mobile communications series

SOFTWARE-DEFINED RADIO for ENGINEERS

TRAVIS F. COLLINS
ROBIN GETZ
DI PU
ALEXANDER M. WYGLINSKI

Software-Defined Radio for Engineers

For a listing of recent titles in the *Artech House Mobile Communications*, turn to the back of this book.

Software-Defined Radio for Engineers

Travis F. Collins
Robin Getz
Di Pu
Alexander M. Wyglinski

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the U.S. Library of Congress.

British Library Cataloguing in Publication Data
A catalog record for this book is available from the British Library.

ISBN-13: 978-1-63081-457-1

Cover design by John Gomes

10 9 8 7 6 5 4 3 2 1

Dedication

To my wife Lauren
—Travis Collins

To my wonderful children, Matthew, Lauren, and Isaac, and my patient wife, Michelle—sorry I have been hiding in the basement working on this book. To all my fantastic colleagues at Analog Devices: Dave, Michael, Lars-Peter, Andrei, Mihai, Travis, Wyatt and many more, without whom Pluto SDR and IIO would not exist.
—Robin Getz

To my lovely son Aidi, my husband Di, and my parents Lingzhen and Xuexun
—Di Pu

To my wife Jen
—Alexander Wyglinski

Contents

CHAPTER 9

Channel Estimation and Equalization

This chapter will introduce the concepts of channel estimation and channel equalization. A simplified error model will be discussed along with several practical strategies for added equalizers to a communication system. Specifically, we will include designs for both decision direction- and training-based equalization. Equalizers are a useful utility since they can be used to handle numerous sources of distortion present in the environment as well as from the mismatches between nodes themselves.

With regard to our receiver outline in Figure 9.1, this chapter will address the last block, equalization, which is highlighted.

9.1 You Shall Not Multipath!

In the previous chapters, we focused on the synchronization between the transmitter and the receiving nodes. By combining the previous chapters, frame recovery now becomes possible and we have reached the threshold of successfully decoding frames. However, under certain scenarios these implementations will not be enough. Assuming that we have adequate SNR of the received signal, the remaining challenge in the environment is the multipath and other additive interferers. Multipath, which is introduced by the dispersive nature of the channel, is the effect of scaled reflections of a transmitted signal traveling along different paths to reach the receiver. Since these reflections travel along different paths they will observe different attenuations and different delays from the perspective of the receiver. Naturally, these impairments can be considered echoes of the transmitted signal and can be easily modeled by a FIR filter.

The time between the first received signal and the final received echo is defined as the *delay spread* of the channel [1]. In Figures 9.2 and 9.3, we present a physical representation of multipath and the resulting time domain representation of the line-of-sight (LOS) signal and two scatterers. This is a *ray-tracing* representation of multipath, but in reality multipath is a continuum of reflections due to the signal radiation characteristics. Ray-tracing is a discretization of that continuum and is commonly used to model multipath since it is far simplier to understand and mathematically model.

When the delay spread has a long duration with respect to the signal bandwidth, multipath effects can cause significant distortion to the received signal and results in intersymbol interference (ISI) as discussed in Chapter 6. The delay spread is a function of the environment, and we must take this into consideration when designing a system. For example, for outdoor environments where the multipath distances are large, this will produce a large delay spread. Mathematically, we can

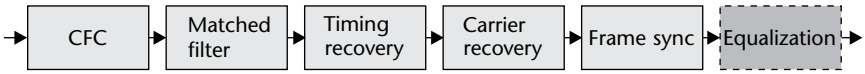

Figure 9.1 Receiver block diagram.

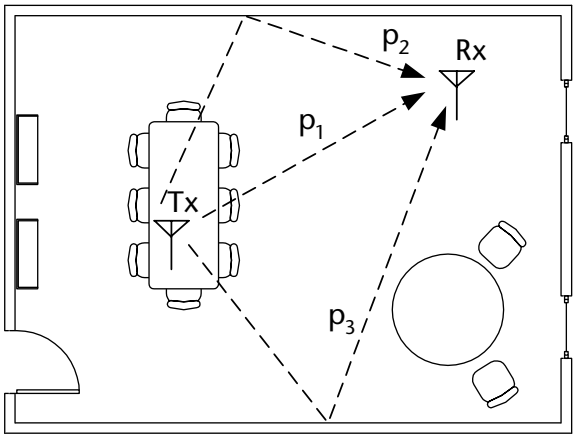

Figure 9.2 Example of multipath in an indoor environment.

relate the distance D that a scatter must travel to the sample delay t_s associated with that distance as

$$t_s = \frac{B \times D}{c}, \tag{9.1}$$

where c is the speed of light. A Wi-Fi signal at 20 MHz would have to travel approximately 15 extra meters to cause a single sample of delay. Therefore, Wi-Fi, which is commonly used indoors, will have a small delay spread. However, since the channel distances are short there can be a large number of high-powered scatterers. Due to path loss, the signal power of the interferers and delay spread are usually inversely related.

Mathematically, we can model a received multipath signal r as in impulse train at random time offsets Δ_n with associated gains α_n of the transmitted signal x as

$$r(t) = \mu(t) + \sum_{n=1}^{N} \alpha_n x(t - \Delta_n), \tag{9.2}$$

where there are $N-1$ scatters and μ is additional interferers or noise. As long as μ is uncorrelated with x and periodic or autoregressive, we can effectively filter it from the signal space [2]. In the case when we do have multipath, this will be experienced at the received as ISI. We can demonstrate such a condition in Figure 9.4 where we view a QPSK signal r. In Figure 9.4, we observe the effects of symbol smearing over time, which we view as symbol drift in the constellation diagram. Based on (9.2), we can simply model multipath effects using an FIR filter.

9.2 Channel Estimation

Before we consider correcting effects of a given channel, we can alternatively consider the estimation of an unknown channel. Channel estimation, as opposed to

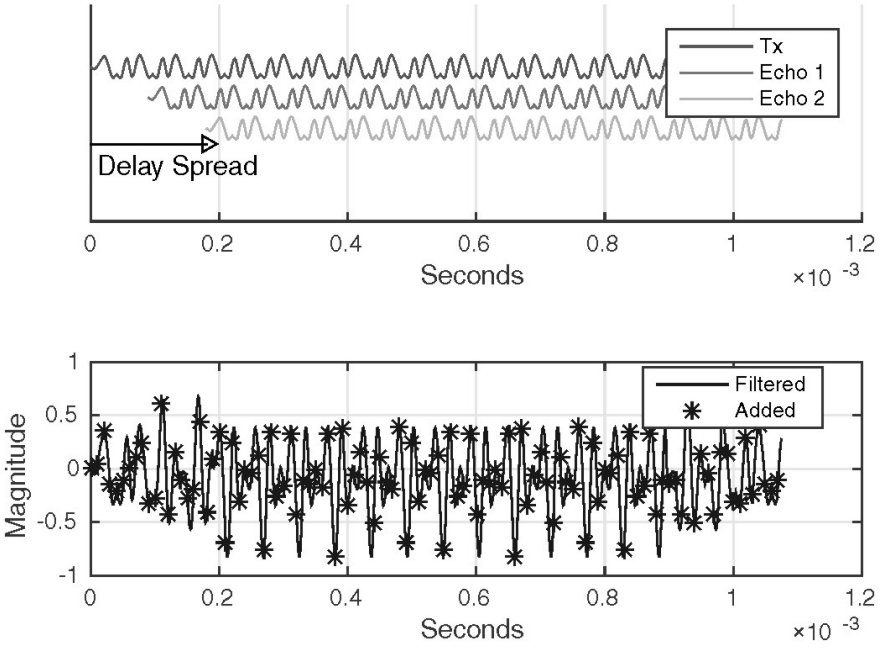

Figure 9.3 Physical characteristics of a multipath propagation environment.

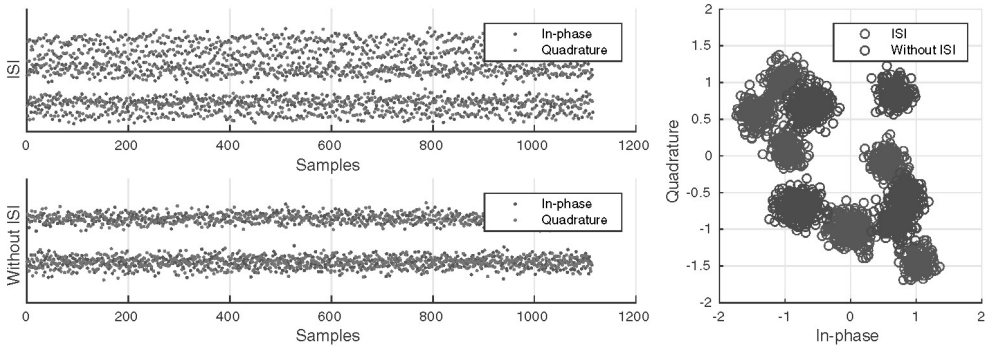

Figure 9.4 Effects of symbol smearing over time.

channel equalization, is a desirable place to start since we have a known solution in the simulation to test against, unlike equalization, which may not produce a unique solution depending on the channel and noise conditions. To perform channel estimation, we will utilize the least mean squares (LMS) algorithm developed by Widrow and Hoff [3]. LMS is a gradient descent or Newton method type algorithm, and can be considered the standard adaptive filter algorithm for signal processing. The LMS algorithm utilizes known information or symbols in the transmitted sequences in order to estimate the corruption of the receive data, which we model here as a FIR filter. We provide a model in Figure 9.5 of a common diagram for channel estimation, which we will use to help derive our system implementation. Alternative adaptive filter algorithms do exist, such as the recursive least squares (RLS) algorithm, which can outperform LMS in many situations. However, RLS can have stability concerns when designed improperly and is more computationally

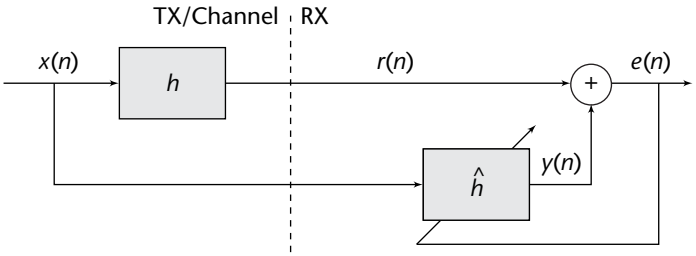

Figure 9.5 Adaptive FIR estimation of FIR channel h using training data.

complex due to a matrix inversion requirement. RLS is beyond the scope of this chapter, but Haykin [4] is a good reference.

We illustrate the pieces of our channel estimation system in Figure 9.5, where we have our unknown static channel $\mathbf{h}$, which affects our training data $x(t)$. The goal of the system shown in Figure 9.5 is to match $\mathbf{h}$ and $\hat{\mathbf{h}}$, which will drive our error down to zero. In order to achieve this goal, the adaptive algorithm will require an error signal $e(n)$ and the original signal $x(t)$. We will utilize LMS to estimate an unknown channel filter $\mathbf{h} \in^{\{L\times 1\}}$, whose estimate is defined as $\hat{\mathbf{h}} \in^{\{M\times 1\}}$ where $M \geq L$. For a transmitted signal $x(t)$ passing through the channel filter $\mathbf{h}$, $\mathbf{h}$ can be estimated using the following recursive algorithm:

$$y(n) = \hat{\mathbf{h}}^H(n)\mathbf{x}(n) \tag{9.3}$$

$$e(n) = r(n) - y(n) \tag{9.4}$$

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) + \mu\,\mathbf{x}(n)e^*(n) \tag{9.5}$$

where

$$\mathbf{x}(n) = [x(n), x(n-1), ..., x(n-M-1)]^T. \tag{9.6}$$

and μ is the governing stepsize, which provides control over convergence rate and stability. Selection of the stepsize is implementation-specific but should be in the range $0 < \mu < \frac{2}{\lambda_{max}}$, where λ_{max} is the maximum eigenvalue of the autocorrelation of the true channel $\mathbf{R} = \mathbf{E}[\mathbf{h}\mathbf{h}^H]$. Alternatively, since $\mathbf{h}$ is unknown, a looser and safer bound is $M\sigma^2$ where σ^2 is the variance of the $r(n)$. For more information on LMS stability analysis, the interested reader should consult Chapter 9 of Haykin [4].

The channel length L is an unknown for an individual environment, but through measurements and the physical nature of the channel environment, estimates can be made for reasonable lengths. This relates to the previous discussion on delay spread, which is an alternative description of the channel length. Studies are commonly performed by standards committees to provide guidance for receiver designers.

Equations (9.3) to (9.6) can be implemented in a simple recursion as in lines 8-17 in Code 9.1 requiring $2L+1$ multiplications and $L+1$ additions. If we examine the mean-squared error (MSE) of $\hat{\mathbf{h}}$ over time for a $L = M = 2$ across a contour of the solution space in Figure 9.6, we can observe the descent to the true solution LMS will take. Figure 9.6 demonstrates this descent for four random starting positions for the estimate $\hat{\mathbf{h}}$. The use of contour plots is a common analysis tool in adaptive filters theory to model behavior of estimation evolution. However, when $M > 2$ we

Code 9.1 LMS Channel Estimation: **`chanEst.m`**

```
h = [0.5; 1; -0.6]; % Channel to estimate
mu = 0.01; % Stepsize
trainingSamples = 1000;
x = sign(randn(trainingSamples,1)); % Generate BPSK data
r = filter(h,1,x); % Apply channel
L = length(h); h_hat = zeros(L,1);
%% Estimate channel
for n = L:trainingSamples
    % Select part of training input
    in = x(n:-1:n-L+1);
    % Apply channel estimate to training data
    y = h_hat'*in;
    % Compute error
    e = r(n)-y;
    % Update taps
    h_hat = h_hat + mu*conj(e)*in;
end
```

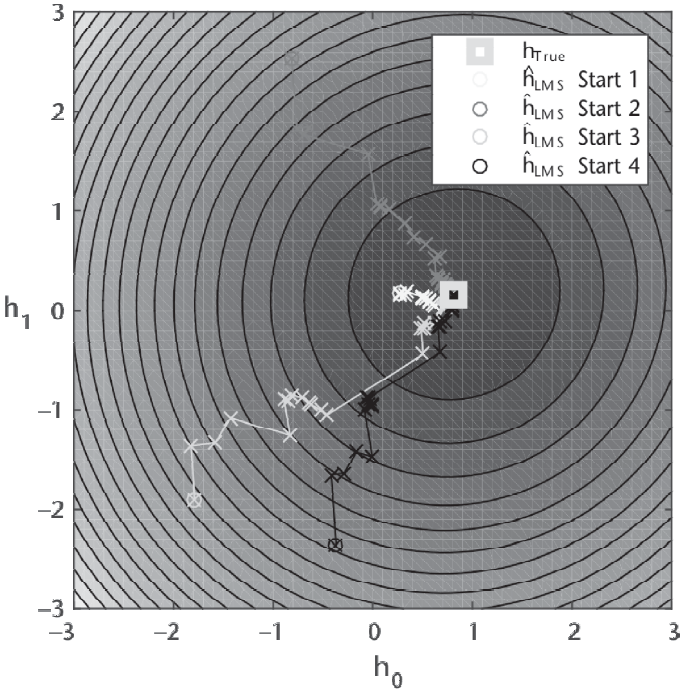

Figure 9.6 Contour of solution space for channel estimation when $M = 2$.

will use a MSE plot similar to Figure 9.7 since we cannot easily visualize all error dimensions on a contour.

We can extend Code 9.1 further and actually visualize the shape of the channel estimates as well to have better understanding of the accuracy of our designed system. Examining the response, especially for rather aggressive channels, can be very useful when determining parameterization of an equalizer design. In Code 9.2, we provide an example on how to visualize the channel responses and their

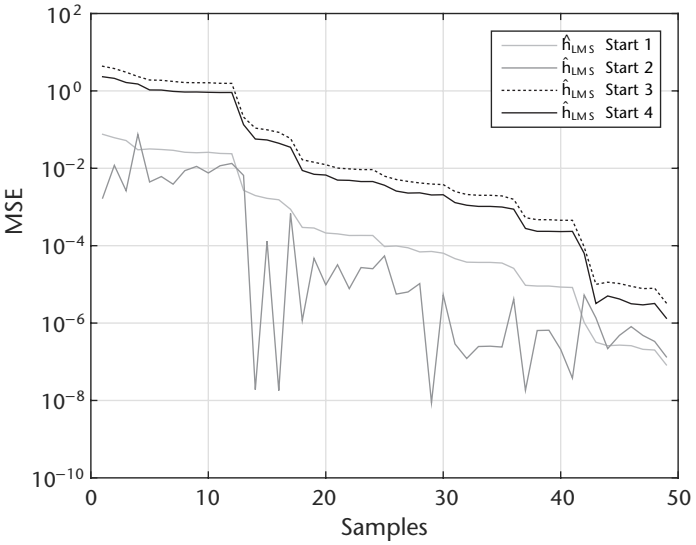

Figure 9.7 MSE of channel estimates over received samples from different starting locations.

estimates. The function `freqz` is utilized here, which will calculate the digital filter response using the `fft` function.

Code 9.2 Plot Responses: **`chanEst.m`**

```
%% Plot responses
Fs = 1e6; N = 64;
htrue=freqz(h,1,N,Fs,'whole');
[hb,we]=freqz(h_hat,1,N,Fs,'whole');
semilogy(we,abs(hb),'b')
hold on;semilogy(we,abs(htrue),'bo');hold off
grid on;xlabel('Frequency (Hz)');ylabel('Magnitude');
legend('Channel Est','Channel Actual','Location','Best');
```

The response and estimate provided in Figure 9.8 yields very accurate results. However, as noise is introduced into the system and the channel becomes more complex, this estimate will become worse. However, assuming the LMS algorithm has converged the error should only be limited based on our chosen stepsize μ and the noise of the channel.

9.3 Equalizers

Unlike channel estimation, an equalizer tries to undo the effects of the channel and remove interference if possible. Similar to channel estimation, some knowledge about the source data is necessary to train the equalizer at the receiver in order to reduce the channel effects. Typically, this information is part of the preamble sequence or header information of the frame since for logical operation we will always transmit some unknown data, which we call the payload. We will discuss several adaptive equalizer implementations here but many permutations and alternatives do exist in the literature [5].

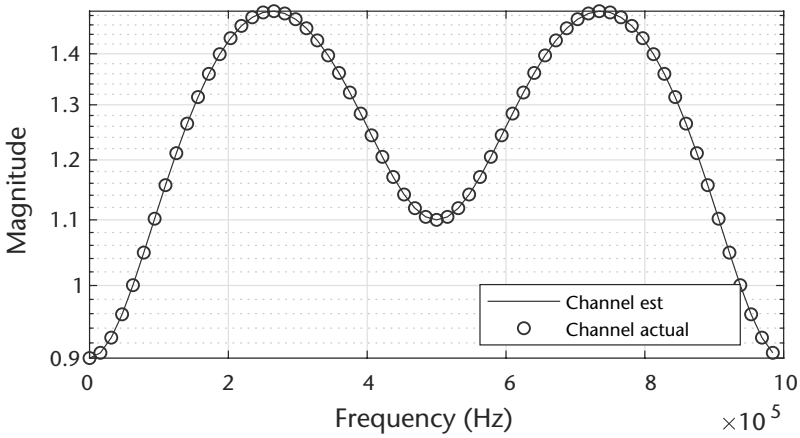

Figure 9.8 Example channel and associated estimate using the LMS algorithm.

The equalizer approaches the filter evolution problem from a different prespective than discussed in Section 9.2. Instead of evolving the filter to match the channel **h**, a filter **f** is evolved to compensate for the channel itself and reproduce the training data at the output of the filter accurately. This arrangement is provided in Figure 9.9, which places the equalizer filter or forward filter in the received data path. The known transmitted data is then used to adapt this filter. To solve for a filter $\mathbf{f} \in^{\{K\times 1\}}$ using LMS that will equalize the effects of a given channel **h**, we need to modify the recursion provided in (9.3) to (9.6) as

$$\hat{x}(n) = \mathbf{f}^H(n)\mathbf{r}(n) \tag{9.7}$$

$$e(n) = x(n) - \hat{x}(n) \tag{9.8}$$

$$\mathbf{f}(n+1) = \mathbf{f}(n) + \mu\, e^*(n)\mathbf{r}(n) \tag{9.9}$$

where:

$$\mathbf{r}(n) = [r(n), r(n-1), ..., r(n-K-1)]^T. \tag{9.10}$$

In this case, we are driving the received samples $r(n)$ to match the transmitted data $x(n)$. If this algorithm converges, the resulting output of the combined filters should just delay the signal:

$$x(n) * \mathbf{h} * \mathbf{f} = \hat{x}(n-\delta), \tag{9.11}$$

where $*$ represents the convolution operation and δ the group delay of the cascaded channel filter and equalizer. Therefore, the combined response of the equalizer and channel should be flat. The updated code is provided in Code 9.3.

The delay δ offsets the training sequence further into the filter, which is useful if the earliest filter taps do not produce the most gain. It can be argued this more efficiently utilizes the equalizer's additional tap advantage of the channel length. Since this equalizer must be a causal, $\delta > 0$ or else we would require knowledge of future samples (noncausal). Also, this delay will affect the output delay, which we must compensate for as well. To correctly evaluate the bit errors in the signal, use Code 9.4.

When we have exact knowledge of the source symbols, LMS is an effective algorithm to reach the Wiener solution for a given channel [4]. However, in many

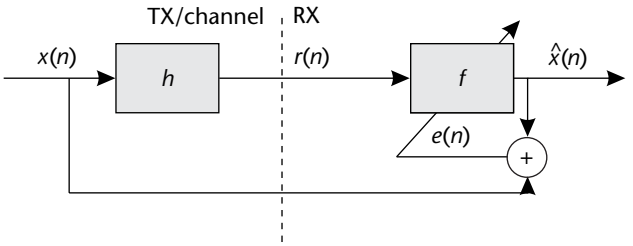

Figure 9.9 Adaptive FIR equalization of FIR channel h using known training data.

Code 9.3 Channel Equalization: **chanEQ.m**

```
h = [0.2; 1; 0.1; 0.02]; % Channel taps
mu = 0.001; % Stepsize
trainingSamples = 1e4;
x = sign(randn(trainingSamples,1)); % Generate BPSK data
r = filter(h,1,x); % Apply channel
K = length(h)+1; f = zeros(K,1);
delta = 3; % Reference tap
%% Equalize channel
index = 1;
[e,x_hat]=deal(zeros(trainingSamples-K+1,1));
for n = K:trainingSamples
    % Select part of training input
    in = r(n:-1:n-K+1);
    % Apply channel estimate to training data
    x_hat(index) = f'*in;
    % Compute error
    e = x(n-delta)-x_hat(index);
    % Update taps
    f = f + mu*conj(e)*in;
    index = index + 1;
end
```

Code 9.4 Determine Bit Errors: **chanEQ.m**

```
% Slice
x_bar = sign(x_hat);
% Calculate bit errors
eqDelay = K-delta;
fprintf('BER %2.4f n',mean(x(eqDelay:end-eqDelay-1) ~= x_bar));
```

cases it will not be possible to use training information and instead will require blind equalization techniques. *Blind equalization* is required when the received signal contains no known preamble or when equalizer updates want to be updated during the payload portions of frames. Such an implementation is necessary when channels have long filter lengths or require more data than provided in the preamble. Alternatively, for highly dynamic channels where the coherence of the channel is shorter than a frame, it may be necessary to update the equalizer to maintain a desired BER. Here we will consider a decision-directed (DD) LMS equalizer,

although other implementations exist, such as the *constant modulus* (CM) equalizer, sometimes called the *dispersion-minimization* (DM) algorithm [6].

The DD equalizer operates by making hard decisions on received data to estimate the most likely symbol and updates the equalizer based on this estimate. The updated equalizer structure is provided in Figure 9.10, which inserts a decision block after the output of the equalizer. Therefore, the receiver has no knowledge of $x(n)$ from this design except for the modulation scheme. This results in an error term in (9.12) to be updated in the case of DD to

$$e(n) = \hat{x}(n) - \bar{x}(n) \tag{9.12}$$

where $\bar{x}(n)$ is the maximum likelihood estimate of the equalized received signal $x(n)$. For the case of BPSK and QPSK we have the following estimators:

$$\bar{x}(n) = \begin{cases} sign(y(n)) & \text{if BPSK} \\ sign(real(x(n))) + i \times sign(imag(x(n))) & \text{if QPSK.} \end{cases} \tag{9.13}$$

DD equalization can be effective when $x(n)$ and $\hat{x}(n)$ are relatively close, but when the difference is large DD equalizer can perform poorly. Alternatively, from a perspective provided in Chapter 6, the eye of the eye diagram plot must be open to some degree initially for the DD equalization to effectively invert the channel effectively or in a reasonable amount of symbols. We provide an received signal with ISI in Figure 9.11(a), the resultant signal after LMS equalization in Figure 9.11(b), and the resultant signal after DD equalization in Figure 9.11(c). The LMS implementation convergences within a few hundred samples, while the DD equalizer take roughly 2,000 samples to open the eye with still a significant amount of noise. LMS has an obvious advantage, but requires knowledge of the exact source symbols unlike the DD method. Note that both equalizers utilize the same μ, with the initialization of the equalizers being equal to

$$\mathbf{f}_{LMS} = \mathbf{f}_{DD} = [1, 0, ..., 0] \in^{\{K \times 1\}}, \tag{9.14}$$

$\mathbf{f}_{DD}$ cannot be initialized to all zeros.

9.3.1 Nonlinear Equalizers

So far we have only considered equalizers with forward filters, also known as linear equalizers. However, it is also possible to implement filters with both feedback and feedforward filters, which we call here *decision feedback equalizers* (DFEs). The feedback filter in the DFE will produce outputs that are subtracted from the

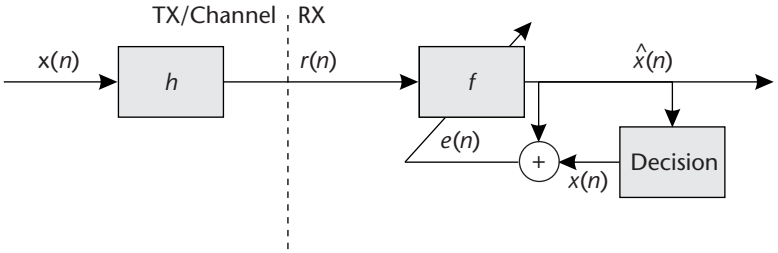

Figure 9.10 Adaptive FIR equalization of FIR channel h using known decision directed data.

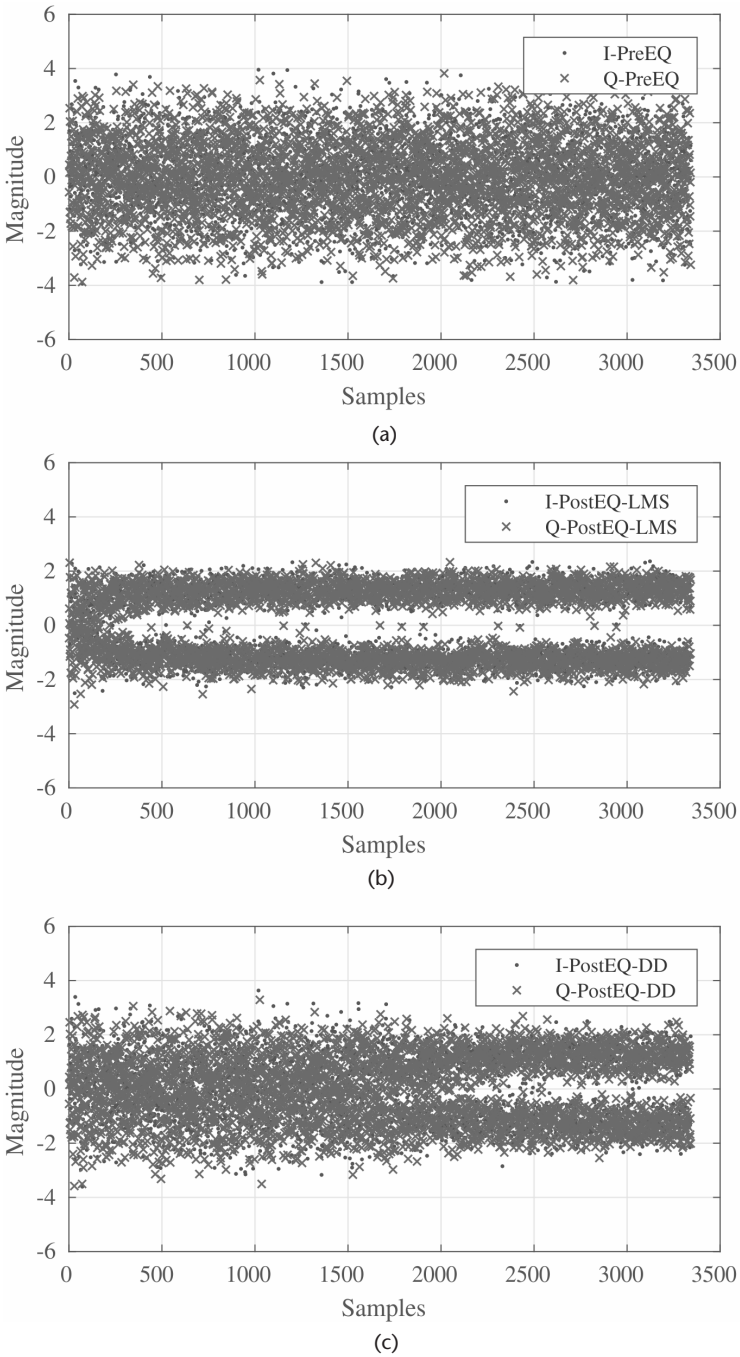

Figure 9.11 Example of using cross correlation to find a sequence with a larger sequence of data. (a) QPSK signal with ISI before equalization, (b) QPSK signal with ISI after LMS equalization, and (c) QPSK signal with ISI after DD equalization.

output of the feedforward filter. This is outlined in Figure 9.12, which shows both filters and their update paths. Since the feedback filter can only estimate the post-cursors symbols, it needs to be used in combination with a feedforward filter. After convergence, the feedback filter contains an estimate of the impulse response of the

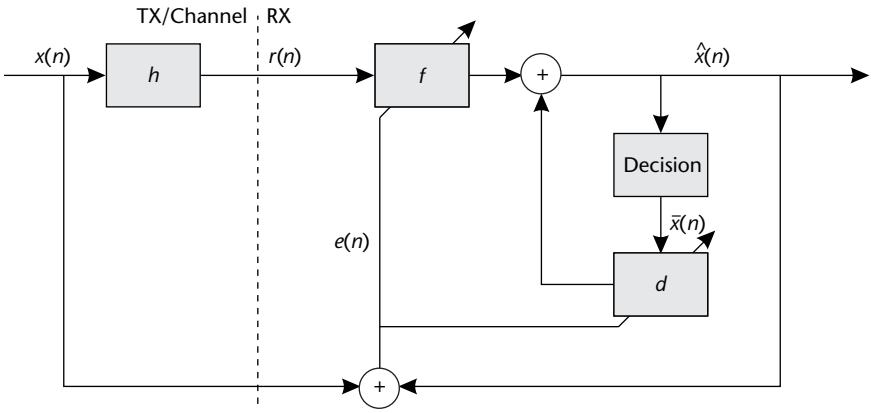

Figure 9.12 Adaptive FIR equalization of FIR channel h using decision feedback equalizer.

channel convolved with of the feedforward filter. Since the feedback filter utilizes this composite output, the DFE can compensate for severe amplitude distortion without increasing the noise in the highly distorted channel.

We update our previous equalizer equations with the addition of a new filter $\mathbf{d} \in^{\{P\times 1\}}$. To solve for a filter $\mathbf{d}$ using LMS that will equalize the effects of a given channel $\mathbf{h}$, we need to modify the recursion provided in (9.3) to (9.6) as

$$\hat{x}(n) = \mathbf{f}^H(n)\mathbf{r}(n) - \mathbf{d}^H(n)\bar{\mathbf{x}}(n) \tag{9.15}$$

$$e(n) = \bar{x}(n) - \hat{x}(n) \tag{9.16}$$

$$\mathbf{f}(n+1) = \mathbf{f}(n) + \mu\, e^*(n)\mathbf{r}(n) \tag{9.17}$$

$$\mathbf{d}(n+1) = \mathbf{d}(n) + \mu\, e^*(n)\bar{\mathbf{x}}(n) \tag{9.18}$$

where

$$\mathbf{r}(n) = [r(n), r(n-1), ..., r(n-K-1)]^T, \tag{9.19}$$

and

$$\bar{\mathbf{x}}(n) = [\bar{x}(n), \bar{x}(n-1), ..., \bar{x}(n-P-1)]^T. \tag{9.20}$$

Here $\bar{x}(n)$ will either be the DD version of $x(t)$ or $x(t)$ itself when training data is used. Again, we can update our MATLAB code to utilize this new feedback filter, which is provided in Code 9.5.

Note that the DFE implementation can be more difficult to tune since it contain two filters. The DFE implementation can also utilize different stepsizes per filter update as well, which may make tuning more managable.

9.4 Receiver Realization

For an actual implementation of an equalizer into our receiver structure, we have several design strategies that we can utilize depending on the system requirements. A reasonable design perspective to consider is the required amount of training data needed to equalize a given channel environment. There are three aspects here: the channel length, the convergence of the equalizer, and the dynamics of the channel. Training data is typically confined to the preamble sequence and the chosen length will be chosen on the maximum value of L, which also determines the value of M.

Code 9.5 Determine Bit Errors: **chanEQDFE.m**

```
h = [0.2; 1; 0.1; 0.02]; % Channel taps
mu = 0.001; % Stepsize
trainingSamples = 1e4;
x = sign(randn(trainingSamples,1)); % Generate BPSK data
r = filter(h,1,x); % Apply channel
K = length(h)+2; f = zeros(K,1);
P = length(h)-1; d = zeros(P,1); x_bar_vec = zeros(P,1);
delta = 4; % Reference tap
%% Equalize channel
index = 1;
[e,x_hat]=deal(zeros(trainingSamples-K+1,1));
for n = K:trainingSamples
    % Select part of training input
    in = r(n:-1:n-K+1);
    % Apply channel estimate to training data
    x_hat(index) = f'*in - d'*x_bar_vec;
    % Compute error
    e = x(n-delta)-x_hat(index);
    % Update taps
    f = f + mu*conj(e)*in;
    d = d - mu*conj(e)*x_bar_vec;
    % Update feedback filter
    x_bar_vec = [x(n-delta);x_bar_vec(1:end-1)];
    index = index + 1;
end
```

When the channel is dynamic, meaning that it can change over small periods of time when compared to the frame length, it becomes necessary to utilize multiple equalizers. For example, we can utilize a LMS equalizer to initially open the eye with the preamble sequence, then across the data we can utilize a DD equalizer to maintain the open eye (see Figure 9.13). If a secondary equalizer was not used, then the frame length would need to be shorted to meet the same BER requirements. However, the preamble must be long enough to allow for the appropriate adaption of the equalizer, which will be dependent on the values of M and L. For a larger equalizer, more data is required for it to converge, thus leading to longer preambles.

We can update the source code in Code 9.3 for a case of BPSK to utilize both LMS and DD with a few simple modifications. In Code 9.6, we simply add a condition to switch between the different modes.

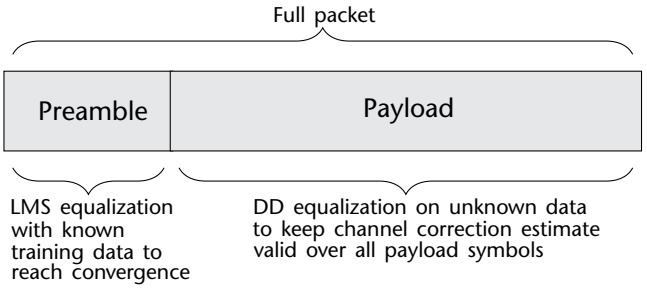

Figure 9.13 Example packet structure and application of different equalizer algorithms.

Code 9.6 Equalizer with DD and LMS Modes: **chanEQLMSDD.m**

```
for n = K+1:FrameLength
    % Apply equalizer to received data
    x_hat = f'*r(n:-1:n-K+1);
    % Estimate error
    if n<(PreambleLength-delta)
      e = x(i-delta) - x_hat;
    else
      e = sign(y) - x_hat;
    end
    % Update equalizer
    f = f + mu*conj(e)*r(n:-1:n-K+1);
end
```

Equalizers themselves are very useful utilities in the system beyond just compensating for multipath in a system. They can also be used to compensate for frequency offset, timing mismatches, and even frame synchronization errors. However, the equalizer must be configured in a way to compensate for such conditions. For example, when dealing with carrier offset, which can only be relatively small compared with conditions in Chapter 7, μ should be increased to deal with this condition. Since the equalizer is itself just a filter, compsenating for frequency offset simply forms the equalizer's taps in a complex gain equivalent to the instantaneous phase associated with the carrier. However, this must be updated continuously to compensate for such a condition, forcing a dual DD and training data implementation.

When considering timing compensation it can be useful to implement a fractional equalizer, which consumes multiple samples on the front feedforward filter. This makes the equalizer multirate, but also similar to the timing compensation designs in Chapter 6. In this configuration the equalizer will be able to weigh or interpolate across symbols to accurately correct for delay in the received signal. In addition to fractional timing offsets, by utilizing our delay variable δ we can compensate for sample errors during frame synchronization. As long as the desired sample, or start of frame sample, is within the feedforward filter we can compensate for an incorrect initial estimate. For a feedforward filter of length L and $\delta = \lfloor \frac{L}{2} \rfloor$, we should be able compensate for a sample offset of $\pm\delta$.

9.5 Chapter Summary

In this chapter we introduced the concept of channel estimation and equalization. We have built several equalizer designs around the gradient descent algorithm LMS, which is the dominant cost function in the field. We provided different strategies for implementing equalizers when training data is available and when blind operation is required. Finally, we provided a discussion on receiver implementation strategies that combine multiple equalizer designs as well as different use cases for nonidealities from the environment. For the interested reader, brief mathematical derivations for the basic linear equalizer design, zero-forcing equalizer, and decision feedback equalizer are available in Appendix C.

References

[1] Goldsmith, A., *Wireless Communications*, Cambridge, UK: Cambridge University Press, 2005.

[2] Johnson, C. R., Jr., W. A. Sethares, and A. G. Klein, *Software Receiver Design: Build Your Own Digital Communication System in Five Easy Steps*, Cambridge, UK: Cambridge University Press, 2011.

[3] Widrow, B., and M. E. Hoff, *Neurocomputing: Foundations of Research, Adaptive Switching Circuits*, Cambridge, MA: MIT Press, 1988, pp. 123–134.

[4] Haykin, S., and S. S. Haykin, *Adaptive Filter Theory*, Pearson Education, 2014.

[5] Farhang-Boroujeny, B., *Adaptive Filters: Theory and Applications*, Wiley, 1999.

[6] Johnson, R., P. Schniter, T. J. Endres, J. D. Behm, D. R. Brown, and R. A. Casas, Blind Equalization Using the Constant Modulus Criterion: A Review, *Proceedings of the IEEE*, Vol. 86, No. 10, 1998, pp. 1927–1950.