}}^2\left|\lambda_{L_{min}}\right|^2}{\mathcal{E}_{L_{min}}\left|\lambda_{L_{min}}\right|^2+\frac{N_0}{2}}\end{bmatrix} && (9.78)\\
&= R_{\boldsymbol{XX}}\cdot\boldsymbol{SNR}\cdot\left[\boldsymbol{SNR}+I\right]^{-1} && (9.79)
\end{aligned}
$$

Thus, using this diagonal expression for $R_{\boldsymbol{ZZ}}$,

$$
\begin{aligned}
H &= \begin{bmatrix}\frac{\mathcal{E}_1^2|\lambda_1|^2}{\mathcal{E}_1|\lambda_1|^2+\frac{N_0}{2}} & 0 & ... & 0 & 0\\ 0 & \frac{\mathcal{E}_2^2|\lambda_2|^2}{\mathcal{E}_2|\lambda_2|^2+\frac{N_0}{2}} & 0 & ... & 0\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & ... & ... & \frac{\mathcal{E}_{L_{min}}^2\left|\lambda_{L_{min}}\right|^2}{\mathcal{E}_{L_{min}}\left|\lambda_{L_{min}}\right|^2+\frac{N_0}{2}}\end{bmatrix}\begin{bmatrix}1/\mathcal{E}_1 & 0 & ... & 0 & 0\\ 0 & 1/\mathcal{E}_2 & ... & 0 & 0\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & ... & 0 & 1/\mathcal{E}_{L_{min}}\end{bmatrix}M^*\\
&= \begin{bmatrix}\frac{\mathcal{E}_1|\lambda_1|^2}{\mathcal{E}_1|\lambda_1|^2+\frac{N_0}{2}} & 0 & ... & 0 & 0\\ 0 & \frac{\mathcal{E}_2|\lambda_2|^2}{\mathcal{E}_2|\lambda_2|^2+\frac{N_0}{2}} & 0 & ... & 0\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & ... & 0 & \frac{\mathcal{E}_{L_{min}}\left|\lambda_{L_{min}}\right|^2}{\mathcal{E}_{L_{min}}\left|\lambda_{L_{min}}\right|^2+\frac{N_0}{2}}\end{bmatrix}M^* && (9.80)\\
&= \begin{bmatrix}\frac{SNR_1}{SNR_1+1} & & & 0 & 0\\ 0 & \frac{SNR_2}{SNR_2+1} & 0 & ... & 0\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & . & 0 & \frac{SNR_{L_{min}}}{SNR_{L_{min}}+1}\end{bmatrix}M^* && (9.81)\\
H &= \boldsymbol{SNR}\cdot\left[\boldsymbol{SNR}+I\right]^{-1}M^* && (9.82)
\end{aligned}
$$

This H is the MMSE (so biased) version of the desired matrix M^*. The diagonal MMSE autocorrelation matrix $R_{\boldsymbol{EE}}=R_{\boldsymbol{XX}}-R_{\boldsymbol{ZZ}}$ is now readily computed using (9.79) as

$$
R_{\boldsymbol{EE}}=R_{\boldsymbol{XX}}\cdot\begin{bmatrix}\frac{1}{SNR_1+1} & 0 & ... & 0 & 0\\ 0 & \frac{1}{SNR_2+1} & 0 & ... & 0\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & ... & 0 & \frac{1}{SNR_{L_{min}}+1}\end{bmatrix}=R_{\boldsymbol{XX}}\cdot\left[\boldsymbol{SNR}+I\right]^{-1} \qquad (9.83)
$$

The MMSE forward-channel linear combiner in (9.82) is a biased-scaled version of M. Removal of the bias leaves M. This depends on knowledge of F to create $\boldsymbol{Z}$. For either the forward or backward models, the unbiased SNR's are the same as SNR_n, $n = 1, ..., N$. Since $\mathcal{E}_n$ is known on each subchannel and $\frac{\mathcal{N}_0}{2}$ is presumed equal on all subchannels, the ratios $g_n = \frac{\lambda_n}{\frac{\mathcal{N}_0}{2}} = \frac{SNR_n}{\mathcal{E}_n}$ are thus also computed, in each case assuming that the "other matrix" (M in the backward case and F^* in the forward case, or their scaled equivalents) are known. Any complex (meaning non-zero phase) λ values should be replaced by $|\lambda_n|$ and their phase absorbed into the corresponding column/row of either F or M (which will not change the MMSE solution, and ensures these estimated singular values are real and positive). M and F remain unitary with such phase change.

Iterative SVD Calculation

The forward and backward adaptive filters depend on each other through the presumption of either F^* or M respectively, and each "adapt to the other's' respective choice' However, neither M nor F is known initially. If the adaptive loops (solutions to the MMSE problem) are alternately executed with the results of the other loop's most recent solution, then the MSE cannot increase because it corresponds to the product of the same set of SNR_n's in each case. A zeroed W-filter SNR on any excited (so $\mathcal{E} \neq 0$) input dimension will correspond to a channel singularity and that dimension is useless, corresponds to a zero singular value in the channel, and the corresponding input dimension should be dropped. These zeroed SNR's will correspond to large error-vector values on the corresponding dimensions of $\boldsymbol{E}$. Large such values that exceed $\mathcal{E}_n/\text{SNR}_{\text{MFB}}(n)$ for that dimension should be eliminated. The input columns of M corresponding to singularity (zero singular values) are not needed in the system. The remaining non-singular dimensions are retained for these H and W matrix "filters," and these will have a non-zero product SNR. When the system has converged, there will be a set of parallel independent channels with W and H that have the same SNR_n's as vector coding (and indeed thus are equivalent other than irrelevant equal-signal-and-noise diagonal scaling to vector coding) so there is no need to actually compute F^* unless there is some need for this exact unitary matrix.

Since the SVD setting is the global MMSE optimum, and each of these loops is convex by itself, the MSE will reduce at each step. Such a system must eventually converge to the global optimum, and the final values of F and M (along with the corresponding non-zero singular values) can be simply derived from H, W, and SNR_n if needed.

Figure 9.7 illustrates the basic program flow where the ϵ is simply a stopping-criterion threshold for noise/error reduction is small enough that further progress is not necessary.

This particular process in Figure 9.7 never needed the explicit identification of the matrix $\hat{\boldsymbol{P}}$, although the action of the channel $\boldsymbol{P}$ is implied as known because the channel inputs and outputs are observed and known (so the channel itself is the $\boldsymbol{P}$ tacit in these observations). What is needed are the two known quantites, the vector-coding inputs $\boldsymbol{X}$ and the channel output $\boldsymbol{y}$ on each tone. Both the receiver and the transmitter can execute the algorithm in parallel (presuming that $\boldsymbol{y}$ is returned to the transmitter through a feedback channel). Equivalently, the receiver could run the process and just return the value of H (or equivalently M^*) to the transmitter, but as will be discussed only the error sequence need be returned.

This process is further simplified when the special training sequence is used for the forward channel. In this case the least-squares estimate of H has the diagonal $L_{train} \cdot I$ for the inverted matrix in (9.56) in the adaptive filter so the estimate $\hat{H}$ is then given by

$$\hat{H} = \frac{1}{L_{train}} \cdot \left[\sum_{k=1}^{L_{train}} \boldsymbol{Z}_{n,k} \cdot \boldsymbol{X}^*_{n,k} \right] \quad . \tag{9.84}$$

However, the W update does not so easily simplify and will require a more complete adaptive algorithm like QR factorization as in Appendix C.

It is relatively simple to compute the $L_{min} \times L_{min}$ relevant portion of $\hat{\boldsymbol{P}}$ according to

$$\hat{\hat{\boldsymbol{P}}} = W^{-1} H \quad . \tag{9.85}$$

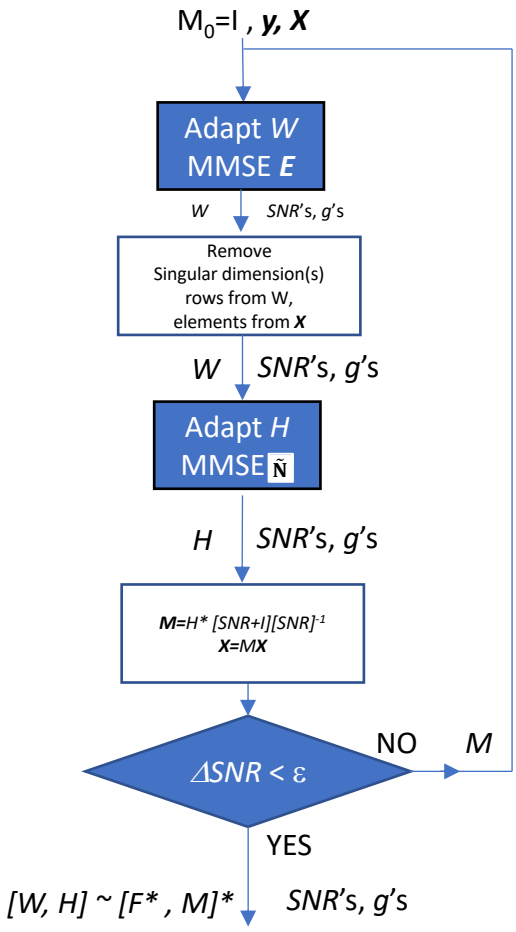

Figure 9.7: Iterative calculation of SVD matrices.

(To avoid unnecessary matrix inversion, W^{-1} simplifies to $W^*\Lambda^2[\boldsymbol{SNR}+1]^2\boldsymbol{SNR}^{-2}$, so transpose and diagonal multiplication.)

Information return to the transmitter and interpolation

The MIMO system could return the matrix H (or equivalently M^*) to the transmitter, in addition to the bits and gains quantities exchanged in SISO systems for all dimensions (or groups of dimensions). The amount of feedback depends on the type of transmission system using the Vector DMT or Vector OFDM. The bit/gains distributions are as in Chapter 4:

$$\boldsymbol{B}_n = [b_{n,1}\ b_{n,2}\ ...\ b_{n,L_{min}}] \tag{9.86}$$

$$\boldsymbol{G}_n = [G_{n,1}\ G_{n,2}\ ...\ G_{n,L_{min}}] \tag{9.87}$$

When the same number of bits and equal energy are transmitted on each spatial dimension of a group, then

$$\boldsymbol{B}_n \rightarrow [B_n\ B_n\ ...\ B_n] \tag{9.88}$$

$$\boldsymbol{G}_n \rightarrow [G_n\ G_n\ ...\ G_n] \tag{9.89}$$

Often wireless systems many or all tones may be grouped, in which case

$$B_n \rightarrow \bar{b} \tag{9.90}$$

$$G_n \rightarrow \frac{\mathcal{E}_{new}}{\mathcal{E}_0} \tag{9.91}$$

or equivalently the number of bits per tone and the transmit power (or flat power-spectral density corresponding to that power over the used tones) are the same on all dimensions.

Fixed-line systems (like DSL, but also powerline or some very high-speed ethernet) will have a substantial training period upon a "reboot" (which means all lines are coordinated and start at same time). This will include sufficient known training sequences across all the crosstalking wires for which the full SVD can be learned on each tone, along with a number of bits and appropriate energy-gain adjustment. Such systems can take a significant amount of time to retrain/reboot (from seconds to even minutes), but once operational can instead track. Shortly, this subsection finds the W in Figure 9.6 to be the MIMO generalization of the earlier gain normalizer or "FEQ" for use in tracking, which is particularly of interest if one initially silent wire from of a MIMO set of wires energizes, the so-called "joining event." There is corresponding a "leaving" event when a wire from the set goes silent. These too are handled by tracking the MIMO FEQ. The wireline systems will usually (almost always) learn the MIMO $\hat{\boldsymbol{P}}_n$ for all n and track them in coordinated fashion on all tones.

For wireless MIMO systems, the grouping of tones substantially reduces the bits/gain feedback to basically a (or a few) constellation size(s), and code rate(s) as in Section **??**. However, for each tone, H is $L_{min} \times L_x$ and there are thus $L_{min} \cdot L_x$ scalar (possibly complex) scalar coefficients to return. Overall, this is $\bar{N} \cdot L_{min} \cdot L_x$ coefficients. An example calculation finds that for 8 transmit antennas and 4 receive antennas and 4096 tones that the system may need to send back as many roughly 128k such coefficients. For systems with very large bandwidth (100's of Mbps to Gbps or more), this is basically small - for systems with bandwidths of a few Mbps, it would probably take too long (and use substantial bandwidth).

Wi-Fi systems tend to use smaller $\bar{N}$ (up to 512 << 4096) and have high bandwidths so they do return the "SVD result" (H or M^*). However, Wi-Fi systems use Vector OFDM and a single QAM constellation for all tones (so one group). Thus the SNR estimation need not be as accurate as in fixed line, and no decisions about carriers to be used – or not used –(like LC or Water-fill loading) occurs.

Wi-fi systems use a **Neighborhood Discovery Protocol (NDP** sequence within the training preamble at the beginning of a series of information packets in Wi-Fi's random access protocol, sometimes also know as the LTF (long-training frame). These are typically just a few symbols long (4-32 such symbols). By recognizing most of the carriers will have the same spatial structure (modulo phase differences that are directly related to symbol rate and carrier frequency, both known), the averaging

to improve the SVD can be "across the tones." Those tones that appear to have fade issues should be deleted from the averaging as in Section **??**. The applied codes must then have sufficient redundancy at the selected data rate (constellation size and code rate) to recover the faded dimensions.

The majority of Wi-Fi training feedback is thus for the SVD-based partitioning. The receiver will track through (see next subsection) matrix FEQ, but no swapping occurs. Wi-Fi uses a collision-detect protocol and thus will retrain for each packet, usually with a fresh start on the SVD calculation, although smart systems may use proprietary methods that store and leverage the earlier SVD M matrices used from legacy operation. Basically, the Wi-Fi system does a gross SVD to help increase the data rate by roughly L_{min}, hopefully enough to partition the channels physically for Vector-OFDM and leaving the rest to an outer code correct. More difficult channels (basically $|R_{\boldsymbol{ee}}|$ is larger) will see lower code rates and smaller constellations used.

LTE systems can have large N's and smaller bandwidths, so they often do not return fully the equivalent information to the transmitter. Instead they use embedded pilots (reference elements) to initialize. These LTE systems also use some periodically inserted primary and secondary synchronization symbols that occupy about 1/2 the tones when transmitted. Interpolation is then used between different tones' measured $\hat{\boldsymbol{P}}_n$ matrices to estimate the rest. The pilots change tone indices n every few symbols in a standardized pattern. For those M_n matrices returned to the LTE transmitter, other matrices $M_{i\neq n}$ can be interpolated on a coefficient by coefficient basis by using known tones' values. Linear interpolation (basically interpolate along between $\hat{\boldsymbol{P}}_n$ and $\hat{\boldsymbol{P}}_{n+\Delta n}$ along the line between them according to tone index) can be used, as can two-point sinc-function interpolation. More sophisticated curve fitting like that of LaGrange interpolation can be used on more than two points/pilots/references to interpolate to frequencies for which no M_n matrix is provided/returned. The MIMO FEQ also can use decisions as correct on all tones and return the error values instead as in Subsection 9.2.4.

Separation of Channel and Noise

The adaptive SVD method does not readily separate the channel and noise when the noise is not white. The separation is not needed for Vector-DMT/OFDM systems to function. If for some diagnostic purpose, this separation is needed, the channel without noise can be computed according to the original formula from (9.56) (using the simplification of the diagonalized inverted matrix and recalling that notation returned to lower case vectors for each input/output tone of the Vector DMT/OFDM channel to distinguish them from the capital letters used for the Vector Coding inputs and outputs on each such tone) as:

$$\hat{\boldsymbol{P}} = \frac{1}{L_{train}} \cdot \left[\sum_{k=1}^{L_{train}} \boldsymbol{y}_k \boldsymbol{x}_k^* \right] \quad . \tag{9.92}$$

The noise is then estimated from the estimated noise vector

$$\hat{\boldsymbol{n}}_k = \boldsymbol{y}_k - \hat{\boldsymbol{P}} \cdot \boldsymbol{x}_k \tag{9.93}$$

as

$$\hat{R}_{\boldsymbol{nn}} = \frac{1}{L_{train}} \cdot \sum_{k=1}^{L_{train}} \hat{\boldsymbol{n}}_k \hat{\boldsymbol{n}}_k^* \quad . \tag{9.94}$$

In the MIMO case though, these noise-free channel and channel-free noise estimates may be of less interest than W and H.

Use of the method to compute general SVD

Singular Value Decomposition is well studied and implemented by mathematicians and computer scientists, whence the nice Matlab implementations engineers often take for granted. However, some comments are in order here: The SVD computed in this text is that for the transmission problem and exploits the non-singularity of $R_{\boldsymbol{yy}}$ (for nonzero noise). In effect, by setting an (somewhat arbitrary from the mathematician's standpoint) $\mathcal{E}_n$ and a $\frac{\mathcal{N}_0}{2}$, the algorithm here is "conditioning the problem" and tacitly making it easier. SVD can only be computed iteratively in general, and indeed the method here

is iterative. That is, there is no general closed-form way to compute SVD. The forward- and backward-filters in the algorithm here are basically instances of the two QR and RQ. These adaptive filters could be implemented by L Householder (really iterative sequences of Householder) Transformations in parallel (one for each row/column of the W or H matrices respectively). In general the SVD calculator will not have the benefit of knowing the SNR to which they must perform so MIMO transmission is a special case for the transmission engineer. Essentially, singular values that are small enough don't matter to the transmission engineer, so the algorithms here simply ignore them. General SVD methods cannot make this assumption.

The method in Figure 9.7 could be used to compute SVD of a matrix P subject to the singularity level limited to the selected set of $\mathcal{E}_n$ (which may well be constant) and $\frac{\mathcal{N}_0}{2}$. The adaptive filter for W then becomes the direct calculation on loop pass k with M_k as most recent transmit-side matrix estimate

$$W_{k+1} = R_{\boldsymbol{XX}} \cdot M_k^* P^* \left[P M_k \cdot R_{\boldsymbol{XX}} \cdot M_k^* P^* + \frac{\mathcal{N}_0}{2} I \right]^{-1} \quad . \tag{9.95}$$

With non-zero noise, the inversion always works. The corresponding MMSE is

$$R_{\boldsymbol{EE}} = [I - W \cdot P M_k] \cdot R_{\boldsymbol{XX}} \tag{9.96}$$

For diagonal elements of $R_{\boldsymbol{EE}} > \mathcal{E}/[\text{SNR}_{\text{MFB}}(n)]$, the corresponding element in $\boldsymbol{X}$ and corresponding column of M_{k+1} should be eliminated. The update of M is then

$$M_{k+1} = [\boldsymbol{SNR} + I] \cdot \boldsymbol{SNR}^{-1} \cdot W_{k+1} \cdot P \quad . \tag{9.97}$$

The inversion in (9.95) could be implemented by QR factorization for each of the output rows/columns. The process terminates when the change in remaining SNRs ceases to be appreciable.

9.2.4 Tracking

Tracking in MIMO systems essentially extends to more dimensions the basic methods already used, as will be detailed in this subsection, with address of the possible feedback of the M matrix.

Tracking and the MIMO FEQ

The adaptive SVD of Subsection 9.2.3 can be continued for tracking. This subsection describes such tracking by interpreting the (scaled) matrix W as a **MIMO-FEQ**. The converged W matrix will implement (after removal of diagonal gains) the F^* of SVD on each tone, namely the MIMO FEQ will be

$$W_U \triangleq [\boldsymbol{SNR} + I] \cdot \boldsymbol{SNR}^{-1} \cdot W = \Lambda^{-1} \cdot F^* \quad . \tag{9.98}$$

The biased SNR on each tone computes directly from $\mathcal{E}_n / R_{\boldsymbol{EE}}(n)$, and the set of sub-channels created will be vector coding. A gradient algorithm[11] to track W is

$$W_{k+1} = W_k + \mu_W \cdot \boldsymbol{E}_k \cdot \boldsymbol{y}_k^* \quad . \tag{9.99}$$

The MSE can be tracked for each tone by the usual FEQ noise tracking from Section 9.1 with its own μ_W', and the input energies $\{\mathcal{E}_n\}$ are known. When the MIMO transmitter is not M, this W matrix will find the MMSE solution for whatever value of M the transmitter chooses to use. The set of SNR's will be worse (indeed each can be no better either) when the transmitter is not using the correct M. Nonetheless tracking can continue in the receiver and any swapping of bits/gains to accommodate performance degradation caused by a sub-optimum transmitter matrix will consequently occur.

[11] The designer may be tempted here to substitute $\boldsymbol{X}_k$ from the decision-device output for $\boldsymbol{y}$ to create a MIMO zero-forcing algorithm, but that only works when $L_{min} = L_y$. Thus, better to use the MMSE-based gradiant algorithm and fix the bias as a final step.

Tracking the Prefilter M

When channel (and crosstalk) vary sufficiently slowly, the transmit matrix can also be adapted through the use of some additional feedback (beyond bits and gains). Instead of periodically returning the H matrix to the transmitter (which would require $N \cdot L_x \cdot L_{min}$ coefficients for each change, the error signal $\boldsymbol{E}_{U,k}$ can be periodically returned to the transmitter[12]. The unbiased transmit matrix, now here called a **Frequency Pre-Coder (FPC)**, from the adaptive SVD computation was

$$H_U \triangleq [\boldsymbol{SNR} + I] \cdot \boldsymbol{SNR}^{-1} \cdot H = M^* \quad . \tag{9.100}$$

In the case of the transmit gradient algorithm to update H, H_u as

$$H_{U,k+1} = H_{U,k} + \mu_H \cdot \tilde{\boldsymbol{N}}_{U,k} \cdot \boldsymbol{x}_k^* \quad . \tag{9.101}$$

The transmit-side error signal is

$$\tilde{\boldsymbol{N}}_{U,k} = -\boldsymbol{E}_{U,k} \quad . \tag{9.102}$$

The individual SNR's can be computed by the transmitter, although bits and gains are set by swapping in any case. The transmitter can calculate the noise on each tone according to the same type of noise loop as in Subsection 9.1 as

$$\sigma^2_{n,k+1} = (1 - \mu'_H)\sigma^2_{n,k} + \mu'_H |\boldsymbol{E}_{U,k}|^2 \quad . \tag{9.103}$$

For SISO systems, this FPC operation was superfluous to the gain settings G_n. In MIMO, it is non-trivial and thus needs implementation. For the SISO system, $H = 1$ always. If the transmitter and receiver used exactly the same loading algorithm, indeed the feedback of bits and gains would no longer be necessary as the error-vector sequence carries the same information. However, that requires a common understanding (and agreement) on the exact loading algorithm, which may be too much to ask of engineers to standardize as they'll all have their own version.

DSL systems such as G.vector and G.fast use this type of error-feedback mechanism, but also feedback bits and gains. No precise loading algorithm is standardized, just swapping that now includes the error sequence. In DSL, the error sequence is only returned for the known synchronization symbols embedded and for training.

Synchronization symbols

to be added.

Codebooks

to be added.

[12] Recalling that $\boldsymbol{E}_U = [\boldsymbol{SNR} + I]\, \boldsymbol{SNR}^{-1} \boldsymbol{E}$.

9.3 Learning Bounds For Linear Channels, Large Dimensionality, and Nonlinear channels

To be added.

9.4 Adaptive Vector Learning algorithms for linear channels

In practice, digital receivers that use equalizers, or linear combiners for MIMO, are usually adaptive. An adaptive equalizer automatically adjusts its internal settings (the values for $\boldsymbol{w}$ so as to realize the MMSE settings as closely as possible. Adaptive equalizers and/or linear combiners perform this self optimization by measuring their own performance and then recursively in time incrementing the vector settings so as to improve performance on the average. It is presumed that the designer does not know the channel response, so that a good linear filter cannot be predesigned, and that the channel response may be subject to variation with time or from use to use.

There are several adaptive algorithms in use in sophisticated equalization and linear-combiner systems today. By far, the most commonly used is (a variant of) the Widrow-Hoff (1960) LMS algorithm, which is an example of the engineering use of what is known in statistics as a "stochastic-gradient" algorithm, see Robins-Munro (1951). For numerical reasons, this algorithm is altered slightly in practice. Other algorithms in use are Lucky's Zero-Forcing algorithm (1966), Recursive Least-Squares methods (Godard), and frequency-domain algorithms.

9.4.1 The LMS algorithm for Adaptive Equalization

The Least Mean Square (LMS) algorithm is used with both $\boldsymbol{w}$ and $\boldsymbol{b}$ (or $\boldsymbol{G}$) in MMSE DFE's and GDFE's as slowly time-varying vector filters. A matched filter is usually not used (only pre-coversion anti-alias filters), and so the adaptive filter will need to have sampling rates sufficiently high (so twice the highest baseband-equivalent frequency) to capture all information. Matched filters are then just absorbed into the adaptation of the filters. For instance, fractionally spaced equalizers were discussed in Chapter **??** for this specific reason. The LMS algorithm attempts to minimize the MSE by incrementing the equalizer parameters in the direction of the negative gradient of the MSE with respect to $\boldsymbol{w}$ (and also $\boldsymbol{b}$ or $\boldsymbol{G}$ for the DFE). This direction is the direction of steepest descent (or most reduction in MSE). Recall the scalar MSE is $E\left[|e_k|^2\right]$, so the idea is to update the filter parameters $\boldsymbol{w}$ by

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \frac{\mu}{2}\nabla_{\boldsymbol{w}_k} E\left[|e_k|^2\right] \quad , \tag{9.104}$$

where $\nabla_{\boldsymbol{w}_k} E\left[|e_k|^2\right]$ is the gradient of the MSE with respect to the filter coefficient vector $\boldsymbol{w}_k$. In practice, the channel statistics are not (exactly) known so that the gradient term,$\nabla_{\boldsymbol{w}_k} E\left[|e_k|^2\right]$, in (9.104) is unknown. It is, instead, estimated by the **stochastic gradient** $\nabla_{\boldsymbol{w}_k}|e_k|^2$, which is easily computed as

$$\nabla_{\boldsymbol{w}_k}|e_k|^2 = -2e_k^*\boldsymbol{y}_k \tag{9.105}$$

where $\boldsymbol{y}_k$ is (again) the vector of inputs to the FIR filter(s) in the adaptive equalizer or linear combiner. The LMS algorithm then becomes

$$e_k = x_k - \boldsymbol{w}_k^*\boldsymbol{y}_k \tag{9.106}$$

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \mu e_k^*\boldsymbol{y}_k \quad . \tag{9.107}$$

There are a variety of proofs, under various mild assumptions and approximations, that the LMS algorithm converges to the MMSE settings if μ is sufficiently small and numerical precision is adequate. Thus the exact channel pulse response and noise statistics need not be known in advance. Usually, the input sequence x_k is known to the receiver for a short training time in advance of unknown data transmission, so that (9.106) can be computed in the receiver, and updating allowed. After training for a time period sufficient for the adaptive filter to converge, the known data sequence must be replaced by the decisions in the receiver for adaptation to continue. This type of adaptive equalization is called **decision directed**.

For stable operation of the LMS adaptive algorithm,

$$0 \le \mu \le \frac{1}{N_f\mathcal{E}_{\boldsymbol{y}} + N_b\mathcal{E}_{\boldsymbol{x}}} \quad . \tag{9.108}$$

Typically, the upper limit is avoided and the step-size μ is usually several orders of magnitude below the upper stability limit, to ensure good noise averaging and performance close to the MMSE level.

In practice, the LMS update must be implemented in finite-precision, and a variant of it, known as the Tap-Leakage Algorithm, or as Leaky LMS, must be used:

$$\boldsymbol{w}_{k+1} = \beta \boldsymbol{w}_k + \mu e_k^* \boldsymbol{y}_k \tag{9.109}$$

where β is close to, but slightly less than unity. This is equivalent to minimizing (via stochastic gradient) the sum of the MSE and $(1-\beta)\|\boldsymbol{w}_k\|^2$, thus preventing an arbitrary growth of $\boldsymbol{w}$ caused by accumulated round-off error in practical implementation.

Zero-Forcing Algorithm

A lesser used, but simpler, adaptive algorithm is the zero-forcing algorithm, which produces the zero forcing settings for the adaptive receiver. This algorithm is derived by noting that the equalizer error, and the input data vector should be uncorrelated when the receiver has the zero-forcing setting. This algorithm is written

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \mu e_k^* \boldsymbol{x}_k \tag{9.110}$$

where $\boldsymbol{x}_k$ is a vector of training (or decision-directed-estimated) data symbol values. Since the symbol values are often integer valued, the multiply in (9.110) is easily implemented with addition and possible minor shifting, thus making the zero-forcing algorithm somewhat simpler than the LMS.[13] In practice, this simplification can be significant at very high-speeds, where multiplication can be difficult to implement cost effectively.

Signed LMS Algorithm

Perhaps, more common than the zero-forcing algorithm is the **signed LMS algorithm**

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \mu \cdot e_k^* \cdot \text{sgn}\left[\boldsymbol{y}_k\right] \tag{9.111}$$

where the sgn function eliminates the need for multiplication.

This algorithm also converges to the MMSE solution in practice, although usually more slowly than the LMS.

9.4.2 Vector-MIMO Generalization of LMS

When the error is a vector, LMS generalizes in the obvious way to (with $\tilde{\boldsymbol{W}} = [\boldsymbol{W} \;\; \boldsymbol{B}]$

$$\boldsymbol{e}_k = \boldsymbol{x}_k - \tilde{\boldsymbol{W}}_k^* \boldsymbol{Y}_k \tag{9.112}$$

$$\tilde{\boldsymbol{W}}_{k+1} = \tilde{\boldsymbol{W}}_k + \mu \boldsymbol{e}_k^* \boldsymbol{Y}_k \quad . \tag{9.113}$$

Different values of μ may be used for different dimensions (as well as for feedback and feedforward components within $\tilde{\boldsymbol{W}}$. The ZF and Signed LMS both also generalize in exactly the same way for the MIMO case.

[13] The step-size μ is usually implemented as a power of 2, and thus also corresponds only to minor shifting.

Exercises - Chapter 7

9.1 *Channel ID*

A DMT system uses the channel ID method of Section 9.1. Slobhouse Designs Inc. has decided that 1dB accuracy in SNR measurement is sufficient.

a. (1 pt) Slobhouse decided to measure gain so as to induce less than 1dB error in SNR. How many training symbols (L) does SDI need?

b. (1 pt) If Slobhouse uses the same number (L) of symbols as in part a to do noise estimation, comment on SDI's noise estimation.

c. (3 pts) If L_1 is the number of measurements to estimate the gain, and L_2 is the number of measurements to estimate the noise, approximately estimate the minimum $L_1 + L_2$ such that the overall SNR error is less than 1 dB. There may be several solutions that are close in answer for this part.

9.2 *Channel ID - Midterm 2000*

Channel identification of Section 9.1 estimates channel SNR's for a system with symbol rate 4 Hz on a linear bandlimited AWGN channel. A periodic training signal is used. This problem estimates training time.

a. How many symbols of training are necessary for an average gain-estimation error of .25 dB (so that the variance of the gain estimation error is less than .25 dB away from the correct answer)? How long will this be? (2 pts)

b. Suppose the FFT size is $N = 8192$, but that the channel length is less than $\nu = 640$ sample periods. If the gain-estimation distortion is distributed evenly in the time domain over all 8192 time samples, but the result was windowed to 640 time-domain samples, what would be the new training time in symbol periods and in absolute time? (2 pts)

c. Suppose .25 dB noise estimation error is tolerable on the average (so that the standard deviation of the noise variance estimate is less than .25 dB away from the true noise). What is the step size of the steady-state FEQ-error based noise estimator, μ', with unit energy transmitted on the corresponding subchannel? (1 pt)

Appendix A

Gradient Algorithms

Stochastic gradient, or in particular the well-known adaptive filtering algorithm Least-Mean Square (LMS), pervade engineering uses in digital transmission.

A.1 Least Mean Square (LMS) - stochastic gradient method

A.1.1 basic LMS

Convergence and Tracking

leakage methods

A.2 Zero-Forcing and Correlation Methods

A.3 Steepest Descent Methods, Newton's Method

A.4 Frequency-Domain and Block Algorithms

Appendix B

Projection Methods methods

B.1 Basic Recursive Least Squares

B.2 QR methods with RLS

ignore this text inserted here for now.

A numerically sound way to solve this problem (even when singular) without directly inverting a matrix is recursively executed for each of the L_x dimensions of $\boldsymbol{X}$ (or correspondingly of $\boldsymbol{E}$) as the matrix problem for the $L_x \times L_x$ criterion in (**??**) is variables separable in the rows of W, or equivalently the columns of W^*. A sequence of L_{train} training samples in any of the L_x dimensions produces this vector summary of the error sequence for that same dimension, call it $j \in \{1, ...L_x\}$.

$$\begin{bmatrix} E_l(j) \\ E_{l-1}(j) \\ \vdots \\ E_0(j) \end{bmatrix} = \begin{bmatrix} X_l(j) \\ X_{l-1}(j) \\ \vdots \\ X_0(j) \end{bmatrix} - \begin{bmatrix} Y_l(L_y) & ... & Y_l(1) \\ Y_{l-1}(L_y) & ... & Y_{l-1}(1) \\ \vdots & \vdots & \vdots \\ Y_0(L_y) & ... & Y_0(1) \end{bmatrix} \begin{bmatrix} W_j(L_y) \\ W_j(L_y - 1) \\ \vdots \\ W_j(1) \end{bmatrix} . \tag{B.1}$$

The norm of the vector in (B.1), or some of its squared element values, is the least-squares error squared magnitude. A method for find the solution is through QR factorization that pre-multiplies both sides of (B.1) by a unitary matrix Q that triangularizes the large data matrix with the Y entries into an $L_y \times L_y$ matrix in the lowest L_y rows:

$$Q_{l,j} \cdot \begin{bmatrix} E_l(j) \\ E_{l-1}(j) \\ \vdots \\ E_0(j) \end{bmatrix} = \begin{bmatrix} e_l(j) \\ e_{l-1}(j) \\ \vdots \\ e_0(j) \end{bmatrix} = \begin{bmatrix} \mathcal{X}_l(j) \\ \mathcal{X}_{l-1}(j) \\ \vdots \\ \mathcal{X}_0(j) \end{bmatrix} - \begin{bmatrix} 0 & 0 & ... & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & ... & 0 \\ r_{L_y,l}(L_y) & ... & r_{L_y,l}(2) & r_{L_y,l}(1) \\ r_{L_y-1,l}(L_y) & ... & r_{L_y-1,l}(2) & 0 \\ \vdots & \iddots & \vdots & 0 \\ r_{1,l}(L_y) & 0 & ... & 0 \end{bmatrix} \begin{bmatrix} W_j(L_y) \\ W_j(L_y - 1) \\ \vdots \\ W_j(1) \end{bmatrix} . \tag{B.2}$$

Because of the triangular structure, the lower L_y entries can be zeroed and the remaining nonzero entries above those sum in squared magnitude to the minimum least-squares sum of errors for dimension j. This can be repeated for all dimensions. The sum of minimized squared errors also will be then equal to the upper corresponding sum of squares in the rotated vector

$$Q_{l,j} \begin{bmatrix} X_l(j) \\ X_{l-1}(j) \\ \vdots \\ X_0(j) \end{bmatrix} = \begin{bmatrix} \mathcal{X}_l(j) \\ \mathcal{X}_{l-1}(j) \\ \vdots \\ \mathcal{X}_0(j) \end{bmatrix} . \tag{B.3}$$

The $Q_{l,j}$ unitary matrix can be computed as a sequence of $l \cdot L_y$ elementary 2×2 rotations, through a recursion of L_y such rotations for each new upper row placed at time l into the large matrix of Y values:

$$\tilde{Q}_k = \begin{bmatrix} \cos(\theta_{2,k}) & 0 & ... & 0 & -\sin(\theta_{2,k}) & 0 \\ 0 & 1 & ... & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & ... & 1 & 0 & 0 \\ \sin(\theta_{2,k}) & 0 & ... & 0 & \cos(\theta_{2,k}) & 0 \\ 0 & 0 & ... & 0 & 0 & 1 \end{bmatrix} \cdots \begin{bmatrix} \cos(\theta_{L_x,k}) & 0 & ... & 0 & -\sin(\theta_{L_x,k}) & 0 & ... & 0 \\ 0 & 1 & ... & 0 & 0 & 0 & ... & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & ... & 1 & 0 & 0 & ... & 0 \\ \sin(\theta_{L_x,k}) & 0 & ... & 0 & \cos(\theta_{L_x,k}) & 0 & ... & 0 \\ 0 & 0 & ... & 0 & 0 & 0 & ... & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & ... & 0 & 0 & 0 & ... & 1 \end{bmatrix} \cdot$$

$$\begin{bmatrix} \cos(\theta_{1,k}) & 0 & ... & 0 & -\sin(\theta_{1,k}) \\ 0 & 1 & ... & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & ... & 1 & 0 \\ \sin(\theta_{1,k}) & 0 & ... & 0 & \cos(\theta_{1,k}) \\ 0 & 1 & .... & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & ... & 1 & 0 \end{bmatrix} \cdot \tag{B.4}$$

$$\cos(\theta_{1,k}) = \frac{x_{1,k}}{\sqrt{\psi_{1,k-1}^2 + x_{1,k}^2}} \tag{B.5}$$

$$\sin(\theta_{1,k}) = \frac{\psi_{1,k-1}}{\sqrt{\psi_{1,k-1}^2 + x_{1,k}^2}} \tag{B.6}$$

B.3 RLS Convergence and Tracking

B.4 Block RLS Methods

Bibliography