Chapter 7

Communication over Dispersive Channels: part 1

From **Introduction to Communication Systems**

From the material in Chapters 4-6, we now have an understanding of commonly used modulation formats, noise models, and optimum demodulation for the AWGN channel model. It is time to discuss how to model and handle channel impairments. In this chapter, we consider the following basic model for a *dispersive* channel: the transmitted signal passes through a linear time-invariant system, and is then corrupted by white Gaussian noise. The LTI model is broadly applicable to wireline channels, including copper wires, cable and fiber optic communication (at least over shorter distances, over which fiber nonlinearities can be neglected), as well as to wireless channels with quasi-stationary transmitters and receivers. For wireless mobile channels, the LTI model is a good approximation over durations that are small compared to the time constants of mobility, but still fairly long on an electronic timescale (e.g., of the order of milliseconds). Methods for compensating for the effects of a dispersive channel are generically termed *equalization,* and our goal in this chapter is to provide a hands-on understanding of some common design approaches for equalization.

We describe here time domain equalization for linearly modulated signals, which incur inter-symbol interference (ISI) when they pass through a dispersive channel. We discuss equalization techniques which are suboptimal from the point of view of minimizing error probability, but are intuitively appealing and less computationally complex than optimum equalization. (We refer the reader to more advanced texts for discussion of optimum equalization and its performance analysis.) We discuss a geometric view of the equalization problem, in which a desired signal vector corresponding to the symbol we wish to demodulate is corrupted by interfering signal vectors corresponding to interfering symbols. We then introduce linear and decision feedback equalizers. While the equalizer parameters depend on system parameters such as the channel impulse response, the transmit filter, the receive filter and the SNR, we introduce adaptive equalizer implementations, based on the Minimum Mean Squared Error (MMSE) criterion, that do not require explicit knowledge or estimates of these system parameters. Such MMSE equalizers can be computed directly from the noisy received signal, given a known sequence of training symbols.

7.1 Singlecarrier System Model

We first provide a system-level overview of linear modulation over a dispersive channel. Figure **??** shows block diagrams corresponding to a typical DSP-centric realization of the transceiver. The DSP operations are performed on digital streams at an integer multiple of the symbol rate, denoted by m/T. For example, we might choose $m = 4$ for implementing the transmit and receive filters, but we might subsample the output of the receive filter down to $m = 2$ before implementing an equalizer. We model the core components of such a system using the complex baseband representation, as shown in Figure **??**. Given the equivalence of passband and complex baseband, we are only skipping modeling of finite precision effects due to digital-to-analog conversion (DAC) and analog-to-digital conversion (ADC). These effects can easily be incorporated into models such as those we will develop, but are beyond our current scope.

We focus on a hands-on development of the key ideas using discrete time simulation models, illustrated by code fragments.

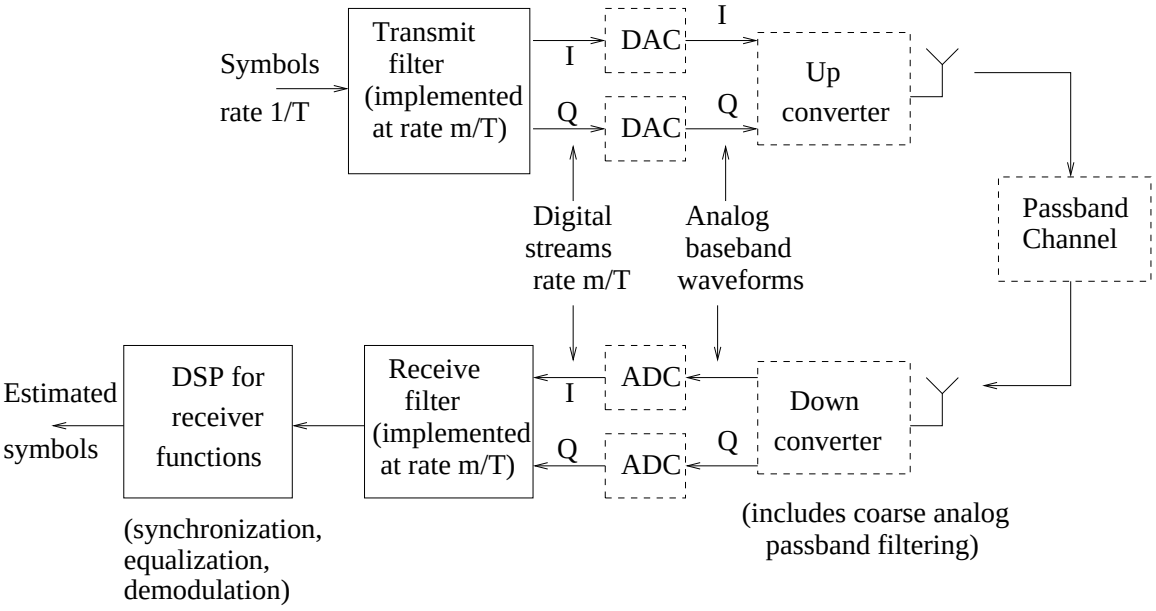


Figure 7.1: Typical DSP-centric transceiver realization. Our model does not include the blocks shown in dashed lines. Finite precision effects due to digital to analog conversion (DAC) and analog to digital conversion (ADC) are not considered. The upconversion and downconversion operations are not modeled. The passband channel is modeled as an LTI system in complex baseband.

7.1.1 Signal Model

We begin with an example of linear modulation, to see how ISI arises and can be modeled. Consider linear modulation using BPSK with a sine pulse, which leads to a transmitted baseband waveform shown in Figure **??**(a). The Matlab code used for generating this plot is given below. We have sampled much faster than the symbol rate (at $32/T$) in order to obtain a smooth plot. In practice, we might sample at a smaller multiple of the symbol rate (e.g. at $4/T$) to generate the input to the DAC in Figure **??**.

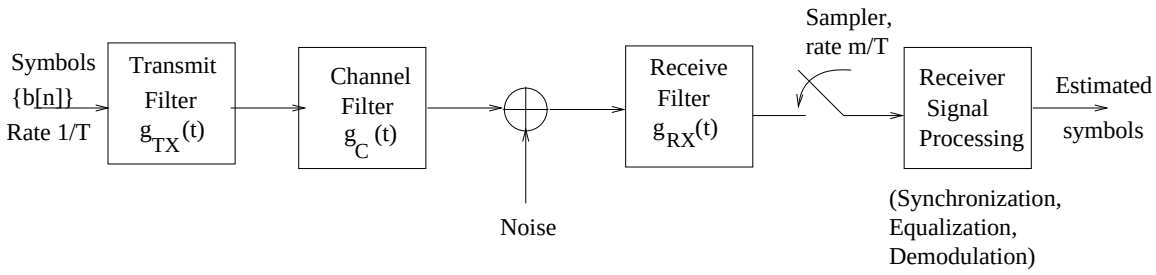
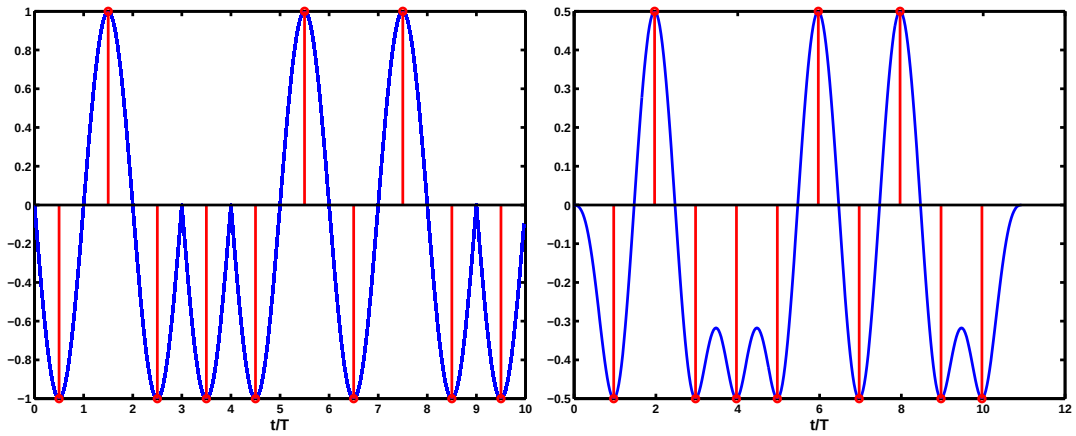


Figure 7.2: Block diagram of a linearly modulated system, modeled in complex baseband.



(a) Transmit filter output.

(b) Receive filter output.

Figure 7.3: The outputs of the transmit and receive filter without channel dispersion. The symbols can be read off from sampling each waveform at the times indicated by the stem plot.

We provide Matlab code fragments that convey the concepts underlying discrete-time modeling and implementation. The code fragments also show how some of the plots here are generated, with cosmetic touches omitted.

Code fragment 1 (transmitted waveform): This shows how to work with discrete time samples using oversampling at rate m/T, including how to generate the plot of the transmitted waveform in Figure **??**(a).

```
%choose large oversampling factor for smooth plots
oversampling_factor = 32;
m = oversampling_factor; %for brevity
%generate sine pulse
time_over_symbol = cumsum(ones(m,1))-1;
transmit_filter = sin(time_over_symbol*pi/m);
%number of symbols
nsymbols = 10;
%BPSK symbol generation
symbols = sign(rand(nsymbols,1) -.5);
%express symbol sequence at oversampled rate using zeropadding,
%(starts and ends with nonzero symbols)
Lpadded = m*(nsymbols -1)+1; %%length of zeropadded sequence
symbolspadded = zeros(Lpadded,1); %%initialize
symbolspadded (1:m:Lpadded) = symbols; %%fill in bit values every m entries
%%now all convolutions can be performed in oversampled domain
transmit_output = conv(symbolspadded,transmit_filter);
%plot transmitted waveform and sampling times
t1 = (cumsum(ones(length(transmit_output)))-1)/m;
figure;
plot(t1,transmit_output,'b');
xlabel('t/T');
hold on;
%choose sampling times in accordance with peak of transmit filter response
[maxval maxloc] = max(transmit_filter); %find peak location
sampling_times = maxloc:m:(nsymbols-1)*m+maxloc;
sampled_outputs = transmit_output(sampling_times);
stem((sampling_times-1)/m,sampled_outputs,'r')
hold off;
```

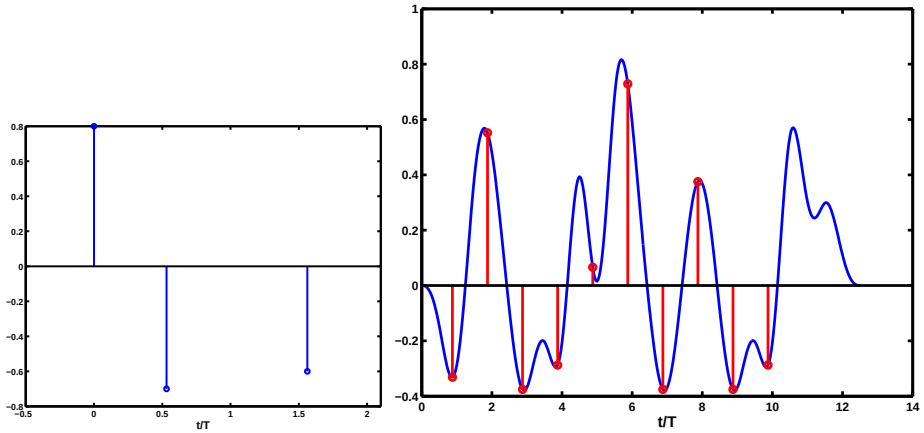
If this waveform now goes through an ideal channel, and we use a receive filter with impulse response matched to the transmitted pulse, then the waveform we obtain is shown in Figure **??**(b). The transmit filter impulse response is time limited to length T and hence square root Nyquist (see Chapter 4), hence the net response to a single symbol, which is a cascade of the transmit filter with its matched filter, is Nyquist. It follows that, by sampling at the right moments (as marked on the plot), we can recover the symbols exactly.

Code fragment 2 (modeling the channel and receive filter): Appending this to code fragment 1 generates and plots the noiseless received waveform. This fragment can be employed for modeling both ideal and dispersive channels.

```
dispersive = 0; %set this to 0 for ideal channel, and to 1 for dispersive channel
if dispersive == 0,
channel = 1;
else
channel = [0.8;zeros(m/2,1);-0.7;zeros(m,1);-0.6]; %or substitute your favorite choice of di
end
%noiseless receiver input
receive_input = conv(transmit_output,channel);
t2 = (cumsum(ones(length(receive_input)))-1)/m;
figure;
plot(t2,receive_input);
xlabel('t/T');
%receive filter matched to transmit filter
%(would also need to conjugate if complex-valued)
receive_filter = flipud(transmit_filter);
%receive filter output (normalized to account for oversampling)
receive_output = (1/m)*conv(receive_input,receive_filter);
t3 = (cumsum(ones(length(receive_output),1))-1)/m;
%plot receive filter output together with sample locations chosen based on peak of net respo
figure;
plot(t3,receive_output,'b');
xlabel('t/T');
hold on;
%effective pulse at channel output
pulse = conv(transmit_filter,channel);
%effective pulse at receive filter output (normalized to account for oversampling)
rx_pulse = conv(pulse,receive_filter)/m;
[maxval maxloc] = max(rx_pulse);
rx_sampling_times = maxloc:m:(nsymbols-1)*m+maxloc;
rx_sampled_outputs = receive_output(rx_sampling_times);
stem((rx_sampling_times-1)/m,rx_sampled_outputs,'r');
hold off;
```

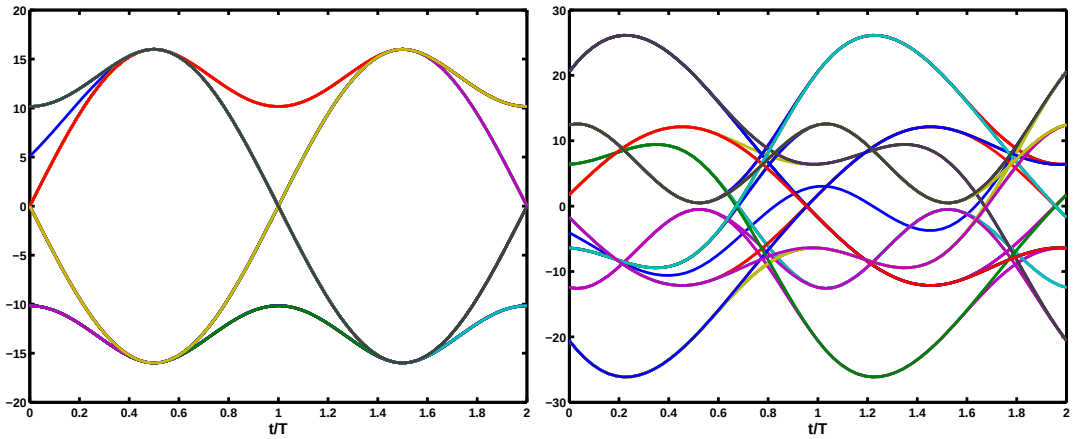
Figure **??** shows a dispersive channel and the corresponding noiseless receive filter output. The effective pulse given by the cascade of the transmit, channel and receive filters is no longer Nyquist, hence we do not expect a symbol decision based on a single sample to be reliable. Figure **??**(b) shows the severe distortion due to ISI with a "best effort" choice of sampling times (chosen based on the peak of the effective pulse). In particular, for the specific symbol sequence shown, one (out of ten) of the symbol estimates obtained by taking the signs of these samples.

Eye diagrams: A classical technique for visualizing the effect of ISI is the eye diagram. It is constructed by overlapping multiple segments of the received waveform over a fixed window, which tells us how different combinations of symbols could potentially create ISI. For an ideal channel and square root Nyquist pulses at either end, the eye is *open,* as shown in Figure **??**(a). However, for the dispersive channel in Figure **??**(b), we see from Figure **??**(b) is *closed*. An open eye implies that, by an appropriate choice of sampling times, we can make reliable single-sample symbol decisions, while a closed eye means that more sophisticated equalization techniques are needed for symbol recovery. Physically, an eye diagram can be generated using an oscilloscope



(a) Dispersive channel. (b) Receive filter output.

Figure 7.4: When the transmitted waveform passes through the dispersive channel shown, we can no longer read off the symbols reliably by sampling the output of the receive filter. For this particular set of symbols, one of the symbols is estimated incorrectly, even though there is no noise.



(a) Ideal channel. (b) Dispersive channel.

Figure 7.5: Eye diagrams with and without channel dispersion. The eye is closed for the channel considered, which means reliable symbol decisions are not possible without equalization.

with the baseband modulated signal as the vertical input, with horizontal sweep triggered at the symbol rate. A code fragment for generating the eye pattern from discrete time samples is given below.

Code fragment 3 (eye diagram): While Matlab has its own eye diagram routine, we provide a code fragment for generating an eye diagram from samples at rate m/T in order to clearly convey the concept. The output of the receive filter generated in code fragment 2 is the input to this fragment, but in general, we could plot an eye diagram based on the baseband waveform at any stage in the system. For complex baseband signals, we would plot the eye diagrams for the I and Q components separately.

```
%remove edge effects before doing eye diagram
r1 = receive_out(m/2:m/2+nsymbols*m-1);
%horizontal display length in number of symbol intervals
K=2;
%break into non-overlapping traces
R1=reshape(r1,K*m,length(r1)/(K*m));
%now enforce continuity across traces
%(append to each trace the first element of the next trace)
row1 = R1(1,:);
L=length(row1);
row_pruned = row1(2:L);
R_pruned = R1(:,1:L-1);
R2 = [R_pruned;row_pruned];
time = (0:K*m)/m; %time as a multiple of symbol interval
plot(time,R2);
xlabel('t/T');
```

7.1.2 Noise Model and SNR

In continuous time, our model for the noisy input to the receive filter is

$$y(t) = \sum_n b[n]p(t - nT) + n(t) \tag{7.1}$$

where $p(t) = (g_{TX} * g_C)(t)$ is the "effective pulse" given by the cascade of the transmit pulse and the channel filter, $\{b[n]\}$ is the symbol sequence, which is in general complex-valued, and $n(t)$ is complex WGN with PSD $\sigma^2 = \frac{N_0}{2}$. We translate this model directly into discrete time by constraining $t = kT/m + \tau$, where m/T is the sampling rate (m a positive integer) and τ equals the sampling offset. The noise at the input to the receive filter is now modeled as discrete time white Gaussian noise (WGN) with variance $\sigma^2 = \frac{N_0}{2}$ per dimension. As we well know from Chapter 6, the absolute value of the noise variance is meaningless unless we also specify the signal scaling, hence we fix either the signal or noise strength, and set the other based on SNR measures such as E_b/N_0 or E_s/N_0. Here $E_s = \mathbb{E}[|b[n]|^2]||p||^2$ for the model (**??**), and $E_b = E_s/\log_2 M$ as usual, where M is the constellation size. Inner products and norms are computed in discrete time.

Note that, with the preceding convention, the noise energy in a fixed time interval scales up with the sampling rate, and so does the signal energy (since we have more samples whose energies

we are adding up), with the SNR converging to the continuous-time SNR as the sampling rate gets large. However, for a sampling rate that is a small multiple of the symbol rate, the SNR for the discrete time system can, in general, be different from that in the original continuous time system. We do not worry about this distinction here. The following code fragment illustrates adding noise to our simulation model.

Code fragment 4 (noise modeling): We add discrete time WGN to the receive filter input, and get colored noise at the output which we add to the signal component already computed in code fragment 2.

```
bn_energy = 1;% for BPSK with current normalization
Es = bn_energy*(pulse'*pulse); %pulse is cascade of transmit and channel filters
constellation_size=2; %for BPSK
Eb = Es/log2(constellation_size);
%specify Eb/N0 in dB
ebnodb=5;
ebnoraw = 10^(ebnodb/10); %raw Eb/N0
N0=Eb/ebnoraw;
%noise standard deviation per dimension
sigma = sqrt(N0/2);
%noise at input to receive filter
%(would also need to add an imaginary component for complex-valued signals)
noise_receive_input = sigma*randn(size(receive_input));
%noise at output of receive filter
noise_receive_output = (1/m)*conv(noise_receive_input,receive_filter);
%noisy receive filter output
receive_output_noisy = receive_output + noise_receive_output;
```

7.2 Linear equalization

We have seen that single-sample symbol decisions are unreliable when the eye is closed. However, what if we are willing to use multiple samples for each symbol decision? Typically, the transmitter and receiver may implement fixed filters in DSP at a faster sampling rate than the sampling rate used eventually for equalization. Thus, suppose that we have samples at rate m/T from the output of the receive filter, but we now wish to use rate q/T samples for equalization, where q divides m. For example, we may have $m = 4$ and $q = 2$. We subsample the output of the receive filter, taking one out of every m/q samples, and then use L consecutive samples, collected into a vector $\mathbf{r}[n]$, to make a decision on symbol $b[n]$. We would want to choose these samples so that the bulk of the response due to $b[n]$ falls within the *observation interval* over which we collect these samples. When we want to make a decision on the next symbol $b[n+1]$, we must slide this observation interval over by T in order to obtain the received vector $\mathbf{r}[n+1]$. Since our sampling rate is now q/T, this corresponds to an offset of q samples between successive observation intervals. Note that an observation interval typically spans multiple symbol intervals, so that successive observation intervals will overlap: for example, for $L = 6$ and $q = 1$, $\mathbf{r}[n]$ and $\mathbf{r}[n+1]$ overlap by $L - q = 5$ samples, whereas for $L = 6$ and $q = 2$, $\mathbf{r}[n]$ and $\mathbf{r}[n+1]$ overlap by $L - q = 4$ samples. This is illustrated in Figure **??**. Note that $q = 1$ is termed symbol-spaced

equalization (sampled spaced by symbol time T), while $q > 1$ is termed fractionally spaced equalization (samples spaced by a fraction $1/q$ of the symbol time).

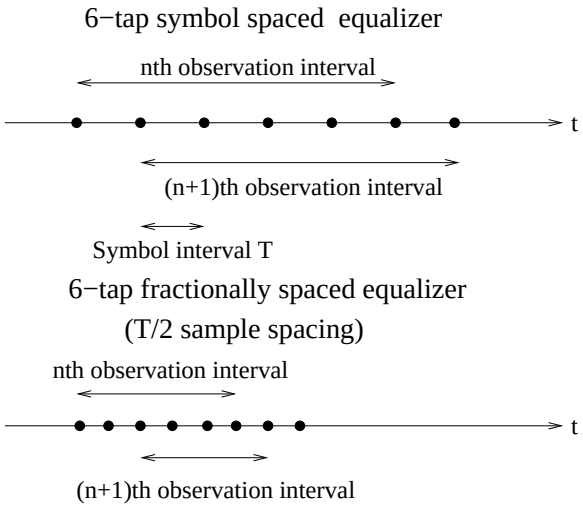


Figure 7.6: Sampling times (denoted by filled circles) and observation intervals for a symbol-spaced and fractionally spaced equalizer.

The transmit, channel and receive filters are LTI systems and the noise is stationary, and successive symbols are input to the system spaced by time T. Since the discrete time symbol sequence is stationary as well, the statistics of the signal at any stage of the system are invariant to shifts by integer multiples of T. Such periodicity in the statistics is termed *cyclostationarity.* This implies that the statistics of the noise and ISI seen in different observation intervals are identical: the only change is in which symbol plays the role of desired symbol. Thus, if we have a strategy for handling ISI over a given observation interval, it will work for other observation intervals as well. This opens up the possibility of realizing *adaptive* equalizers which can learn enough about the statistics of the ISI and noise to compensate for them.

We focus here on *linear* equalization, which corresponds to using the decision statistic $\mathbf{c}^T\mathbf{r}[n]$ to estimate $b[n]$, where $\mathbf{c}$ is an appropriately chosen correlator. The choice of $\mathbf{c}$ can be independent of n, by virtue of cyclostationarity. For BPSK signaling, for example, this leads to a decision rule

$$\hat{b}[n] = \text{sign}\left(\mathbf{c}^T\mathbf{r}[n]\right) \tag{7.2}$$

7.2.1 MMSE Equalizer

While constraining ourselves to linear equalization is suboptimal (discussion of optimal equalization is beyond our present scope), we can try to optimize $\mathbf{c}$ to combat ISI and noise. In particular, the linear MMSE criterion corresponds to choosing $\mathbf{c}$ so as to minimize the mean squared error (MSE) between the decision statistic and the desired symbol, defined as

$$MSE = J(\mathbf{c}) = \mathbb{E}\left[(\mathbf{c}^T\mathbf{r}[n] - b[n])^2\right] \tag{7.3}$$

Minimizing the MSE in this fashion leads to minimizing the contribution due to ISI and noise at the correlator output, which is clearly a desirable outcome.

The MSE is a quadratic function of $\mathbf{c}$, and can therefore be minimized by setting its gradient with respect to $\mathbf{c}$ to zero. Due to linearity, the gradient can be taken inside the expectation, and we obtain (see Problem **??**)

$$\nabla_{\mathbf{c}} J(\mathbf{c}) = 2\mathbb{E}\left[\mathbf{r}[n](\mathbf{c}^T\mathbf{r}[n] - b[n])\right] = 2\mathbb{E}\left[\mathbf{r}[n](\mathbf{r}^T[n]\mathbf{c} - b[n])\right]$$

Defining

$$\mathbf{R} = \mathbb{E}\left[\mathbf{r}[n]\mathbf{r}^T[n]\right] \ , \quad \mathbf{p} = \mathbb{E}\left[b[n]\mathbf{r}[n]\right] \tag{7.4}$$

we can rewrite the gradient of the MSE as

$$\nabla_{\mathbf{c}} J(\mathbf{c}) = 2\left(\mathbf{R}\mathbf{c} - \mathbf{p}\right) \tag{7.5}$$

Setting the gradient to zero yields the following expression for the MMSE correlator:

$$\mathbf{c}_{MMSE} = \mathbf{R}^{-1}\mathbf{p} \tag{7.6}$$

In order to compute this, we must know, or be able to estimate, the expectations in (**??**). If we know the transmit filter, the channel filter, the receive filter, the sampling times, and the noise PSD, we can compute these expectations using a model such as (**??**). However, we often do not have explicit knowledge of one or more of these quantities. Thus, an attractive approach in practice, therefore, is to exploit the stationarity of the model as we vary n to estimate expectations using their *empirical averages.* These expectations involve the received vectors $\mathbf{r}[n]$, which we of course have access to, and the symbols $b[n]$, which we assume we have access to over a *training* period in which a known sequence of symbols is transmitted. This approach leads to *adaptive* equalization techniques that do not require explicit knowledge or estimates of the model parameters.

Least Squares Adaptation: Assuming that the first *ntraining* symbols are known, least squares adaptation corresponds to replacing the expectations in (**??**) by their empirical averages as follows:

$$\hat{\mathbf{R}} = \frac{1}{ntraining}\sum_{n=1}^{ntraining}\mathbf{r}[n]\mathbf{r}^T[n] \ , \quad \hat{\mathbf{p}} = \frac{1}{ntraining}\sum_{n=1}^{ntraining} b[n]\mathbf{r}[n] \tag{7.7}$$

where the normalization by $\frac{1}{ntraining}$ is not needed, but is put in to make the averaging interpretation transparent. The MMSE correlator is now approximated by the least squares solution:

$$\hat{\mathbf{c}}_{LS} = (\hat{\mathbf{R}})^{-1}\hat{\mathbf{p}} \tag{7.8}$$

This correlator can now be used to make decisions on the unknown symbols following the training period. It can be checked that the preceding solution minimizes the empirical MSE over the training period:

$$\hat{MSE} = \sum_{n=1}^{ntraining}\left(\mathbf{c}^T\mathbf{r}[n] - b[n]\right)^2$$

Other, closely related, approaches to adaptation are discussed in Problems **??** and **??**.

Filter implementation of linear equalization: For conceptual clarity, we have introduced linear equalization as a correlator operating on received vectors, as in (**??**) and (**??**), obtained by windowing the samples at the output of the receive filter. However, an efficient technique for generating the decision statistics $\mathbf{c}^T\mathbf{r}[n]$ is by passing the received samples through a discrete

time filter matched to **c**, and then subsampling the output at the symbol rate with an appropriate delay.

Code fragment 5 (adaptive equalization): The following code fragment implements and tests least squares adaptation, comparing it with unequalized estimates obtained by sampling at the peaks of the net response to a symbol.

```
%Use code fragments 1 and 2 with a large value of nsymbols
%(first ntraining symbols assumed to be known)
%Insert noise using code fragment 4
%downsample to q/T to get input to equalizer
q=2;
r = receive_output_noisy(1:m/q:length(receive_output_noisy));
%figure out net response to a single symbol at receive filter output
rx_pulse = (1/m)*conv(pulse,receive_filter);
%effective response after downsampling
h= rx_pulse(1:m/q:length(rx_pulse));
%set equalizer length
L=6;
%choose how to align correlator (e.g., to maximize desired vector energy)
desired_energy = conv(h.^2,ones(L,1));
[max_energy loc_max_energy] = max(desired_energy);
%choose offset to align correlator with desired vector to maximize energy
offset = max(loc_max_energy-L,0)
%another option: set equalizer length equal to effective response
%L=length(h);
%offset = 0;
%initialize for least squares adaptation
phat = zeros(L,1);
Rhat = zeros(L,L);
for n = 1:ntraining,
    rn=r(1+q*(n-1)+offset:L+q*(n-1)+offset); %current received vector r[n]
    phat = phat + symbols(n)*rn;
    Rhat = Rhat + rn*rn';
end
%least squares estimate of MMSE correlator
cLS = Rhat\phat; %often more stable computation than inv(Rhat)*phat
%implement equalizer as filter
h_equalizer = flipud(cLS); %would also need conjugation for complex signals
equalizer_output = conv(r,h_equalizer);
%sample filter output at symbol rate after appropriate delay
delay = length(h_equalizer)+offset;
%symbol decision statistics
decision_stats = equalizer_output(delay:q:delay+(nsymbols-1)*q);
%payload = non-training symbols
payload = symbols(ntraining+1:nsymbols);
%estimate of payload (for BPSK)
payload_estimate = sign(decision_stats(ntraining+1:nsymbols));
```

```
%number of errors
nerrors = sum(ne(payload,payload_estimate))
%COMPARE WITH UNEQUALIZED ESTIMATES
%unequalized estimates obtained by sampling at peaks of effective response
[maxval maxloc] = max(h);
sampling_times = maxloc:q:(nsymbols-1)*q+maxloc;
unequalized_decision_stats = r(sampling_times);
sampled_outputs = transmit_output(sampling_times);
%estimate of payload (for BPSK)
payload_estimate_unequalized = sign(unequalized_decision_stats(ntraining+1:nsymbols));
%number of errors
nerrors_unequalized = sum(ne(payload,payload_estimate_unequalized))
```

Putting code fragments 1, 2, 4 and 5 together, we have a simulation model for adaptive linear equalization over a dispersive channel. As a quick example, for the dispersive channel considered, at $E_b/N0$ of 7 dB, we estimate (using nsymbols = 10000, ntraining = 100) the error probability after equalization at rate $2/T$ ($q = 2$) to be about 3.5×10^{-3} and the unequalized error probability to be about 0.16. Linear equalization in this case is quite effective, although exhibiting some degradation relative to the ideal BPSK error probability of 7.7×10^{-4}. We can now build on this code base to run a variety of experiments, as suggested in the problems: for example, probability of error as a function of E_b/N_0 for different equalizer lengths, for different channel models, and for different choices of the transmit and receive filters. Our model extends easily to complex-valued constellations, as discussed below.

Extension to complex-valued signals: All of the preceding development goes through for complex-valued constellations and signals, except that vector transposes $\mathbf{x}^T$ are replaced by conjugate transposes $\mathbf{x}^H$. We skip derivations, and state that the decision statistics are given by $\mathbf{c}^H\mathbf{r}[n]$, the MSE expression is

$$MSE = J(\mathbf{c}) = \mathbb{E}\left[(\mathbf{c}^H\mathbf{r}[n] - b[n])^2\right]$$

and the MMSE solution is given by (**??**) as before, with

$$\mathbf{R} = \mathbb{E}\left[\mathbf{r}[n]\mathbf{r}^H[n]\right] \ , \quad \mathbf{p} = \mathbb{E}\left[b^*[n]\mathbf{r}[n]\right] \tag{7.9}$$

As before, these statistical expectations can be replaced by empirical averages for a least squares implementation.