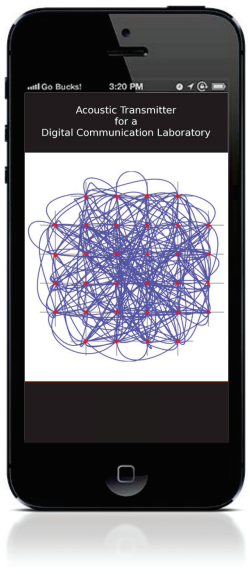
A Digital Communication Laboratory

Implementing a Software-Defined Acoustic Modem



Lee C. Potter and Yang Yang

This volume was typeset by the authors in LaTeX.

First edition, December 2015

The routine `plottf.m` and Figures 1.5, 1.7, 2.1, 2.2, 4.2, and 4.4 are adapted from [31] with permission from the author. Figure B.5 is provided courtesy of Aaron Wise; Figure B.6 is used with permission.

Contents

Preface

This text is intended as a student guide for hands-on exploration of physical layer communication. Through a sequence of guided explorations, students design and implement a baseband digital communication system with modulation to an acoustic carrier frequency. The acoustic operation allows students to hear, see, and wirelessly transmit signals using readily available, low-cost hardware, such as a PC with sound card or a smartphone. Acoustic operation, while readily accessible, nonetheless presents a student with the channel impairments and synchronization issues encountered in radio frequency systems.

Our approach is based on experiences, both successful and disappointing, with several alternative pedagogical choices, including simulation-based laboratory instruction, direct digital conversion to a radio frequency carrier, and commercially available radio frequency platforms. Our preference for an acoustic modem derives from four goals. First, we seek a very low-cost pathway to provide laboratory experiences to students using ubiquitous equipment. An emphasis on cost is heightened by consideration of resource challenges encountered world-wide and by Ohio State collaborations[1] in Honduras and Colombia. Second, in order to provide meaningful student experiences in a short, seven-week format, we seek a balance between investing time in tools versus concepts. We choose to tilt the balance in favor of concepts, adopting familiar software tools for a rapid learning curve. Third, we seek to require students to engage in systems-level consideration of the full modem architecture, rather than week-by-week operate in a black-box world of single substeps. We are hopeful that we have achieved a proper balance in this regard. Fourth, we seek to develop instructional materials that are somewhat independent of specialized hardware platforms which

[1] See https://osuhe.engineering.osu.edu/.

inevitably experience rapid successions of updates or replacements.

At The Ohio State University, this text accompanies a laboratory course, ECE 5007, which meets three hours weekly during the second half of each semester. The laboratory course was introduced in conjunction with our university's transition from quarters to semesters. Undergraduate students enter the laboratory course with a semester-long introduction to signals and systems and co-requisite enrollment in a digital communication lecture course. The laboratory course uses Chapters 1 through 7 as guided learning exercises, reinforcing and exploring concepts abstractly presented in a typical digital communication lecture course. The second term schedule facilitates coordination of the laboratory topics with a lecture syllabus based on any of the many excellent digital communication textbooks, e.g., [10, 22, 31, 27, 18, 13]. The scope of each chapter has been purposefully selected with the aim that students successfully implement a portion of an acoustic digital modem during a three hour session.

The text supports other learning experiences beyond the Ohio State course offering. First, the chapters are written with the intent of allowing self-study. A brief review of concepts is presented in each chapter, with emphasis on intuition and algorithm implementation; more detailed and rigorous treatments of topics can be found in referenced sources. Second, the approach can be adopted for use in high school STEM education. Pilot high school projects have been taught in Annandale-on-Hudson, New York and Seattle, Washington by Taylor Williams, a Knowles Science Teaching Foundation Fellow. Third, additional chapters are included in the text to accommodate a full semester laboratory course; additional topics include forward error correction coding, equalization of frequency-selective fading channels, multi-carrier modulation, and adaptive processing. Further, the acoustic signal can provide an intermediate frequency signal for radio frequency modulation; several radio frequency hardware options are discussed in Appendix B.

The guided explorations challenge students to design and implement a digital modem operating at audio frequencies. The computations for the software-defined modem are performed in a high-level language, such as MATLAB; the exercises presented in the text have been run in MATLAB .

The basis for our acoustic modem approach to a digital communication laboratory comes from a senior design project conducted some years ago at Ohio State. Undergraduate teams were judged on communication system performance measured as bits per second per Hertz per dollar per decameter; transmission was at 918 MHz, an acceptable error rate was specified, and a laptop or smartphone was considered a zero-cost option for baseband processing. The open-ended design project elicited solutions ranging from high-gain directional antennas to merely radiating with paper clips, and students typically chose PSK or OFDM designs. The combination of systems-level design and detailed subroutine development was judged a strength of the experience.

The authors thankfully acknowledge the insights and contributions we have gathered from a variety of sources. The approach is heavily influenced by the digital communication textbook by Rick Johnson and Bill Setheras [18] and course notes by Phil Schniter [31]. We have benefited from the laboratory courses taught by Steven Tretter [37] and Robert Heath [15]. We are grateful to the teaching assistants and students who embraced the senior design challenge of building a 918 MHz modem; particular thanks go to Adam Margetts, Aditi Kothiyal, Gene Whipps, Marshall Haker, and Aaron Wise. We have learned much from Rohit Aggarwal's development of exercises for the National Instruments RF-RIO board. We thank Abigail Gillett, Adam Rich, and Philip Chen for development of mobile apps for acoustic transmission. Finally, we are grateful to The Mathworks, Inc. for hosting course materials at MATLAB Courseware.

This text and associated course materials are freely available in electronic format at Mathworks MATLAB Courseware, iTunesU, and OpenStax CNX. Additionally, this text is available in hardcopy from Lulu Press. An acoustic transmitter is available as a free mobile app at the iTunes App Store and Google Play.

Lee C. Potter
Riverlea, Ohio

Yang Yang
San Diego, California

CHAPTER 1

Introduction

The goals of this introductory exercise are to review basic concepts from signals and systems and to introduce the hardware and software tools to be used during the course. Concepts reviewed include sampling, aliasing, sinc reconstruction, low-pass filtering, frequency upconversion and frequency downconversion. The chapter begins with a survey of physical layer communication and discussion of the software-defined radio concept.

1.1 A Physical Layer Model

Figure 1.1 displays a digital communication processing chain for both the transmitter and receiver. In the seven layer Open Systems Interconnection model, the *physical layer* (PHY), or Layer 1, defines the means of transmitting raw bits over a physical link. For digital communication, the chain begins and ends with a binary message; this binary message may represent, for example, speech, audio, an image, video, or high-definition television. At the transmitter, bits are encoded for error protection and converted to a sequence of discrete symbols. Symbols must be converted to a waveform suitable for transmission. In a process known as "pulse shaping," symbols are converted to a sampled-data baseband message; then, the sampled data are converted to an analog voltage signal through a digital-to-analog converter. This conversion often also entails a frequency shift to an intermediate frequency. The intermediate frequency signal is then modulated to a frequency that is matched to the desired transmission channel, such as a licensed band for commercial radio frequency transmission.

The receiver processing chain mirrors the transmission chain, but with added complexity to handle synchronization of the receiver to the transmitter and to accommodate signal distortions, or impairments, due to transmission or resulting from thermal noise at the receiver. Receiver design to mitigate these impairments is considered in Chapter 5 and Chapter 6; additional techniques to overcome channel impairments are presented in Chapters 8 through 12.

Symbols are illustrated in Figure 1.2(a) for the case of the quadrature phase shift keying (QPSK) symbol constellation, also known as 4-ary quadrature amplitude modulation (4-QAM). In QPSK, pairs of bits are mapped to one of four equal energy symbols. The mapping of bits to symbols and the decoding of noisy symbols back to bits are considered in Chapter 3. A typical impulse response used for pulse shaping is illustrated in Figure 1.2(b). The role of the pulse shaping filter and the associated design trade-offs are considered in Chapter 4. Moving down the transmission chain in Figure 1.1, the in-phase channel of a 47-bit baseband message is shown in Figure 1.2(c), and the corresponding bandpass signal with acoustic carrier frequency 1800 Hz is shown in Figure 1.2(d).

In this laboratory course, students are guided through a sequence of directed explorations, culminating in the design and implementation of an acoustic modulator/demodulator (modem). The acoustic modem implemented in this course works much like a fax modem.

1.2 Software Radio

A radio is any kind of device that wirelessly transmits or receives signals at the radio frequency (RF) wavelengths of the electromagnetic spectrum. Radios are found in many products, such as cell phones, tablet computers, automobiles, televisions, spectrometers, and magnetic resonance imagers. A *software-defined radio* (SDR) can be defined as a radio in which some or all of the physical layer functions are implemented through modifiable software operating on programmable processing technologies [32, 12].

Conventional hardware-based radio devices provide operational flexibility only by physically changing hardware components. In contrast, a SDR provides an inexpensive multi-mode, multi-functional, multi-protocol device that can be modified using software upgrades

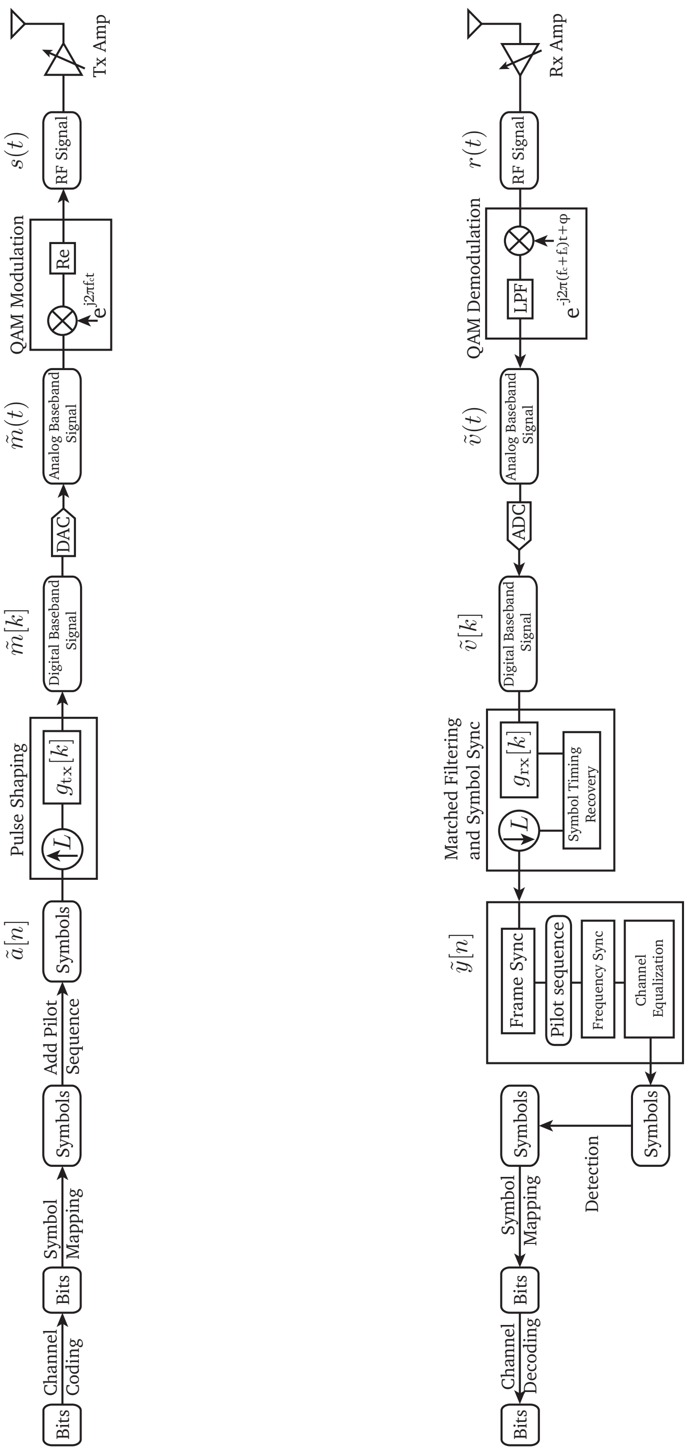


Figure 1.1: Physical layer communication processing chain for transmission and reception.

to provide new features to existing systems. The steady advancement of digital technologies has provided digital sampling rates and processing speeds to process both baseband and intermediate frequency signals. SDR may soon provide interoperability of communication devices in public safety and emergency response scenarios.

A typical SDR architecture is shown in Figure 1.3. For operation at frequencies above the UHF band, current SDRs use mixers at the front end to perform analog upconversion and downconversion to and from the desired frequency band. On receive, the wireless signal induces on the antenna a current, which is boosted by a low-noise am-

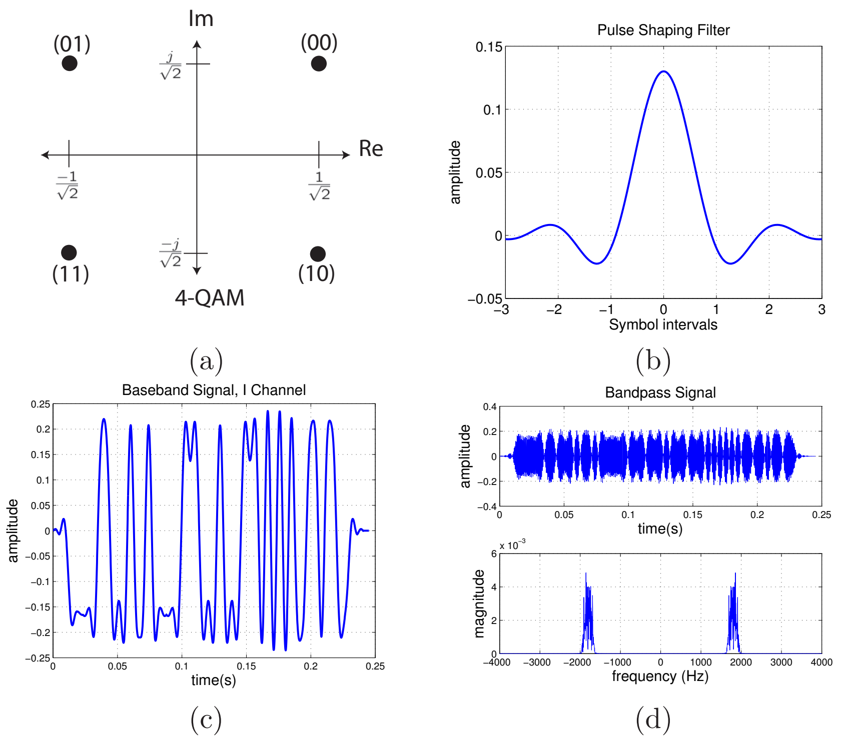


Figure 1.2: Illustrations of (a) an example symbol constellation; (b) pulse shaping filter impulse response; (c) baseband signal, I channel; and, (d) bandpass signal for $f_c = 1800\,\text{Hz}$.

plifier and filtered before downconversion to in-phase (I) and quadrature (Q) channels at an intermediate frequency (IF). The IF signal is then directly sampled by an analog-to-digital converter (ADC). Digital downconverters (DDCs) then further downconvert the signal via multiplication by a signal generated by a numerically controlled oscillator (NCO). Multiplication results in signals at the sum and difference of frequencies. The sum-of-frequencies signal is filtered out, and the difference-of-frequencies signal provides the baseband waveform, which is decimated by a factor of L, meaning only every L^{th} sample is retained. The decimated signal lowers the sample rate to the baseband bandwidth, simplifying data processing yet retaining the information carried in the signal.

A similar set of steps occurs on transmission. Digital upconversion (DUC) numerically mixes a baseband signal to an IF carrier frequency. The DUC, like the DDC at the receiver, is commonly packaged as a single integrated circuit. The IF signal signals in the I and Q channels feed an analog quadrature mixer that produces the RF signal, which is amplified and fed to the antenna.

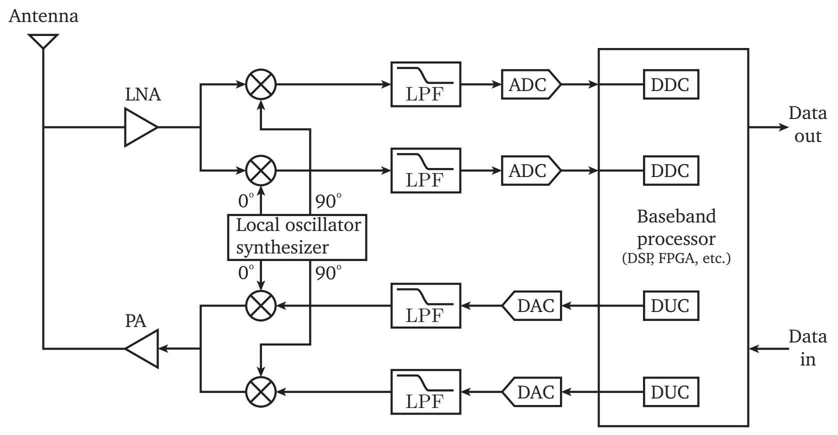


Figure 1.3: A typical SDR architecture.

1.3 Background

This section provides a brief review of concepts explored in the laboratory exercises. First, sampling and aliasing are reviewed. Second, upconversion and downconversion are defined for the real-valued signal case. Complex-valued sampling and quadrature modulation are explored in Chapter 2.

1.3.1 Sampling and aliasing

The basic intuition of sampling and aliasing is evident in Figure 1.4: given only samples of a sinusoidal waveform, there exist tones of infinitely many possible frequencies that pass through the samples. The frequencies of these possible interpolating waveforms differ by a multiple of the sampling frequency. In the figure, the solid line shows $m(t) = \sin(2\pi 20t)$, a sine wave at 20 Hz. Samples, shown by dots-on-sticks, are taken every 0.01 s, for a *sampling rate* of $f_s = 100$ samples per second (sps). The top panel shows that $\sin(2\pi(20 + f_s)t) = \sin(2\pi 120t)$ also is consistent with the same set of samples; likewise, the bottom figure illustrates that $\sin(2\pi(20-f_s)t) = \sin(2\pi(-80)t) = \sin(2\pi 80t + \pi)$ is another tone yielding the same samples.

The Fourier spectrum, $X(f)$, of the sampled signal contains the sum of these possibilities. Let $x(t) = m(t) * \sum_n \delta(t - nT)$ be the sampled signal; then,

$$X(f) = \frac{1}{T} \sum_{k=-\infty}^{\infty} M(f - kf_s), \tag{1.1}$$

where $M(f)$ is the spectrum of the original analog signal, $m(t)$. Each shifted replica of the spectrum $M(f)$ in Equation 1.1 is called an *image*. To eliminate confusion among the possible signals yielding a sampled waveform, the *Nyquist criterion* stipulates that $M(f) = 0$ for all $|f| \geq f_s/2$; that is, the Nyquist criterion stipulates that the sampling rate yield at least two samples per cycle for any frequency present in $m(t)$.

More generally, *bandpass sampling* of a real-valued signal $m(t)$ requires that the two-sided bandwidth of the spectrum $M(f)$ be confined to a single Nyquist zone,

$$\frac{(k-1)f_s}{2} \leq |f| \leq \frac{kf_s}{2}, \qquad \text{for some positive integer } k. \tag{1.2}$$

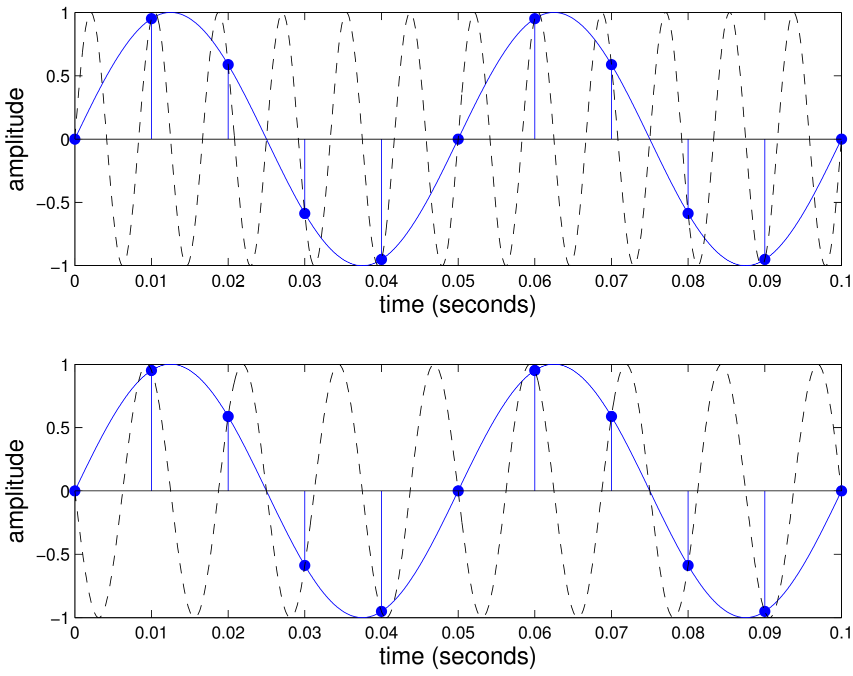


Figure 1.4: Illustration of sampling and aliasing.

In this manner, an alias of $M(f)$ appears with frequency support confined to $f \in [-f_s/2, f_s/2)$.

1.3.2 Frequency upconversion

Upconversion (amplitude modulation or mixing) of a real-valued message $m(t)$ is given by

$$s(t) = m(t)\cos(2\pi f_c t) \tag{1.3}$$

and serves to translate the message to a desired frequency band centered on the carrier frequency, f_c. Upconversion is graphically depicted in Figure 1.5. From the Euler identity, $e^{jx} = \cos x + j\sin x$; therefore,

$$\cos(2\pi f_c t) = \frac{1}{2}\left\{e^{j2\pi f_c t} + e^{-j2\pi f_c t}\right\}. \tag{1.4}$$

The spectrum, $S(f)$, is found via the Fourier transform

$$\begin{aligned} S(f) &= \int_{-\infty}^{\infty} m(t)\cos(2\pi f_c t)e^{-j2\pi f t}dt \\ &= \frac{1}{2}\int_{-\infty}^{\infty} m(t)e^{-j2\pi(f-f_c)t}dt + \frac{1}{2}\int_{-\infty}^{\infty} m(t)e^{-j2\pi(f+f_c)t}dt \\ &= \frac{1}{2}M(f-f_c) + \frac{1}{2}M(f+f_c). \qquad (1.5) \end{aligned}$$

Because $m(t)$ is real-valued, $M(f)$ is conjugate symmetric about $f = 0$, implying that the upconverted spectrum for $f > 0$ is conjugate symmetric about f_c and that the spectrum lower sideband (below f_c) is redundant with the upper sideband (above f_c). The magnitude spectrum is illustrated in Figure 1.6.

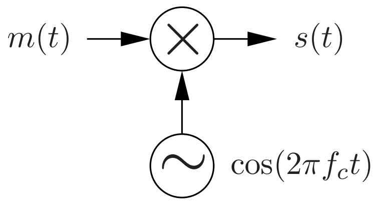


Figure 1.5: Amplitude modulation performs a translation in frequency.

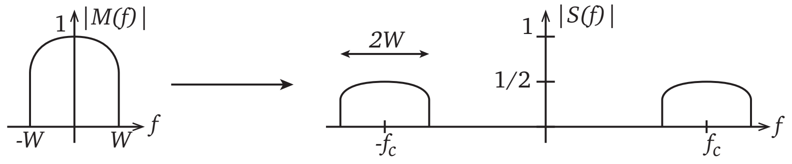


Figure 1.6: Magnitude spectrum resulting from amplitude modulation.

1.3.3 Frequency downconversion

With f_c known, downconversion, or amplitude demodulation, can be accomplished by mixing and low-pass filtering

$$\begin{aligned} v(t) &= \mathrm{LPF}\{s(t) \times 2\cos(2\pi f_c t)\} \\ &= \mathrm{LPF}\{m(t)\underbrace{2\cos^2(2\pi f_c t)}_{1+\cos(4\pi f_c t)}\} \\ &= \mathrm{LPF}\left\{m(t) + m(t)\cos\left(2\pi\{2f_c\}t\right)\right\} \\ &= m(t) \end{aligned} \tag{1.6}$$

This process in depicted in Figure 1.7 and Figure 1.8.

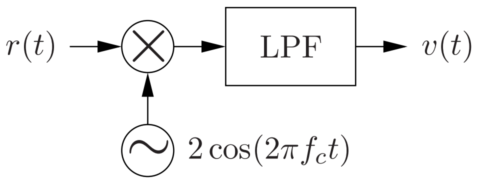


Figure 1.7: Amplitude demodulation.

In Equation 1.6 and Figure 1.7, the low-pass filter (LPF) has a passband cutoff $B_p \geq W$ Hz and stopband cutoff $B_s \leq (2f_c - W)$ Hz, where W is the one-sided bandwidth of the baseband signal, $m(t)$. The filter's magnitude response is illustrated in Figure 1.8.

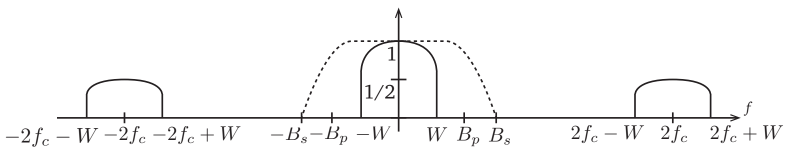


Figure 1.8: Low-pass filter (dashed line) used in amplitude demodulation.

When the receiver oscillator has frequency offset, f_Δ, or phase offset, ϕ, relative to the upconversion, the demodulated signal is distorted:

$$\begin{aligned} v(t) &= \text{LPF}\{m(t)\underbrace{\cos(2\pi f_c t)\times 2\cos(2\pi(f_c+f_\Delta)t+\phi)}_{\cos(2\pi f_\Delta t+\phi)\ +\ \cos(2\pi(2f_c+f_\Delta)t+\phi)}\} \\ &= m(t)\underbrace{\cos(2\pi f_\Delta t+\phi)}_{\text{time-varying attenuation}} \quad . \end{aligned} \tag{1.7}$$

The time-varying attenuation can be viewed as an upconversion of $m(t)$ to a frequency equal to the offset, f_Δ. Estimation and correction of this offset will be considered in Chapter 2 and Chapter 5.

Interpolation

Interpolation is a digital filtering procedure that increases the sampling rate by an integer factor, L:

$$y[k] = \sum_{n=-\infty}^{\infty} x[n]\frac{\sin(\pi(k-nL)/L)}{\pi(k-nL)/L}. \tag{1.8}$$

The processing to produce the interpolated samples can be viewed as two steps: upsampling (i.e., interlacing $L-1$ zeros between consecutive samples of $x[n]$) followed by low-pass filtering with gain L and cutoff frequency $\frac{\pi}{L}$ radians per sample. See Figure 1.9.

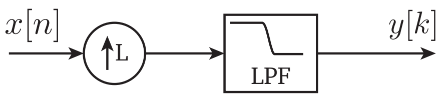


Figure 1.9: Block diagram for ideal interpolation.

DAC: sinc interpolation

The ideal digital-to-analog conversion (DAC) is implemented by sinc interpolation, which is illustrated in Figure 1.10:

$$x(t) = \sum_{k=-\infty}^{\infty} x[k]\frac{\sin(\pi(t-kT)/T)}{\pi(t-kT)/T}, \tag{1.9}$$

where T is the sampling period for $x[k]$. The summation is the convolution of a low-pass filter impulse response (i.e., a scaled sinc function) with the impulse train of samples, $x[k]\delta(t-kT)$. This summation is depicted in the figure, where the solid circles denote the sample values $\{-2, 2, 4, 4, 2\}$, the dashed lines denote each shifted and scaled sinc function, as in Equation 1.9, and the summation is shown by the solid line. The zero crossings of the sinc function, which occur every T seconds, ensure the faithful interpolation of the original samples.

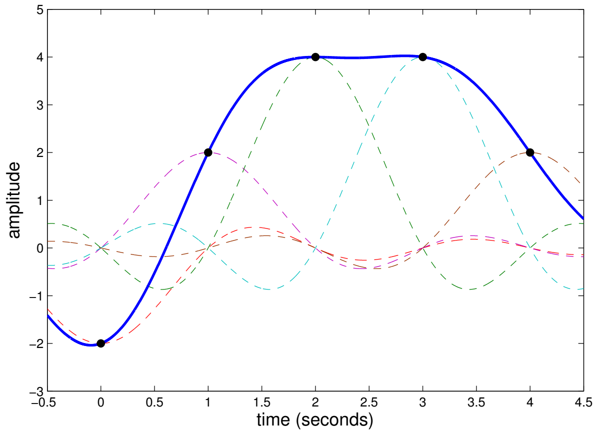


Figure 1.10: Illustration of sinc interpolation.

Decimation

Decimation is a digital filtering procedure that decreases the sampling rate. Given an original sampling rate of f_s sps, decimation by a positive integer L produces a new sampling rate of f_s/L. The signal is first low-pass filtered, typically in stages, with a cutoff frequency of $\frac{f_s}{2L}$ ($\frac{\pi}{L}$ radians per sample). The filtering prevents aliasing from resulting when the signal is downsampled. Only every Lth sample of the filtered output is computed and retained. Refer to Figure 1.11. Note that the low-pass filtering is an averaging that can reduce noise, particularly noise due to quantization error.

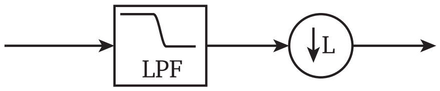


Figure 1.11: Block diagram for decimation by L; the anti-aliasing low-pass filter has cutoff frequency π/L radians per sample.

1.4 Explorations

Here, as in each chapter, a guided set of explorations is presented. Useful commands are introduced and used in example snippets of code to reinforce and visualize key concepts. Chapter by chapter, students build towards a fully functional acoustic modem.

Each chapter is designed for a single laboratory session with a brief report and a demonstration. In subsequent chapters, exercises are increasingly less scripted, with freedom for creative detours and experimentation. Students are encouraged to be brief and direct in reporting; to this end, questions to be answered in the report are enumerated in the margins. The recommended demonstration is detailed in a separate subsection.

Microphone input

For Matlab to recognize a sound recording device in some Windows configurations, an external microphone must be inserted into the microphone jack of the PC tower *before* starting Matlab. Right-click on the volume controls in the system tray (a speaker icon typically located at the bottom right of the Windows screen), select “Audio Devices” and select the “Recording” tab. Ensure that there is some reaction from the microphone as shown by the volume bars beside the microphone entry. If there is no reaction, then go to the properties of the microphone (double click) and adjust the microphone volume and/or boost until an input reaction is observed. More generally, from right-click on the speaker icon, you can select playback devices, recording devices, and volume control options. From these command windows, you can select a playback device and a recording device. By clicking on the icon for a device, you can enable the device, check device properties, and set sound levels.

Workflow suggestions

For numerical and graphical explorations, several practices may be helpful. First, students are advised to create a memory folder for laboratory work; download files and select the folder using the "browse for folder" icon in the MATLAB tool bar. For example, the explorations in Chapter 1 use the function `plottf.m`, which is given in Appendix C. A useful practice is to create a new script file (available at the top left of the menu bar) for each chapter. An editor window can then be used to copy and execute code snippets, write new commands, or edit and re-run codes. For each chapter, the code snippets appearing in the chapter are available as a stand-alone `.m` script, such as `Chapter1snippets.m`. All scripts and functions are available for download at OpenStax CNX and Mathworks MATLAB Courseware.

Students are encouraged to embrace the development of engineering skills, and software design skills in particular, that are a part of this laboratory experience. To this end, we highlight several suggestions in Appendix A. Subsections of code developed in each laboratory section will eventually be integrated into a single working modem; thus, purposeful attention to a few simple practices will greatly simplify work as the lessons progress.

1.4.1 Getting started with audio I/O

In this experiment, you will familiarize yourself with recording and playing audio signals.

Recording

Four commands allow you to access the sound card hardware from within MATLAB to record audio signals.

`>> recObj = audiorecorder(Fs,Nbits,nChannels)` sets the sample rate `Fs` (in samples per second), the number of bits per sample `Nbits`, and the number of channels `nChannels`. The defaults are `Fs`=8000, `Nbits`=8, and `nChannels`=1.
`>> recordblocking(recObj,length)` records sound for `length` seconds and does not return control until recording completes.
`>> getaudiodata(recObj)` creates an array to store the recorded signal values.
`>> get(recObj)` queries the properties of the audiorecorder object `recObj`.

Valid values of the sampling rate depend on the specific audio hardware installed. Typical values supported by most sound cards are 8000, 11025, 22050, 44100, 48000, and 96000 samples per second. The code snippet in Table 1.1 provides an example. Typing your own code as you do example exercises will help in building and retaining familiarity with simple commands. In addition, code snippets are available for download; see links given in Appendix C. If, in executing the function `audiorecorder`, you encounter the error message `no audio device found`, then exit MATLAB, unplug and re-plug the microphone into the PC input jack, and re-start MATLAB.

Playing audio

Similarly, two commands allow a user to play audio through a sound card from within MATLAB.

`>> playObj = audioplayer(Y,Fs,Nbits)` creates an audioplayer object for the signal `Y`, using sample rate `Fs` and `Nbits` bits per sample. The function returns a handle to the audioplayer object, `playObj`. The audio signal is represented by a vector (mono) or two-dimensional array (for stereo). The default is `Nbits`=16.
`>> play(playObj)` plays audio from beginning to end.

The following code snippet provides two examples.

Suggested explorations:

(a) Follow the example in Table 1.1 to record and plot 2 seconds of speech.

(b) Follow the examples in Table 1.2 to play a tone and the recorded speech.

1.4.2 Spectral content of signals

Visualization can be a powerful tool for understanding signals and the effects of systems on signals. Here we consider use of the routine `plottf.m` to view a sampled signal in both the time and frequency domains. The custom routine is given in Appendix C.

`>> plottf(x,Ts)` plots the time-domain samples in vector `x`, assuming that `Ts` is the sampling interval in seconds, and also plots the Riemann-sum approximation of the Fourier transform between the frequencies of -1/(2`Ts`) and 1/(2`Ts`) Hertz.

`>> plottf(x,Ts,'t')` plots only the time-domain signal.

`>> plottf(x,Ts,'f')` plots only the frequency-domain signal.

In all cases, `plottf` returns handles to the graphical objects; as with any MATLAB command, display to the command window is suppressed by a semi-colon.

Suggested explorations:

(c) Use `plottf` to view the speech signal recorded in Experiment 1.4.1 and stored as `myRecording`. Note that `plottf` asks for the sampling interval, which is the reciprocal of the sampling rate. To start a new figure window, rather than overwrite the most recently accessed figure, use the `figure` command. The error statement "Undefined function 'plottf'" indicates that the routine `plottf.m` is neither in the current working directory nor specified in the current search path (see `help path`).

(d) Use the zoom tool in the figure to look more closely at speech in the time domain. What qualitative differences do you observe between vowels and consonants? Is a DC bias present in your recorded speech signal? **Q1.1**

(e) Look more closely at speech in the frequency domain. What are the lowest and highest frequencies displayed in the graph, and why? What band of frequencies is non-negligibly occupied by the speech signal? How would a DC bias be evident in the frequency domain? (Hint: use the zoom option with the frequency domain plot to observe $X(f)$ for $f = 0$.) **Q1.2**

(f) With sampling rate fixed at `Fs`=16000 sps, plot 40 milliseconds of the sinusoidal signal, $\cos(2\pi f_c t)$, for frequencies 1000, 7000, 9000, 16000 and 17000. For an example, refer to Table 1.2. View the signal in both the time and frequency domains using `plottf`. What do you observe, and why?

Q1.3 (g) With sampling rate fixed at `Fs`=8000 sps, specify four values of `f0`, $0 <$ `f0` < 16000, so that `cos(2*pi*f0*(0:Fs/2)/Fs)` sounds like one-half second of a 440 Hz tone.

(h) Experiment with the `audioplayer` command to discover what sampling rates are supported by your sound card hardware; an unsupported rate will return an error message when the `play` command is invoked.

1.4.3 Low-pass filtering

For design of a low-pass filter, the custom function `firlpf.m` is provided in Appendix C. The command-line function requires the syntax

```
h = firlpf(Lh,Fpb,Fsb,Fs);
```

where `h` is the impulse response of length `Lh`, `Fpb` is the passband edge (in Hertz), `Fsb` is the stopband edge (in Hertz), and `Fs` is the sampling frequency (in samples per second). The routine computes the least-squares fit to the ideal magnitude response illustrated in Figure 1.12. The frequency-selective filtering using the impulse response `h` can be implemented via the convolution operation by the command `conv` or `filter`.

(i) Create a low-pass noise signal to experiment with frequency translation. The following code snippet illustrates use of a Gaussian random number generator and a low-pass filter.

```
Fs=16000;% 16000 sps
t=0:1/Fs:0.25;% 250 milliseconds segment
whitenoise=randn(size(t));% Gaussian random vbls
figure;plottf(whitenoise,1/Fs);% view signal
title('White Noise')
%filter design; see custom firlpf.m in Appendix
h = firlpf(49,1200,1800,Fs);
pinknoise=filter(h,1,whitenoise);% convolution
figure;plottf(pinknoise,1/Fs);% view signal
title('Bandlimited Noise')
```

(j) View both your `whitenoise` and `pinknoise` signals in the time and frequency domains. Comparing the two signals, what do you observe in each domain?

(k) **[Optional]** Repeat step (j) for several smaller values of filter length small than the `Lh=49` used above. What do you observe?

(l) **[Optional]** Add a tone at 2100 Hz to your `whitenoise signal`. Observe the resulting signal before and after the low-pass filter.

1.4.4 Amplitude modulation

In this experiment, you will explore amplitude modulation illustrated in Figure 1.5 and gain familiarity with several basic instructions.

(m) Modulate your signal from step (i) to a carrier frequency $f_c = 4\,\text{kHz}$ and view in time and frequency. Note that to multiply sampled signals *point-by-point*, the MATLAB syntax ".*" should be used. From step (i) the sampling rate is 16000 sps.

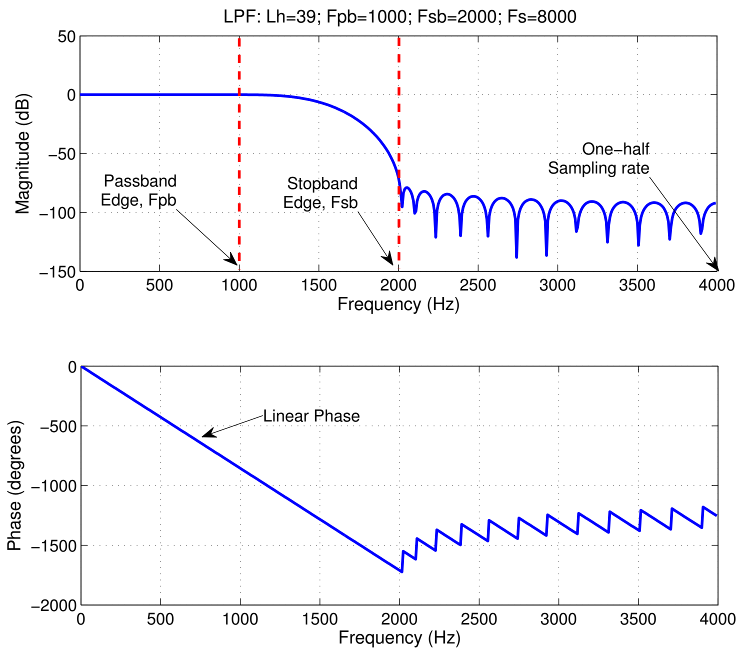


Figure 1.12: Example of FIR low-pass filter design using `firlpf`.

```
fc=4000;%4000 Hz
s = cos(2*pi*fc*t) .* pinknoise;
figure;plottf(s,1/Fs);
title('Modulated Bandpass Noise')
```

Q1.4 (n) Keep `Fs=16000`. Experiment with various values of the carrier frequency, `fc`, from the previous step. What do you observe for carrier frequencies $6,500 \leq \texttt{fc} \leq 16,000\,\text{Hz}$?

Q1.5 (o) For the bandlimited noise `pinknoise` above, suppose you wish to modulate to a carrier frequency of $19,000\,\text{Hz}$. Suggest an appropriate sampling rate.

1.4.5 AM demodulation

For frequency downconversion, follow the diagram in Figure 1.7. Recall $2\cos(\alpha) \times \cos(\beta) = \cos(\alpha - \beta) + \cos(\alpha + \beta)$. The role of the low-pass filter is to remove this second term, the so-called "double frequency" term.

(p) Demodulate the signal `s` constructed in step (m) above. View the demodulated signal in both the time and frequency domains. For the low-pass filter, select a filter length `Lh=49`. What are appropriate values for the passband and stopband edge frequencies, `Fpb` and `Fsb` (refer to Figure 1.8)?

(q) Omit the filter convolution step in the demodulator, and again view the result. What do you observe?

(r) [**Optional**] For demonstration to the instructor, implement the following amplitude modulation and demodulation steps. Use `plottf` to view each step. The aim is to reinforce the principles and techniques explored above.

- Record 900 milliseconds of speech at a sampling rate of your choice. (Hint: read all steps below before making your choice).
- Amplitude modulate your recorded speech signal using a carrier frequency of $f_c = 12,000\,\text{Hz}$. Use `play` to listen to the modulated speech signal.

- Open simultaneously a second MATLAB command window. Use one window to play the modulated speech signal, and the other to record the audio transmission.
- Demodulate the received AM signal and play the recovered audio.

1.5 Demonstration

For demonstration, use the Communication Laboratory mobile app to transmit an acoustic signal using the app's default settings. The app is freely available at iTunes and Google Play; a description of the app is found in Section 7.1.1.

Record the transmission with sampling frequency `Fs=44100`. View the received signal using the `plotttf.m` function. Observe and report the following parameters from your graph.

1. What is the approximate time duration of the transmission, in milliseconds?
2. What, approximately, is the carrier frequency?
3. What, approximately, is the two-sided bandwidth of this band-pass signal?

1.6 Summary

In Chapter 1, basic signals and systems concepts have been reviewed: sampling, aliasing, interpolation, low-pass filtering, frequency upconversion and frequency downconversion. The exercises introduced the recording and playing of audio-rate signals. And, the explorations served to introduce the commands and conventions summarized in Table 1.3.

Table 1.1: Code snippet for audio recording

```
%% Create an audiorecorder object with 8000 sps and
%    a single channel (mono); view its properties
Fs=8000;
recObj = audiorecorder(Fs, 16, 1);
get(recObj)

%% Collect a sample of your speech with a microphone;
%    and, plot the signal data

% Record your voice for 2 seconds. Use display and
% pause as aids to control the start of the recording.
Trec = 2; %2 second record time
disp('Press Enter to start recording.')
pause;%wait for keystroke
disp('Recording.')
recordblocking(recObj, Trec);
disp('End of Recording.')

% Store data in double-precision array.
myRecording = getaudiodata(recObj);

%% Plot the waveform.
t = (0:length(myRecording)-1)/Fs; %sample times (sec)
plot(t,myRecording);
```

Table 1.2: Code snippet for audio play

```
%% Example:  play a one second tone at 440Hz
Fs=8000; % sampling rate of 8000 sps
t = 0:1/Fs:1; %list of sampling times
signal = cos(2*pi*440*t);

% Create audioplayer object
ToneObj = audioplayer(signal,Fs);

% Play the sound
play(ToneObj)

%% Example: play back the speech recorded previously
play(recObj);
```

Table 1.3: Summary of commands introduced in Chapter 1.

Command	Description
`audiorecorder`	create audio recording object
`audioplayer`	create audio player object
`recordblocking`	record sound
`getaudiodata`	store recorded sound samples
`play`	play an audio object
`get`	display object properties
`disp`	display text to command window
`pause`	pause execution awaiting key stroke
`plot`	make a line plot
`figure`	initiate new figure window
`title`	place title on figure
:	construct lists
;	suppress output
*	matrix multiply
.*	array multiply
/	division
`size`	returns size of a data array
`cos`	cosine function
`plottf`	convenient plotting routine
`randn`	Gaussian random number generator
`firlpf`	least-squares design of a low-pass filter
`conv`	convolution
%	remainder of line is a comment
%%	creates an executable subsection of code
`path`	get or set the search path
`help`	display help text in command window

CHAPTER 2

Quadrature Amplitude Modulation

Chapter 1 reviewed basic signals and systems concepts; the exercises also introduced the recording and playing of audio-rate signals. In this lesson, quadrature amplitude modulation (QAM) and coherent quadrature demodulation are introduced. Complex baseband is presented as a convenient mathematical notation for representing QAM signals. Here we build upon the introduction in the previous chapter to implement a QAM transmitter and receiver. The exercise also introduces the practical impairments of frequency and phase offsets, which are compensated in the receiver design to be implemented in Chapter 6.

2.1 Background

2.1.1 Quadrature amplitude modulation

In quadrature amplitude modulation two real-valued messages are modulated simultaneously, resulting in a bandpass spectrum that is not conjugate symmetric about the carrier frequency, f_c. The modulation exploits the orthogonality of the sine and cosine functions

$$\int_{-1/(2f_c)}^{1/(2f_c)} \sin(2\pi f_c t)\cos(2\pi f_c t)dt = 0. \tag{2.1}$$

The quadrature modulator is shown in Figure 2.1. The modulator has two “branches,” one for an *in-phase signal*, $m_{\mathrm{I}}(t)$, that modulates the cosine carrier and a second for a *quadrature signal*, $m_{\mathrm{Q}}(t)$, that modulates the sine carrier. The sine wave is 90 degrees ($2\pi/4$ radians) phase delayed from the cosine, giving rise to the name “quadrature.”

The two modulated waveforms, sometimes called "I" and "Q" for short, are combined to produce the modulated waveform, $s(t)$, given by

$$s(t) = m_\mathrm{I}(t)\cos(2\pi f_c t) - m_\mathrm{Q}(t)\sin(2\pi f_c t). \tag{2.2}$$

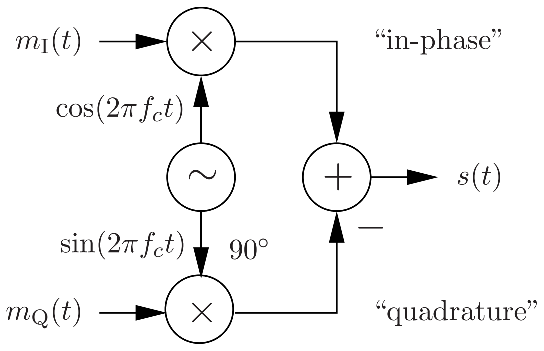


Figure 2.1: Quadrature amplitude modulator.

Ideal QAM demodulation is accomplished as shown in Figure 2.2. For baseband signals $\{m_\mathrm{I}(t), m_\mathrm{Q}(t)\}$ with one-sided bandwidth W Hz, the lowpass filter in the demodulator has passband edge frequency $B_p \geq W$ Hz and stopband edge frequency $B_s \leq (2f_c - W)$ Hz. (See Figure 1.8.)

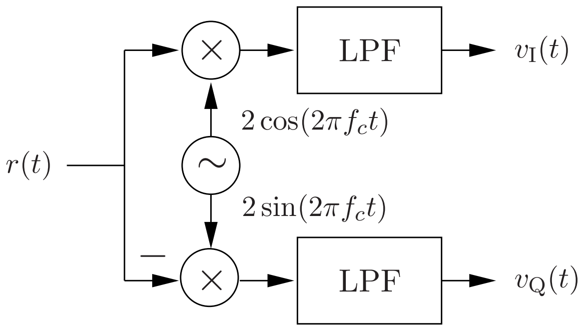


Figure 2.2: Quadrature amplitude demodulator.

Simple trigonometric identities can be used to verify that, for $r(t) = s(t)$ (i.e, a noiseless and distortionless channel), the message signals are perfectly recovered.

$$\begin{aligned}
v_{\mathrm{I}}(t) &= \mathrm{LPF}\{r(t) \times 2\cos(2\pi f_c t)\} \\
&= \mathrm{LPF}\{m_{\mathrm{I}}(t)\underbrace{2\cos^2(2\pi f_c t)}_{1+\cos(4\pi f_c t)} - m_{\mathrm{Q}}(t)\underbrace{2\sin(2\pi f_c t)\cos(2\pi f_c t)}_{\sin(4\pi f_c t)}\} \\
&= m_I(t) \qquad (2.3) \\
v_{\mathrm{Q}}(t) &= \mathrm{LPF}\{-r(t) \times 2\sin(2\pi f_c t)\} \\
&= \mathrm{LPF}\{-m_{\mathrm{I}}(t)\underbrace{2\cos(2\pi f_c t)\sin(2\pi f_c t)}_{\sin(4\pi f_c t)} + m_{\mathrm{Q}}(t)\underbrace{2\sin^2(2\pi f_c t)}_{1-\cos(4\pi f_c t)}\} \\
&= m_Q(t). \qquad (2.4)
\end{aligned}$$

Low-pass filtering removes the *double frequency terms*, $\cos(4\pi f_c t)$ and $\sin(4\pi f_c t)$. For the coherent receiver described by Equation 2.3 and Equation 2.4, we require that the receiver oscillator is perfectly synchronized to the transmitter oscillator: that is, the frequency, f_c, and phase of the sine and cosine terms at the demodulator must exactly match the frequency and phase at the modulator. When the oscillators are not synchronized, there results a coupling at the receiver between the I and Q components, as well as attenuation of each.

2.1.2 Complex-baseband representation

The use of a complex-baseband signal representation provides simplification in writing, analyzing, and programming quadrature amplitude modulation waveforms. In the complex-baseband form, the in-phase and quadrature message signals are conveniently packaged together as the real and imaginary components of a single, complex-valued signal,

$$\begin{aligned}
\tilde{m}(t) &= m_{\mathrm{I}}(t) + j m_{\mathrm{Q}}(t), \\
\tilde{v}(t) &= v_{\mathrm{I}}(t) + j v_{\mathrm{Q}}(t). \qquad (2.5)
\end{aligned}$$

The superscript tilde is used to denote a complex-valued signal. Employing the complex-baseband notation, the modulation and demodulation are very simply described, as seen in Figure 2.3.

Euler's identity is invoked to verify the complex-baseband model shown in Figure 2.3. For the modulator, observe

$$\begin{aligned}
&\mathrm{Re}\{\tilde{m}(t)e^{j2\pi f_c t}\} && (2.6)\\
&= \mathrm{Re}\{(m_\mathrm{I}(t)+jm_\mathrm{Q}(t))(\cos(2\pi f_c t)+j\sin(2\pi f_c t))\}\\
&= m_\mathrm{I}(t)\cos(2\pi f_c t) - m_\mathrm{Q}(t)\sin(2\pi f_c t) = s(t). && (2.7)
\end{aligned}$$

The demodulation structure is verified in a similar manner:

$$\begin{aligned}
\tilde{v}(t) &= \mathrm{LPF}\{s(t)\cdot 2e^{-j2\pi f_c t}\}\\
&= \mathrm{LPF}\{(m_\mathrm{I}(t)\cos(2\pi f_c t) - m_\mathrm{Q}(t)\sin(2\pi f_c t))\cdot 2e^{-j2\pi f_c t}\}\\
&= \mathrm{LPF}\{m_\mathrm{I}(t)(e^{j2\pi f_c t}+e^{-j2\pi f_c t})e^{-j2\pi f_c t}\\
&\quad - m_\mathrm{Q}(t)(je^{-j2\pi f_c t} - je^{j2\pi f_c t})e^{-j2\pi f_c t}\}\\
&= \mathrm{LPF}\{m_\mathrm{I}(t)(1+e^{-j4\pi f_c t}) - m_\mathrm{Q}(t)(je^{-j4\pi f_c t}-j)\}\\
&= m_\mathrm{I}(t)+jm_\mathrm{Q}(t) = \tilde{m}(t). \qquad (2.8)
\end{aligned}$$

Frequency-domain interpretations of the steps in the modulation and demodulation are given by the frequency spectra in Figure 2.3. Observe the asymmetry of the passband signals about the carrier, f_c, and the corresponding asymmetry of the baseband signal's magnitude spectrum. Thus, the upper and lower sidebands are not redundant, in contrast to AM. The Re operation at the modulator denotes taking the real part of the complex-valued signal and can be interpreted using the Fourier transform relation

$$\mathrm{Re}\{u(t)\} = \frac{1}{2}[u(t)+u^*(t)] \overset{\mathcal{F}}{\longleftrightarrow} \frac{1}{2}[U(f)+U^*(-f)]. \qquad (2.9)$$

The convenience of the complex-baseband representation results in widespread use of complex-valued signal notation for passband signals encountered, for example, in communication, radar, spectroscopy, and medical imaging.

2.1.3 Complex-baseband equivalent channel

Next, we augment the quadrature modulation model to include a channel described by an impulse response, $h(t)$, and additive noise, $w(t)$. This is depicted in the top panel of Figure 2.4. All signals present in the top panel are real-valued, consistent with the physical I and Q voltages present in a hardware implementation of QAM. The

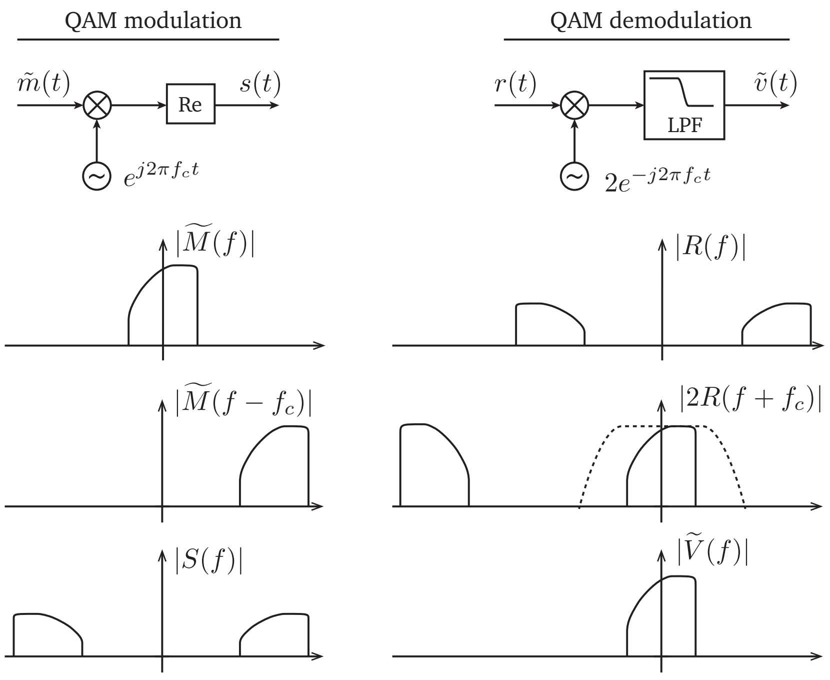


Figure 2.3: Quadrature amplitude modulation and demodulation are simplified using a complex-baseband notation. The steps are illustrated by the frequency spectra shown beneath the block diagrams.

behavior of $h(t)$ is of consequence only on the frequency band occupied by the modulated signal, $s(t)$. Thus, let $h_{bp}(t)$ be the bandpass equivalent channel

$$H_{bp}(f) = \begin{cases} H(f), & f_c - W \leq |f| \leq f_c + W \\ 0, & \text{else.} \end{cases} \tag{2.10}$$

This equivalence is depicted in the middle panel of Figure 2.4. With proper modulation and demodulation and commuting the low-pass and channel filters, we arrive to the bottom panel of Figure 2.4 which is known as the *complex-baseband equivalent model*.

The complex-baseband channel model is given by

$$\tilde{H}(f) = \begin{cases} H_{bp}(f + f_c), & -W \leq f \leq W \\ 0, & \text{else.} \end{cases} \tag{2.11}$$

This model simplifies representation, implementation, simulation, and analysis of the communication system.

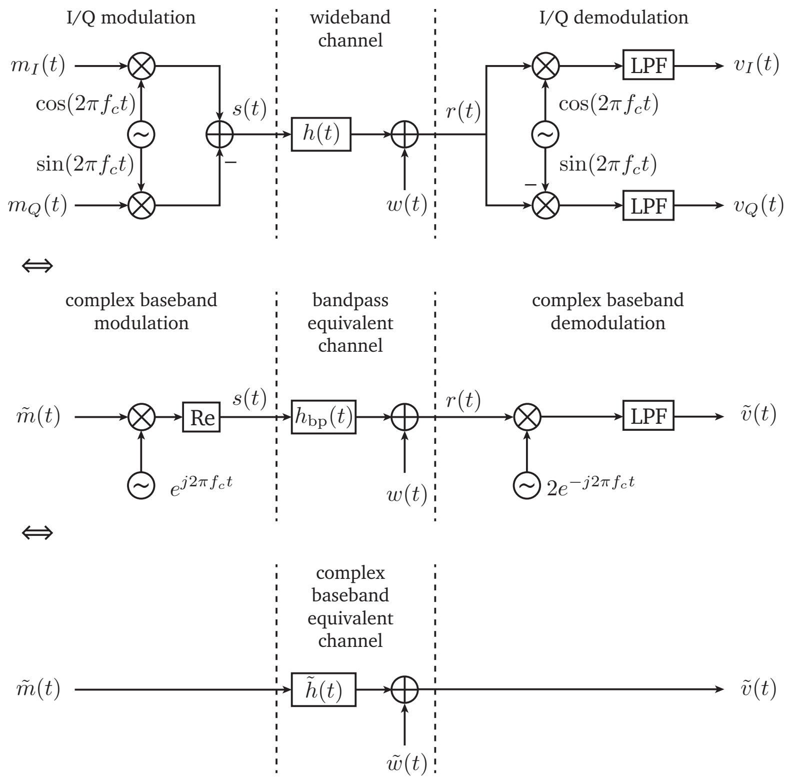


Figure 2.4: Complex-baseband equivalent channel model.

2.1.4 Coherent demodulation

For coherent demodulation, the local oscillator must have frequency and phase synchronized with the transmitter. The impairment is introduced here, and implementation of a frequency recovery algorithm is the topic of Chapter 6.

Consider a receiver oscillator given by $e^{j(2\pi(f_c+f_\Delta)t+\phi)}$ with a frequency offset, f_Δ Hz, and a phase offset, ϕ radians. Using the convenient complex-baseband representation of Figure 2.3 we have that the received message is

$$\tilde{v}(t) = \tilde{m}(t)e^{-j(2\pi f_\Delta t+\phi)}. \tag{2.12}$$

For sampled values indexed by k with sampling period T_s seconds per sample, we have

$$\tilde{v}[k] = \tilde{m}[k]e^{-j(2\pi f_\Delta T_s k+\phi)}. \tag{2.13}$$

Thus, if there is a frequency offset between transmitter and receiver, then a time-varying phase error results. Define the phase error as the true transmitted phase minus the received signal's phase; then, the phase error, $(2\pi f_\Delta t+\phi)$, at the receiver is a linear function of time, with slope proportional to the frequency offset. Figure 2.5 gives an example graph of this time-varying phase offset, which is due to both the phase mismatch (intercept) and frequency mismatch (slope).

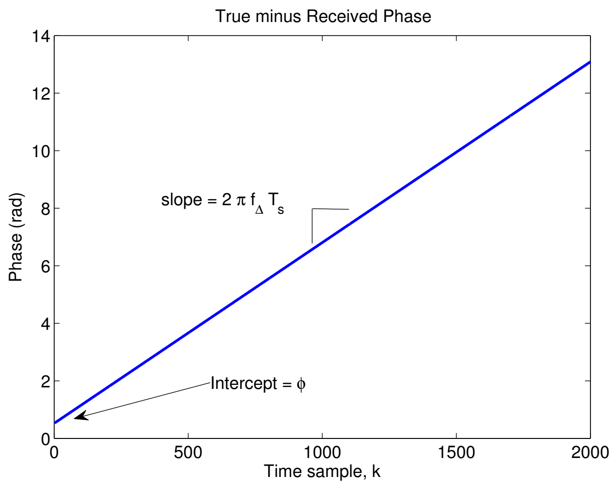


Figure 2.5: A time-varying phase error $\phi + 2\pi f_\Delta T_s k$, appears at the receiver output due to a frequency offset, f_Δ, and phase offset, ϕ.

2.1.5 Linear phase filters

The filters used at the transmitter and receiver will have linear phase, meaning the filter imparts the same time delay, in samples, to every frequency component of the discrete-time input signal. The linear phase is characterized in the filter impulse response as symmetry about the midpoint of the filter's impulse response. The delay in the output of a linear phase filter is one-half the filter order, and the order is one less than the length of the filter impulse response. Thus, in the notation of Section 1.4.3, the delay is `(Lh-1)/2` samples. When indexing into a filter output signal, this delay must be taken into account. For input `s` and impulse response `h`, the convolution output, `conv(s,h)` exhibits transients at the beginning and end of the output, due to zero boundary conditions used in the convolution sum. A communication packet signal is typically measured with noise, and the noise-only samples provide the boundary conditions.

Delay in a linear phase filter is illustrated in the following example. The noiseless signal is a truncated square-root raised cosine pulse, where the center of the pulse sits at index 100. The noise has variance 5×10^{-5}. The noisy signal is then low-pass filtered with cutoff of 1000 Hz. Here, the delay is 24 samples, which is one-half the filter order of 48.

```
Fs=8000;
k =0:200;
varw=0.00005;
s = sqrt(varw)*randn(size(k)); %create noise
s(51:151)=s(51:151)+srrc(2,0.7,25); %signal+noise
%LPF order 48, passband edge 1000 Hz
h = firlpf(49,1000,2000,Fs);
y = conv(s,h); %convolution
figure;plot(k,s);
xlabel('sample index','fontsize',13)
axis([0 250 -0.1 0.3]);
title('Filtering example','fontsize',13)
k2=0:length(y)-1;
hold on;plot(k2,y,'r--');hold off
xlabel('sample index','fontsize',13);
legend('input','output')
```

2.2 Digital Hardware [Optional]

In this section, optional background material is presented regarding digital hardware implementation of quadrature amplitude modula-

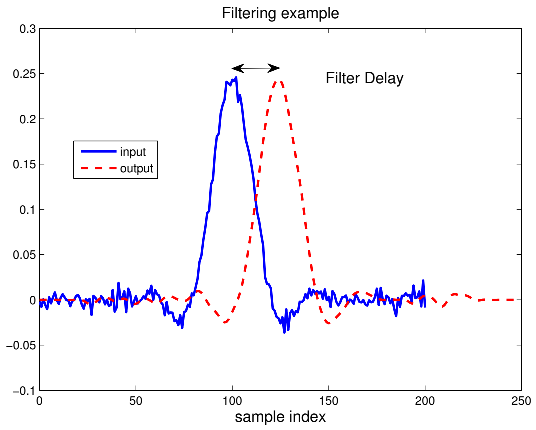


Figure 2.6: Illustration of delay imparted by a linear phase filter.

tion in radio frequency systems.

Advantages of digital implementation of upconversion and downconversion, versus analog mixers, are: (a) the ability to closely match the in-phase and quadrature branches in amplitude, frequency, and 90 degree phase offset; and, (b) the ability to avoid local oscillator leakage in the analog circuit. Thus, in many communication systems, digital processing is used to produce an intermediate frequency (IF) signal via quadrature modulation; the real-valued IF signal is then mixed and filtered, via analog hardware, to create the RF signal.

Digital upconversion

To illustrate digital implementation, consider, for example, the Analog Devices AD9857. The digital-to-analog converter (DAC) is combined with a digital upconverter (DUC) on a single integrated circuit. The AD9857 integrates a high-speed direct-digital synthesizer (DDS), a high-performance, high-speed 14-bit digital-to-analog converter (DAC), clock multiplier circuitry, digital filters, and other DSP functions onto a single chip, to form a complete quadrature digital upconverter device. The AD9857 is intended to function as a universal quadrature modulator and agile upconverter, single-tone DDS, or interpolating DAC for communication applications. The basic operation is abstracted in Figure 2.7.

A direct digital synthesizer (DDS) is the core of the AD9857 and is a numerically controlled oscillator (NCO). A phase accumulator provides a selectable modulus counter that increments with each clock pulse; the increment is determined by a digital word, and the value in the accumulator is used to index a sine-wave look-up table to produce accurate, stable, and phase-continuous frequency tuning. When used as a quadrature synthesizer, both cosine and sine outputs are produced, yielding excellent matching of in-phase (I) and quadrature (Q) outputs.

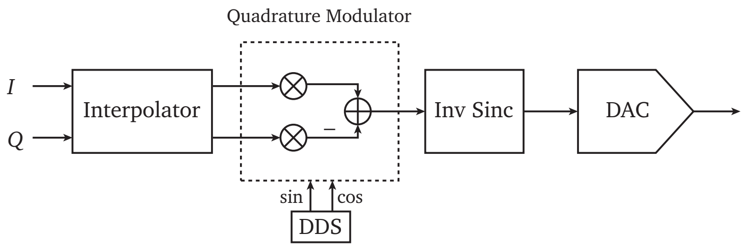


Figure 2.7: Basic operation of a quadrature digital upconverter.

Digital downconversion

The AD6654 is a mixed-signal IF-to-baseband receiver consisting of a 14-bit, 92.16M sps analog-to-digital converter (ADC) and a multichannel digital downconverter (DDC). It has been optimized for wideband standards such as CDMA2000, UMTS, and TD-SCDMA. The AD6654 is used as part of a radio system that digitally demodulates and filters IF sampled signals. The basic operation is abstracted in Figure 2.8.

2.3 Explorations

In this chapter, students explore quadrature modulation and the complex-baseband signal representation. The quadrature modulation and demodulation steps are illustrated on the right-hand side of Figure 1.1. For the laboratory exploration, work through the guided steps below using an acoustic carrier frequency. Recommendations for preparing a brief, descriptive laboratory report are included in the steps marked in the margins.

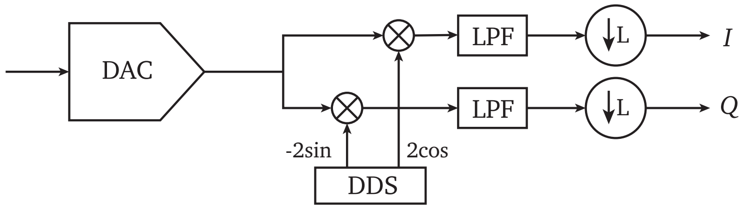


Figure 2.8: Basic operation of a quadrature digital downconverter.

Q2.1

(a) Verify Equation 2.12. Referring to Equation 2.8, derive the I and Q received signals, in terms of $m_I(t)$ and $m_Q(t)$, when the receiver operates with reference signal

$$2\exp\{-j(2\pi(f_c + f_\Delta)t + \phi)\}$$

instead of the ideal $2\exp\{-j2\pi f_c t\}$. Here, f_Δ is called the frequency offset, and ϕ is the phase offset. For reporting, compose a brief derivation relating $\tilde{v}(t)$ to $\tilde{m}(t)$.

(b) Create and plot a complex exponential signal $s(t) = \exp(j2\pi f t)$. Here is an example code snippet:

```
Fs=16000;%sampling rate, sps
t=0:1/Fs:0.005; %5 milliseconds
s = exp(1j*2*pi*1000*t);%1j is sqrt(-1)
figure; %starts a new figure window
plottf(s,1/Fs);
```

Notice the use of colons to construct a list. Given three numbers separated by colons, a uniformly sampled list is generated by starting at the first entry, incrementing by the middle entry, and stopping at the greatest number not exceeding the last entry. Note also that the semi-colon is used at the end of a command to suppress display of output to the command screen.

(c) Use the rotate tool from the figure menu bar to adjust the view of the time-domain plot of the complex-valued signal. What do you see? Likewise, explain the visual appearance of the frequency domain plot.

(d) Implement, in software, quadrature amplitude modulation and demodulation for sampled message signals, $m_I(t)$ and $m_Q(t)$. Use the following system parameters: sampling rate of 8000 sps; 200 time samples; and, carrier frequency of 2.1 kHz. For the low-pass filter, use length 35, passband edge 1 kHz, and stopband edge 2 kHz. For input signal $\tilde{m}(t) = m_I(t) + jm_Q(t)$, use the following values for $\tilde{m}(t)$:

- first 50 samples, $1/\sqrt{2} + j/\sqrt{2}$
- second 50 samples, $-1/\sqrt{2} + j/\sqrt{2}$
- third 50 samples, $-1/\sqrt{2} - j/\sqrt{2}$
- last 50 samples, $1/\sqrt{2} - j/\sqrt{2}$.

Hint: a constant list of 50 complex-valued samples with modulus 1 and angle 45 degrees can be generated as

```
(1/sqrt(2) + 1j/sqrt(2))*ones(1,50).
```

From the output of your quadrature amplitude demodulator, make three plots.

- time plot of received "I" signal
- time plot of received "Q" signal
- time plot of the phase of the received complex-baseband signal

Note that the phase of a complex number `v`, in degrees, can be obtained as `angle(v)*180/pi`. It is recommended that the I and Q plots be combined as subpanels within a single figure using `subplot`, for easy side-by-side comparison.

```
figure;%create new figure window
subplot(2,1,1) %2 rows, 1 column, first plot
plot(real(v));title('In-phase signal')
subplot(2,1,2)
plot(imag(v));title('Quadrature signal');
xlabel('sample number')
```

Q2.2 (e) View, using `plottf.m`, the demodulated signal in step (d) before and after applying the low-pass filter that removes the double-frequency terms. For reporting, present the graphs; briefly compare and explain the two frequency domain plots.

(f) For the received signal generated in step (d), create an IQ plot by plotting the quadrature signal (vertical axis) versus the in-phase signal (horizontal axis). Mark a red circle for every L^{th} point, as in the following code snippet for some signal, `v`, and $L = 10$. **Q2.3**

```
figure %create new figure window
plot(real(v),imag(v));
L=10;
hold on; %keep existing plot and overlay
plot(real(v(1:L:end)),imag(v(1:L:end)),'ro');
%option 'ro' uses red circles as markers
hold off;title('IQ Plot')
xlabel('In-phase');ylabel('Quadrature')
```

Additionally, plot the phase difference between the message samples, $\tilde{m}(t)$, and the received signal, $\tilde{v}(t)$. Take note to accommodate the filter delay, which is one-half the filter order. Further, truncate the list of received delayed samples to be the same length as the list of message samples. For reporting, present the IQ plot of the received signal.

(g) Recreate the IQ plot and phase difference plot from step (f), but add a phase offset, ϕ, at the demodulator. That is, implement the demodulator as in Figure 2.3 but with receiver oscillator $2e^{-j(2\pi f_c t+\phi)}$ for $\phi \neq 0$. What do you observe in your plots? Why?

(h) Recreate the IQ plot and phase difference plot from step (f), but insert both a phase offset, ϕ, and a frequency offset, f_Δ, at the demodulator. Try two or three different offsets, both negative and positive. What do you observe in your plots? Why? **Q2.4**

For reporting, present the IQ plot and the phase difference plot for one of your parameter selections. Briefly interpret the graphs.

2.4 Demonstration

For demonstration, use the Communication Laboratory mobile app to transmit a carrier tone. Descriptions of the app and how to download it are found in Section 7.1.1.

Record the transmission with sampling frequency `Fs=44100`. View the received signal using the `plotttf.m` function.

Demodulate the received signal, and view graphs of the I and Q components of the demodulated signal. What do you observe? Provide an explanation for your observations.

2.5 Summary

In Chapter 2, quadrature amplitude modulation and coherent quadrature demodulation have been explored. Complex baseband was presented as a convenient mathematical notation for representing QAM signals. The exercise also introduced the practical impairments of frequency and phase offsets, which are compensated in the receiver design to be implemented in Chapter 6. The explorations served to introduce the commands and conventions summarized in Table 2.1.

Table 2.1: Summary of commands introduced in Chapter 2.

Command	Description
`angle`	compute phase, in radians, of a complex number
`exp`	exponential function
`real`	real part
`imag`	imaginary part
`1j`	imaginary unit
`length`	returns length of vector
`sqrt`	square root
`xlabel`	create a label on the horizontal axis of a plot
`ylabel`	create a label on the vertical axis of a plot
`subplot`	create array of plots within a figure
`hold`	hold current graph (to create overlay)

CHAPTER 3

Digital Modulation

In this lesson we review digital modulation, with emphasis on the special cases of binary and quadrature phase-shift keying. The chapter equips students to implement the initial steps of the digital modulation chain depicted in Figure 1.1, mapping text to bits to symbols. Likewise, the chapter develops the inversion of these steps in the receiver chain. Through exercises and a demonstration, students visualize digital modulation via signal space plots and conduct both analytical and empirical characterizations of bit error rate.

3.1 Background

3.1.1 Digital modulation

The transmission chain in Figure 1.1 depicts a *digital modulator* that maps digital information to an analog waveform suitable for the communication channel. The mapping is typically performed on blocks of $\log_2 M$ binary digits (bits) at a time. When the mapping of bits to waveforms is performed without dependence on previously transmitted waveforms, then the modulator is *memoryless*. In addition, if the superposition principle holds for the mapping of the digital sequence to an analog waveform, then the modulator is *linear*. Throughout these chapters, we consider a class of memoryless linear modulators known as complex pulse amplitude modulation, or *quadrature amplitude modulation* (QAM). In QAM, bits are mapped to complex symbols, which then modulate pulse shapes. The *complex baseband* *signal*, or message, is given by

$$\tilde{m}(t) = \sum_n \tilde{a}[n] g_{\mathrm{tx}}(t - nT) \tag{3.1}$$

where $\tilde{a}[n]$ is the sequence of complex-valued symbols, $g_{\mathrm{tx}}(t)$ is a *pulse shape* waveform, and T is the symbol period. The symbol rate, or "*baud rate*," of $\tilde{m}(t)$ is $1/T$ symbols per second, and the *bit rate* is $(\log_2 M)/T$ bits per second for a *constellation* of M symbols. The pulse shape, $g_{\mathrm{tx}}(t)$, is normalized to unit energy,

$$\int_{-\infty}^{\infty} |g_{\mathrm{tx}}(t))|^2 dt = 1. \tag{3.2}$$

Design of the pulse-shaping function is considered in Chapter 4.

In this chapter, we consider the mapping of a sequence of bits to a sequence of symbols and the detection of a symbol from a noisy received data sample. Each symbol is a complex number selected from a constellation of M symbols. Four example symbol constellations are given in Figure 3.1. The three constellations shown in Figure 3.1(a-c) have points that differ only in their phase; this special case of digital quadrature amplitude modulation is known as digital phase modulation, or *phase-shift keying* (PSK). The name "keying" comes from physical operation of the electrical telegraph demonstrated by Samuel Morse in 1837 [27]. The Morse code is a variable-length binary code in which letters of the English alphabet are mapped to dots and dashes.

The constellation for $M = 2$ in Figure 3.1(a) is *binary phase-shift keying* (BPSK). The graph depicts modulated signals using axes corresponding to the two orthogonal signals in QAM: $g_{\mathrm{tx}}(t)\cos(2\pi f_c t)$ and $g_{\mathrm{tx}}(t)\sin(2\pi f_c t)$. BPSK is a real-valued constellation and does not utilize the quadrature carrier, $\sin(2\pi f_c t)$; however, bandwidth efficiency is improved by simultaneously placing separate values on both orthogonal carriers, $\cos(2\pi f_c t)$ and $\sin(2\pi f_c t)$. For $M = 4$, *quadrature phase-shift keying* (QPSK) is the resulting constellation and is shown in Figure 3.1(b). QPSK is essentially BPSK for both the I (cosine) and Q (sine) carriers. Figure 3.1(c) shows an 8-PSK constellation and an associated mapping of bits to symbols. Finally, Figure 3.1(d) shows a 16-QAM constellation; here, the symbols differ in both amplitude and phase. 64-QAM and 256-QAM constellations are often used in digital cable television and cable modem applications for high spectral efficiency over low-noise channels. The

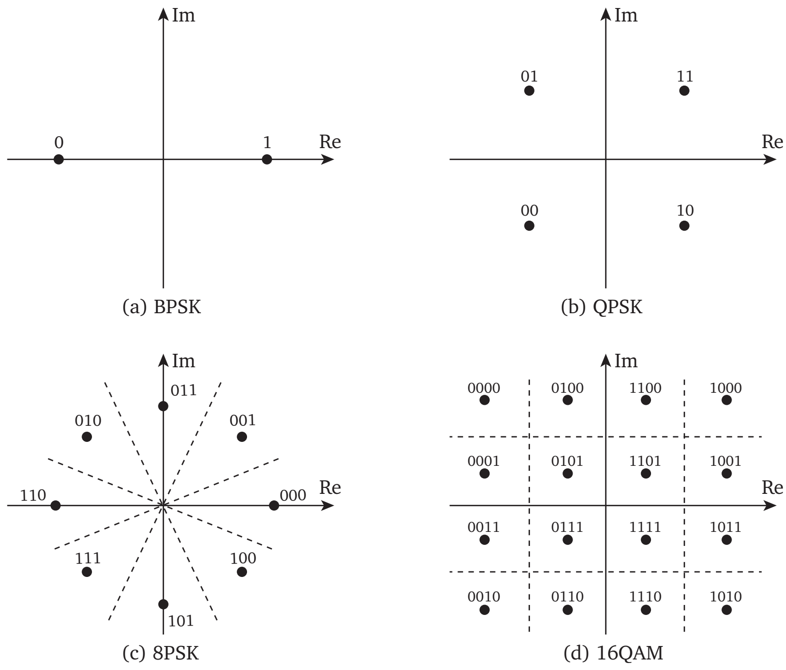


Figure 3.1: Illustrations of four symbol constellations: (a) BPSK; (b) QPSK; (d) 8-PSK; and (d) 16-QAM.

16-QAM constellation is used, for example, in the ITU-T V.22bis standard for digital communication over telephone networks.

It is convenient to normalize the constellation so that the M symbols, $\{\tilde{a}_1, \cdots, \tilde{a}_M\}$, have unit energy on average, i.e.,

$$\frac{1}{M}\sum_{i=1}^{M} |\tilde{a}_i|^2 = 1. \tag{3.3}$$

With an M-ary alphabet for a constellation, there are $\log_2 M$ bits per symbol; therefore, a single symbol error can result in up to $\log_2 M$ bit errors. And, there are M factorial possible assignments of bits to the M symbols. In general, a preferred assignment is to map bits to symbols so that neighboring symbols differ by only a single bit, thereby limiting the number of bit errors resulting from a symbol error. Such a mapping is called a *Gray code* or *unit-distance code.*

Table 3.1: Bit assignments for (a) BPSK and (b) QPSK digital modulation.

Input bit	Symbol
1	1
0	-1

(a)

Input bits MSB, LSB	Symbol
11	$(1+j)/\sqrt{2}$
01	$(-1+j)/\sqrt{2}$
00	$(-1-j)/\sqrt{2}$
10	$(1-j)/\sqrt{2}$

(b)

Suggested assignments of bits to symbols for BPSK and QPSK are given in Figure 3.1(a)-(b) and are repeated in Table 3.1. In QPSK it is customary that the most significant bit (MSB) is the first taken from the bit stream. With this convention, for each pair of bits QPSK modulates the first bit onto the I channel and the second bit onto the Q channel.

3.1.2 Symbol detection

At the receiver, we must detect the transmitted symbol and recover the associated bits. Referring to Figure 1.1, receiver processing results in a data stream modeled as a noisy sequence of complex-valued samples,

$$\tilde{y}[n] = \tilde{a}[n] + \tilde{w}[n]. \tag{3.4}$$

Design of receiver processing to justify this model is considered in Chapter 4. The received noisy samples $\tilde{y}[n]$ can be visualized using a *scatter plot*, as illustrated for QPSK in Figure 5.3. The following code creates this example.

```
%% Scatter plot example for QPSK or BPSK
N = 160; %number of symbols
M = 4; %4=QPSK or 2=BPSK
if(M==2)
   a = (2*round(rand(1,N))-1); %noiseless I
elseif(M==4)
   a = (2*round(rand(1,N))-1) ... %noiseless I
       + 1j*(2*round(rand(1,N))-1); %and add Q
   a = a/sqrt(2); %unit length QPSK symbols
```

```
end
EbperN0dB = 6;
EbperN0 = 10^(EbperN0dB/10);
varw = (EbperN0*log2(M))^(-1)/2; %per channel sample var
w = sqrt(varw)*(randn(1,N) + 1j*randn(1,N));
y = a + w; %add noise
% make scatter plot
figure;
plot(real(y),imag(y),'x','MarkerSize',8,'LineWidth',2);
title('Scatter Plot','fontsize',13)
xlabel('In-phase','fontsize',13);
ylabel('Quadrature','fontsize',13);
axis([-1.5 1.5 -1.5 1.5]);axis square% set axes extents
grid % draw grid lines
```

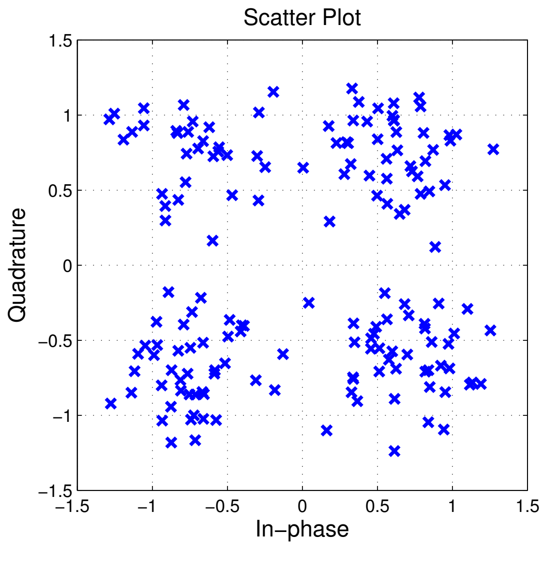


Figure 3.2: Example scatter plot for 160 noisy QPSK symbols.

Here we consider symbol detection, whereby a decision is made on a symbol-by-symbol basis; in contrast, for modulation schemes with memory a receiver jointly performs detection of an entire sequence of symbols. An intuitive symbol detection rule is to map a received, noisy symbol to the nearest constellation point. Indeed, for equally likely symbols and additive white Gaussian noise of equal variance in the I and Q channels, the nearest neighbor criterion minimizes the probability of symbol error.

The nearest neighbor rule, or *minimum distance detector*, defines *decision regions*; each decision region contains those points closest to a given constellation element. The decision regions are denoted by the dashed lines in Figure 3.1. For BSPK, decision regions are simply the half-planes separated by the imaginary axis; mapping to logical `1` or `0` is easily implemented via the inequality test, `(real(y) > 0)`. In low-level software, this test is merely a test on the sign bit of `y`. Similarly, for QPSK the decision regions are the four quadrants of the complex plane.

3.1.3 Bit and symbol error rates

The probability of symbol error depends on the signal-to-noise ratio in the received data stream, $\tilde{y}[n] = \tilde{a}[n] + \tilde{w}[n]$. The SNR of $\tilde{y}[k]$ is given by

$$\mathrm{SNR} = \frac{P}{N_0 \times W} = \frac{E_s/T}{N_0/T} = \frac{E_s}{N_0} \tag{3.5}$$

where P is the received signal power experienced at $\tilde{y}$, due to amplifier gains, channel loss, and receiver processing. The noise power spectral density is N_0 Watts per Hertz, to model primarily thermal noise in the receiver amplifiers; with a receiver bandwidth $W = T^{-1}$ Hz, the noise power becomes the product, $N_0 W$. And, E_s is the energy per symbol, while $E_b = E_s/\log_2(M)$ is energy per bit. Thus, we have the variance of $\tilde{y}$ conditioned on $\tilde{a}$ sent,

$$\mathrm{var}(\tilde{w}) = \mathrm{SNR}^{-1} = \frac{N_0}{E_s}. \tag{3.6}$$

BPSK

Considering the special case of BPSK, the energy per bit, E_b, is simply $E_b = E_s$. For $\tilde{a} = 1$, an error occurs if the real part, $y_I = \mathrm{Re}(\tilde{y})$, is less than zero. The real part carries half the noise variance, and hence the variance of $\mathrm{Re}(\tilde{w})$ is $N_0/(2E_b)$. This distribution is shown in Figure 3.3. The probability of bit error for BPSK when transmitting $\tilde{a} = 1$, then, is the area under the curve for $y_I \leq 0$.

Evaluating this integral we have

$$\begin{aligned} P_{BPSK} &= \int_{-\infty}^{0} \frac{1}{\sqrt{\pi N_0/E_b}} \exp\left(-\frac{(y_I-1)^2}{N_0/E_b}\right) dy_I \\ &= Q\left(\sqrt{\frac{2E_b}{N_0}}\right), \end{aligned} \tag{3.7}$$

where the *Q function*, $Q(x)$, is implemented via `qfunc(x)` and is defined as

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_{x}^{\infty} \exp\left(-t^2/2\right) dt. \tag{3.8}$$

By symmetry, the probability of error is the same when conditioned on the symbol 0 transmitted. The bit error rate (BER) for BSPK is shown versus E_b/N_0 (SNR per bit) in Figure 3.4.

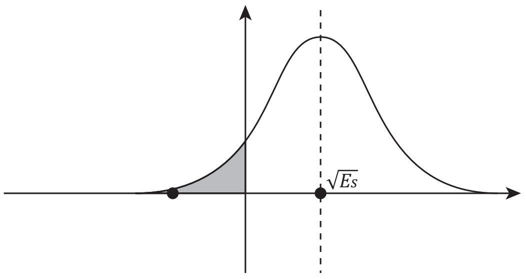


Figure 3.3: Probability of error for BSPK modulation on a binary Gaussian channel.

QPSK

Consider next the error probability for QPSK in two cases. First, suppose the bandwidth, $W = T^{-1}$, is identical for BSPK and QPSK and fix the energy per symbol, E_s. Then, QPSK provides twice the data rate as BPSK, but suffers a worse bit error rate. The worse bit error rate is evident from Figure 3.1 where we observe that the distance between nearest QPSK constellation points is only $\sqrt{2}/2$, compared to a distance of 2 for BPSK. Thus, following Equation 3.7 the BER for QPSK is given by $Q\left(\sqrt{2E_b/N_0}\right)$; but, for a *fixed symbol*

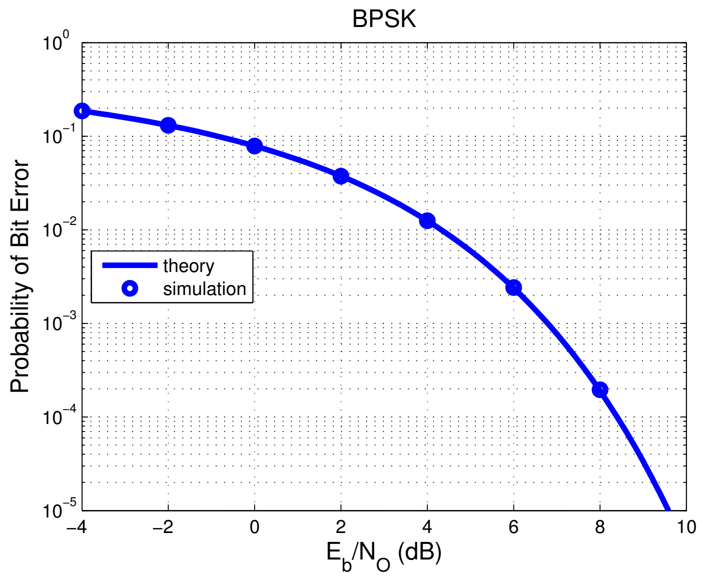


Figure 3.4: Bit error probability for BPSK versus E_b/N_0.

energy, the energy per bit, E_b, for QPSK is one-half the energy per bit for BSPK.

Second, set the data rate equal for BPSK and QPSK. Then, the signal bandwidth $W = T^{-1}$ for BPSK must double, thereby doubling the receiver noise power, $N_0 W$. Thus, for equal data rate, QPSK and BPSK have the same bit error rate, but QPSK uses only half the spectral bandwidth. For this reason, QPSK is said to be more *spectrally efficient* than BPSK.

In summary, in a bandwidth-limited application, QPSK is preferred over BPSK, because QPSK can offer the same rate of correct bits using only half the bandwidth. On the other hand, in a power-limited application BPSK may be preferred.

The detection error analysis above is a simple example demonstrating that communication engineers can analytically predict performance and evaluate system design trade-offs without expensive build-and-test cycles. And, the quality of a prediction is limited only by the validity of the attendant assumptions, such as channel models (including AWGN) and amplifier linearity. Given that engineers are able to design both the transmitter and receiver, modeling assumptions can typically be very accurate for commercial systems. Limiting bounds on performance provide additional insight to design trade-offs; Shannon's channel capacity bound is considered in Chapter 9.

3.2 Explorations

In this lesson students will implement the digital modulation and demodulation steps shown in Figure 1.1. In particular, students are given template functions in `char2psk.m` and `psk2char.m` found in Appendix C. From these templates, two functions will be constructed: one function to map a character string to a sequence of symbols, and a second function to map noisy received symbols to a character string. The code templates provide examples of the format, syntax, and comment style for students to consider. Use of well-commented modular code will facilitate construction of an end-to-end acoustic digital modem.

The steps adopted in the code templates for mapping an ASCII character string to bits are illustrated in Table 3.2. The command `double` converts an ASCII character to a decimal label. Next, the command `dec2bin` converts the decimal numbers to a binary representation of specified length, which here is 8 bits. Then, `.'` performs a 90° rotation of the bit array, turning rows into columns, and the command `reshape` reads the transposed bit array, column by column, into a single long row.

A lab report should provide answers to the four questions enumerated in the page margins and the bit error rate curve measured in the demonstration.

(a) Create a function to convert a character string to a sequence of symbols. The inputs should be a character string and an identifier to select a symbol constellation of size 2 (BPSK) or 4 (QPSK); the outputs should be a list of bits and a list of complex-valued symbols, each of unit magnitude. Study the example template given in Appendix C and complete the portions that are left blank.

(b) Create a function to implement a minimum distance decoder for BPSK and QPSK modulation. The inputs should be a

Table 3.2: Example illustrating one possible path to convert an ASCII character string to bits.

```
str='abc';
str_dec=double(str);
str_bin=dec2bin(str_dec,8);
bits=reshape(str_bin.',1,8*length(str));

str_dec =
    97    98    99

str_bin =
    01100001
    01100010
    01100011

bits =
    011000010110001001100011
```

complex-valued string of noisy symbols and an identifier to select BPSK or QPSK modulation; the outputs should be a list of detected bits and a character string.

(c) Verify that your two functions from steps (a) and (b) work together to properly map characters to bits to symbols and back again. (Save these functions for re-use in subsequent laboratory sessions as you continue to design and construct an acoustic modem.)

Q3.1 (d) Suppose the pulse-shaping function, $g_{\text{tx}}(t)$, in Equation 3.1 has two-sided bandwidth of $(1+\alpha)/T$ Hz, where T is the symbol period. What is the bandwidth efficiency of BPSK in *bits per second per Hertz*? What is the bandwidth efficiency of QPSK? Of 64-QAM?

Q3.2 (e) Suppose a random bit stream is BPSK modulated such that each symbol has energy E_b. In an AWGN channel, what is the average probability of bit error for a minimum distance decoder as a function of E_b/N_0, Hint: you may leave the answer in terms of the Q-function. In your report, include a rough sketch that accompanies your analysis by showing probability as an area under a curve.

(f) Create a scatter plot of 1000 simulated received QPSK symbols. Refer to the example given in Section 3.1.2 and set the noise power for the complex additive white Gaussian noise to be 5 dB per symbol. From the analysis given in Section 3.1.3, what BER do you expect? **Q3.3**

(g) Qualitatively and intuitively, describe how BER and E_s are evidenced in the signal space scatter plot. **Q3.4**

(h) [**Optional**] Extend your functions from steps (a) and (b) to implement 8-PSK modulation.

(i) [**Optional**] Extend your functions from steps (a) and (b) to implement 16-QAM modulation.

(j) [**Optional**] Implement a function to estimate E_b/N_0, in decibels, from symbol rate samples at the receiver; the inputs should be M, $\tilde{a}[n]$ and $\tilde{y}[n]$.

(k) [**Optional**] Convert the code example in Section 3.1.2 into a function, `SimScatterPlot.m`. The inputs are `N,M,EbperN0dB`. The outputs should be an observed error rate and a scatter plot.

3.3 Demonstration

Create a simulator for QAM symbols in additive white Gaussian noise. Via simulation, create bit error (BER) curves for BPSK and QPSK as a function of E_b/N_0 over the range -4 dB to 10 dB in 0.25 dB steps. Use `N=1e6` random trials. (Note that, in general, to simulate an error rate of 10^{-d} you should use at least $N = 10^{d+1}$ random trials.) Compare your empirical result to your conclusion from step (e). You can re-use your code from step (b) to implement the minimum distance decoder, and the simulation was considered in step (f) (and in step (k)). Hint: one method for counting bit errors is

```
BER = length(find(Rxbits-Txbits))/N;
```

where `Rxbits` and `Txbits` are the length `N` bit streams at the receiver and transmitter, respectively.

3.4 Summary

In Chapter 3, digital modulation has been explored with emphasis on the special cases of binary and quadrature phase-shift keying. The exercises prepare students to implement the initial steps and final steps of the modem processing chain depicted in Figure 1.1: mapping text to bits to symbols at the transmitter, and mapping receiver samples to symbols to bits to text. The explorations introduced several new commands and conventions summarized in Table 3.3.

Table 3.3: Summary of commands introduced in Chapter 3.

Command	Description
>	returns 1 at indices where inequality is true
`qfunc`	Q function gives tail probability of a Gaussian pdf
`log2`	base 2 logarithm
`log10`	base 10 logarithm
∧	exponentiation, e.g., `10∧6` is 10^6
`find`	returns indices of nonzero entries
`round`	nearest integer
`axis`	control axis scaling and appearance
`switch, case,end`	switch among several cases based on expression
`if,elseif, else,end`	conditionally execute statements
`end`	can also be used as pointer to end of an array
`.'`	turn rows into columns or vice versa; array transpose
`sign`	returns 1 if an element is greater than zero, 0 if zero, and -1 if less than zero
`reshape`	reformat array by reading elements columnwise
`bin2dec`	convert binary string to decimal integer
`dec2bin`	convert decimal integer to a binary string
`char`	convert ASCII integer codes to a character array
`double`	convert to double precision

CHAPTER 4

Pulse Shaping and Matched Filtering

In Chapter 3 the digital modulation and detection parts of a baseband modem were implemented. Here, development of the baseband modem is continued by incorporating pulse shaping at the transmitter and filtering at the receiver. The complete baseband processing chain is illustrated in Figure 4.1.

This chapter begins with an abbreviated review of pulse shaping. The review summarizes design issues associated with intersymbol interference and maximization of receiver signal-to-noise ratio. Through exercises and a demonstration, students implement transmitter and receiver filtering and visualize received baseband BPSK signals using an eye diagram.

4.1 Background

A sequence of symbols, $\tilde{a}[n]$, is not suitable for direct transmission over a communication channel. Symbols must be converted to a waveform for transmission. In a process known as *pulse shaping*, symbols are converted to a sampled-data baseband message; then, as seen in Figure 1.1, the sampled data are converted to an analog voltage signal through a digital-to-analog converter (DAC) and mixed to a frequency band suitable for the communication channel. From Equation 3.1, the sampled-data baseband message is given by

$$\tilde{m}[k] = \sum_n \tilde{a}[n] g_{\text{tx}}(kT_s - nT) \tag{4.1}$$

where $\tilde{a}[n]$ is the sequence of complex-valued symbols, $g_{\text{tx}}(t)$ is a pulse shape waveform, T is the symbol period, and $T_s = 1/f_s$ is the

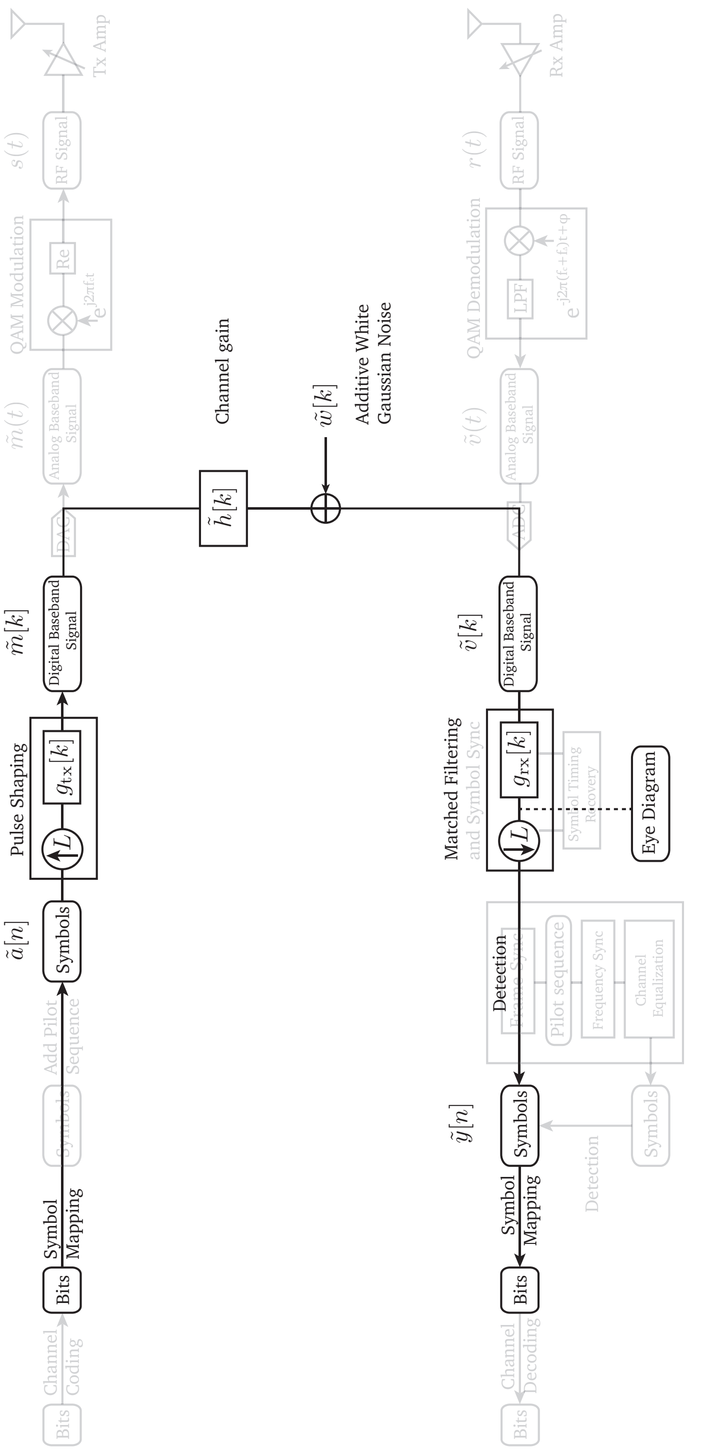


Figure 4.1: Fractionally sampled baseband system model for physical layer digital communication processing.

DAC sampling interval. The summation in Equation 4.1 creates a superposition of scaled and time-shifted copies of the sampled pulse waveform, $g_{\text{tx}}[k]$. The summation is a convolution of the upsampled symbol sequence, $\tilde{a}_{\uparrow}[k]$, and the sampled pulse shape waveform, $g_{\text{tx}}[k]$. The upsampling operation merely inserts $L-1$ zeros between consecutive samples of $\tilde{a}[n]$, with $L = T/T_s$, in order to match the baseband sampling interval, $T_s < T$. Note that here we have required the symbol period, T, to be an integer multiple of the sampling period, T_s. The processing is illustrated in Figure 4.2.

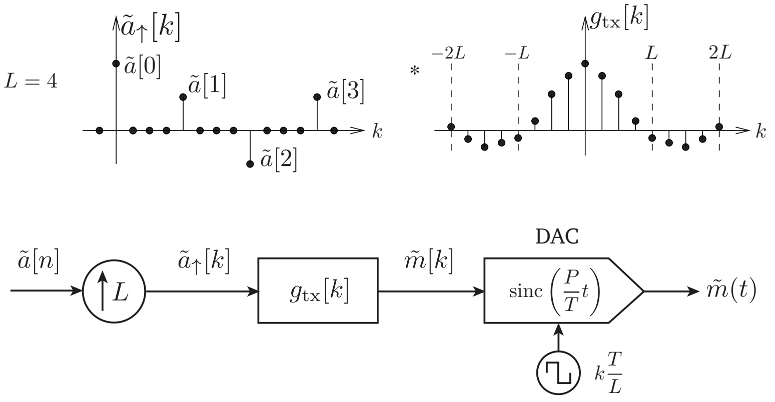


Figure 4.2: Illustration of pulse shaping for $L = 4$; upsampling by L is followed by convolution with the pulse shape, $g_{\text{tx}}[k]$.

At the receiver, ideal performance would result in the n^{th} output, $\tilde{y}[n]$, equal to the n^{th} input symbol, $\tilde{a}[n]$. However, in practice $\tilde{y}[n]$ is corrupted by both noise and interference from other symbols, known as *inter-symbol interference* (ISI). The receiver filter seen in Figure 4.3 has impulse response $q[k]$ and is designed to meet two goals: prevent ISI and suppress noise.

4.1.1 Inter-symbol interference

The first design goal for the receiver filter is to prevent inter-symbol interference. In an idealized system with no noise and ideal channel $\tilde{h}[k] = \delta[k]$, the pulse shaping filter at the transmitter, $g_{\text{tx}}[k]$, and the linear filter, $g_{\text{rx}}[k]$, at the receiver are in series; therefore, their

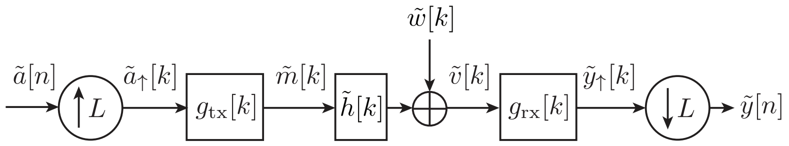


Figure 4.3: Illustration of the fractionally sampled complex-baseband data model for additive noise, $\tilde{w}[k]$, and a channel described by filter impulse response $\tilde{h}[k]$.

combined effect is a single filter with impulse response, $p(t)$, given by the convolution of $g_{\text{tx}}(t)$ and $g_{\text{rx}}(t)$, and frequency response given by the product of the two frequency responses, $P(f) = G_{\text{tx}}(f)G_{\text{rx}}(f)$. Thus, the downsampled output sequence is

$$\tilde{y}[n] = \sum_k \tilde{a}[k]p(nT - kT) = \sum_k \tilde{a}[k]p(T(n-k)). \tag{4.2}$$

To force $\tilde{y}[n] = \tilde{a}[n]$, we therefore require that the pulse $p(t)$ must have gain 1 at the time origin and a value of zero at every integer multiple of the symbol interval. Any waveform with this property is called a *Nyquist pulse*, and the property is simply stated as $p(nT) = \delta[n]$, where $\delta[n]$ is the *Kronecker delta sequence*,

$$\delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0. \end{cases} \tag{4.3}$$

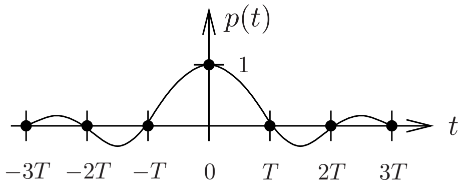
In the frequency domain, we equivalently have

$$p(nT) = \delta[n] \longleftrightarrow \frac{1}{T}\sum_{k=-\infty}^{\infty} P(f - k/T) = 1. \tag{4.4}$$

That is, the superposition of frequency-shifted spectra must sum to T at all frequencies. These time and frequency domain characterizations of a Nyquist pulse are illustrated in Figure 4.4.

4.1.2 Matched filter

The second design goal for the receiver filter is to combat noise. To maximize the signal to noise ratio at $\tilde{y}[n]$, application of the



$$p(nT) = \delta[n]$$

$$\frac{1}{T}\sum_{k=-\infty}^{\infty} P\left(f - k/T\right) = 1.$$

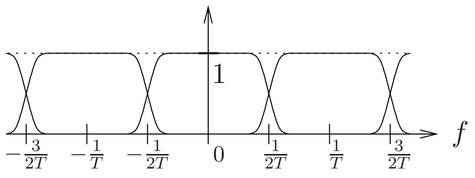


Figure 4.4: Time-domain (top) and frequency-domain (bottom) illustrations of the Nyquist pulse property.

Cauchy-Schwarz inequality leads to the conclusion that $g_{\mathrm{rx}}(t)$ must be matched to the the pulse waveform:

$$g_{\mathrm{rx}}(t) = g_{\mathrm{tx}}^*(-t) \longleftrightarrow G_{\mathrm{rx}}(f) = G_{\mathrm{tx}}^*(f). \tag{4.5}$$

Thus, for SNR-maximizing ISI-free performance, the filter design requires:

1. The convolution of the pulse waveform, $g_{\mathrm{tx}}(t)$, and the receiver filter impulse response, $g_{\mathrm{rx}}(t)$, must yield a Nyquist pulse. In the frequency domain, we have $G_{\mathrm{tx}}(f)G_{\mathrm{rx}}(f) = P(f)$.

2. The receiver filter must be a matched filter for the pulse waveform; thus, $G_{\mathrm{rx}}(f) = G_{\mathrm{tx}}^*(f)$.

Together, these two requirements imply that $|G_{\mathrm{tx}}(f)|^2$ must satisfy the Nyquist pulse criterion.

A common design choice for the pulse shape, $G_{\mathrm{tx}}(f)$, is the *square-root raised cosine* (SRRC) pulse, which is given in the time-domain as [18]

$$g_{\mathrm{SRRC}}(t) = \begin{cases} \frac{1}{\sqrt{T}}\frac{\sin(\pi(1-\alpha)t/T)+(4\alpha t/T)\cos(\pi(1+\alpha)t/T)}{(\pi t/T)(1-(4\alpha t/T)^2)} & t \neq 0, \pm\frac{T}{4\alpha} \\ \frac{1}{\sqrt{T}}(1-\alpha+4\alpha/\pi) & t = 0 \\ \frac{\alpha}{\sqrt{2T}}\left\{\left(1+\frac{2}{\pi}\right)\sin\left(\frac{\pi}{4\alpha}\right)+\left(1-\frac{2}{\pi}\right)\cos\left(\frac{\pi}{4\alpha}\right)\right\} & t = \pm\frac{T}{4\alpha}. \end{cases} \tag{4.6}$$

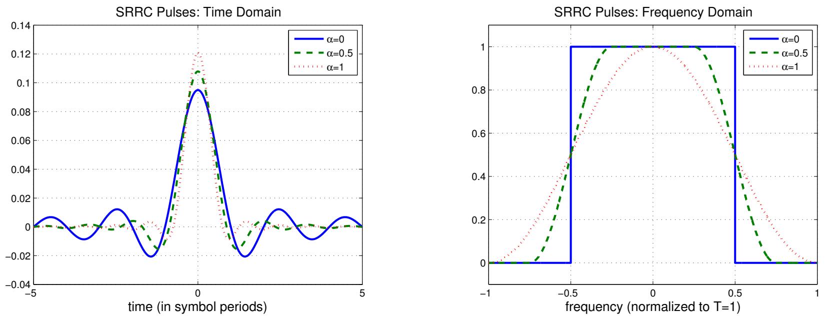


Figure 4.5: Square-root raised cosine pulse in time domain (left) and frequency domain (right).

Because $g_{\mathrm{SRRC}}(t)$ is real-valued and symmetric about the origin, we have

$$g_{\mathrm{rx}}(t) = g^*_{\mathrm{SRRC}}(-t) = g_{\mathrm{SRRC}}(t). \tag{4.7}$$

The SRRC filter provides a parametrized family of filter designs. The parameter $\alpha \in [0,1]$ is called the *excess bandwidth* and provides a trade-off. The two-sided bandwidth of $g_{\mathrm{SRRC}}(t)$ is $(1+\alpha)/T$; a larger α spreads the frequency content of the pulse to a larger bandwidth, but results in a more rapid decay of the time-domain signal. The SRRC pulse is illustrated in Figure 4.5. Note that, for $\alpha > 0$, the square-root raised cosine pulse is *not* a Nyquist pulse; however, the convolution of $g_{\mathrm{SRRC}}(t)$ with itself does produce a Nyquist pulse.

For digital implementation, the filter $g_{\mathrm{SRRC}}(t)$ is sampled and truncated for implementation as a finite impulse response (FIR) filter, $g_{\mathrm{SRRC}}[k]$, such that $g_{\mathrm{SRRC}}[k] = 0$ for $|k| > DT$. Setting the tails of the filter to zero slightly violates the Nyquist pulse property and therefore results in small amount of inter-symbol interference. However, for $\alpha \in [0,1]$ large, the fast decay of the time signal limits the truncation error and hence the ISI. This trade-off is explored in the exercises.

Note that the impulse response $g_{\mathrm{SRRC}}[k]$ is symmetric about its midpoint and is therefore a linear phase filter. For an impulse response with `Ng = length(g_tx)` samples, the delay of the linear phase filter is `(Ng - 1)/2` samples. Thus, in the baseband processing of Figure 4.3 the first symbol appears in the upsampled sequence $\tilde{y}_{\uparrow}[k]$ with a delay of `(Ng -1)` due to the combined action of the transmitter and receiver filters.

4.1.3 Eye diagram

An *eye diagram* may be used to visualize received symbols and better understand error behavior. For real-valued symbols, the eye diagram is a plot that superimposes many T-second segments of $\text{Re}\{\tilde{y}(t)\}$. For sampled baseband data, the eye diagram is made by `eyediagram.m` by superimposing L-sample segments from the real part of the upsampled signal, $\tilde{y}_{\uparrow}[k]$. For visualization, $L \geq 8$ samples per symbol is preferred. The traces on the plot depict the voltage paths taken by the received signal as it progresses from one symbol to the next. Owing to the filtering caused by the transmitter, the channel, and the receiver, the path taken can depend on many neighboring symbols. When there is neither noise nor inter-symbol interference, then the diagram passes through the ideal symbol values, creating the opening seen in Figure 4.6 for BPSK and metaphorically called an "open eye." When the eye is open, symbol decisions are reliable; when the eye becomes closed, errors occur. In the noiseless case, the eye may narrow due to ISI, and therefore limit the reliability of decisions when noise is present.

Symbol timing, the topic of Chapter 5, can likewise be visualized in the eye diagram. Ideally, the downsampled data, $\tilde{y}[n]$, should be acquired by taking samples at the time location where the eye opening is the widest, thereby providing the most noise robustness. If the symbols are complex-valued, then eye diagrams may be plotted for both the I and Q channels separately.

The function `eyediagram.m` takes four inputs: the upsampled data sequence, $\tilde{y}_{\uparrow}[k]$, the downsampling factor, L, the number of symbols, N, and (optionally) the flag `'complex'` if the user wishes to view both the I and Q channels. The first sample of the input $\tilde{y}_{\uparrow}[k]$ is assumed to correspond to the first symbol time instant, with subsequent symbols occurring every L^{th} sample.

4.2 Explorations

In this lesson, students implement a digital baseband modem for an additive white Gaussian noise channel with ideal channel response,

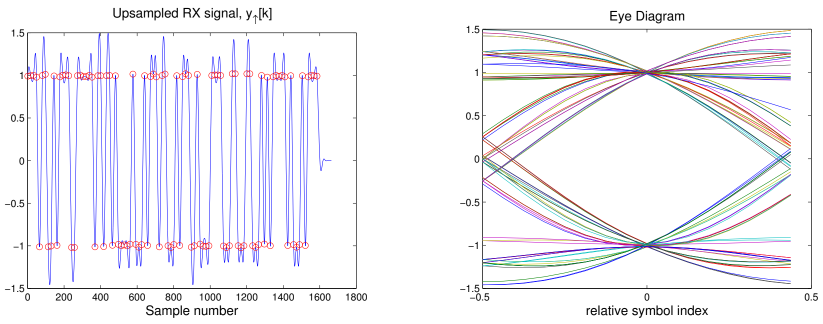


Figure 4.6: Left: upsampled received BPSK signal, $\tilde{y}_\uparrow[k]$ with symbol-rate samples marked as circles. Right: eye diagram.

$\tilde{h}[k] = \delta[k]$. The processing chain is illustrated in Figure 4.1 and Figure 4.3. The baseband modem is implemented by combining the modulation and demodulation routines from Chapter 3 with the upsampling, filtering, and downsampling steps reviewed in Section 4.1. These new steps can be implemented using the `conv` command and the `(1:L:end)` indexing syntax, both introduced in Chapter 1. The SRRC filter design is conveniently provided by the function `srrc.m` given in Appendix C. Students are reminded to review the engineering skills briefly discussed in Appendix A. Short subsections of code developed in each chapter will be integrated into a single working modem; thus, purposeful attention to a few simple practices will greatly simplify work as the lessons progress.

For your lab report, provide a brief answer to each of the seven questions enumerated in the page margins. In addition, provide plots for Questions (b), (e), and (f); with each figure, provide a descriptive caption. Finally, the report should include your code for Questions (f) and (g).

Q4.1 (a) Use the functions `srrc.m` and `plottf.m` (with `Ts=1`) to construct and view square-root raise cosine pulse shape filters with the following values for excess bandwidth, $\alpha \in [0, 1]$, truncation half-width, D, and upsampling, L. (Note L must be an integer.)

- $D = 3, \alpha = 0.10, L = 10$.
- $D = 3, \alpha = 0.75, L = 10$.
- $D = 8, \alpha = 0.10, L = 10$.

Qualitatively, what do you observe? Does the Nyquist property approximately hold for g_{tx} with the settings above? Why or why not?

(b) Using the three settings given in step (a), plot the result of the filter convolved with itself, `conv(g_tx,g_tx)`. Does the Nyquist property approximately hold for the settings above? Why or why not? What do you observe about the effects of α and D when comparing the three plots to each other? From the three plots, identify the largest amplitude sample of `conv(g_tx,g_tx)` contributing to ISI; i.e., which symbol-rate sample deviates most from the ideal Nyquist pulse? Finally, suppose the symbol $\tilde{a}[20] = 1$ is transmitted; for each of the three settings above; determine the interference (sign and amplitude) caused by that symbol at the received symbol for sample index $n = 23$. **Q4.2**

(c) Short answer: Consider a digital-to-analog converter (DAC) in Figure 4.2 operating at 8000 samples per second. **Q4.3**

- Suppose you desire a symbol period of 10 milliseconds. What should you choose for the integer upsampling factor, L?
- Can you choose L so that the baud rate is 1500 symbols per second? Why or why not?

(d) Short answer: Express the quantities listed below in terms of these modem parameters: DAC sampling rate f_s; QAM constellation size, M; upsampling factor, L; excess bandwidth, α; SRRC truncation, D. **Q4.4**

- Symbol period.
- Baud rate.
- Bit rate.
- Approximate pulse duration, in seconds, for $g_{\text{tx}}(t)$.
- Two-sided bandwidth of the baseband analog signal.

- Number of samples in the pulse shaping filter impulse response, $g_{\text{tx}}[k]$. (Verify using `srrc` and `length`.)

Q4.5 (e) Create a random string of 1000 bits, and encode the bits as BPSK symbols:

```
bits=round(rand(1,1000));
a=2*bits-1;
```

Create an eye diagram for a noiseless channel using the following design parameters for your pulse shape and matched filter.

- $D = 3, \alpha = 0.75, L = 15$
- $D = 3, \alpha = 0.10, L = 15$
- $D = 8, \alpha = 0.10, L = 15$

Note that the function `eyediagram.m` expects that the first sample in $\tilde{y}_{\uparrow}[k]$ corresponds to the first symbol; therefore, the filter delay should be considered in passing arguments to the `eyediagram` function.

Q4.6 (f) For $D = 3, \alpha = 0.75, L = 15$, add white Gaussian noise to the simulated sequence of matched filter outputs, $\tilde{y}_{\uparrow}[k]$. Use noise variance of $\sigma^2/2$ in the real and imaginary parts. For $\sigma^2 = 0.5$ describe what you observe in the eye diagram. Repeat for $\sigma^2 = 0.02$.

Q4.7 (g) In preparation for explorations given in Chapters 5 through 7, create a *software template*, in four parts, for the acoustic modem graphically depicted in Figures 1.1, 4.1, and 4.3. The four parts should be: (i) Modem parameters; (ii) Transmitter; (iii) Channel Impairments; and, (iv) Receiver. Within each part, create a section (%%) for each processing step. Provide a one or two line descriptive caption for the functionality of each section. Each section should include generation of a graphic for use in debugging. For a data source at the transmitter, allow a choice between a text input or random bits.

Fill in your outline with those sections already developed in Chapters 1 through 4; leave incomplete the other sections. Preparation of this commented template will serve to accelerate your progress through Chapters 5 through 7.

(h) [**Optional**] From Section 4.1 we can conclude that a desirable pulse shape has low energy in spectral sidelobes to limit bandwidth and has low temporal side-lobes that can contribute to inter-symbol interference. Are these conflicting or complimentary goals and why?

(i) [**Optional**] What value of α produces a sinc pulse? Is the sinc pulse a Nyquist pulse? Is the convolution of a sinc pulse with itself a Nyquist pulse? Is the convolution of *any* Nyquist pulse with itself also a Nyquist pulse?

4.3 Demonstration

Use your complex baseband simulator to consider sample timing errors with and without noise. For example, a sample timing offset of 2 samples can be inserted for a row-vector `y_up` via the command

```
Offset = 2; y_up = [ rand(1,Offset) y_up];
```

First compute the BER and plot an eye diagram for two pulse designs in the noiseless case with a sample timing offset of 5 samples:

- $D = 8, \alpha = 0.1, L = 21$
- $D = 8, \alpha = 0.75, L = 21$

Which pulse is more susceptible to sample timing errors? Why?

Second, repeat with 6 dB SNR in the real part of $\tilde{y}_{\uparrow}[k]$ and noise of equal variance in the imaginary part. Note,

$$6\,\text{dB} = 10\log_{10}\frac{1}{\sigma^2} \Rightarrow \sigma^2 = 10^{-6/10}.$$

(You are invited to optionally experiment by changing the noise variance and the timing offset.)

4.4 Summary

In Chapter 4, pulse shaping and matched filtering have been introduced to complete a baseband signal representation for a digital modem. The eye diagram has been used to visualize the baseband received signal and to observe the effects of ISI and symbol timing error for an AWGN channel. The explorations introduced the new commands summarized in Table 4.1.

Table 4.1: Summary of commands introduced in Chapter 4.

Command	Description
`srrc`	function to design square-root raised cosine pulse
`eyediagram`	function to display an eye diagram
`clear all`	removes all variables, globals, functions
`close all`	deletes all figures whose handles are not hidden

CHAPTER 5

Synchronization

In Chapter 5 students build sub-components for a baseband digital modem with the aim of synchronizing the receiver to the transmitter. *Symbol timing* will be estimated for the downsampling of the matched filter outputs, and *frame timing* will be recovered to align receiver processing with a data packet. In conjunction with the symbol timing and frame timing synchronization, students will also account for a gain and phase due to the channel.

Non-ideal effects degrade received signals. These effects include: a) symbol timing and frame synchronization errors; b) offset in the receiver oscillator's frequency and phase relative to the transmitter; c) inter-symbol interference (ISI) due to a frequency-selective fading channel. These non-ideal effects are viewable in the example BPSK eye diagrams of Figure 5.1. Ideally, channel impairments due to timing, frequency offset, phase offset and channel filtering would be jointly estimated at the receiver, along with the bits encoded in the transmitted waveform. However, for manageable complexity at the receiver, the tasks are typically addressed in a staged manner, with impairments sequentially addressed in several steps. A system-level view of the receiver processing is depicted in Figure 5.2. This chapter employs *block processing* of matched filter outputs to implement synchronization steps. Processing steps to mitigate additional impairments are considered in subsequent chapters. Frequency recovery is implemented in Chapter 6, and Chapters 8, 10, and 11 consider impairments due to a frequency-selective fading channel. *Adaptive processing* is also widely employed to address unknown and time-varying channel impairments and is considered in Chapter 12.

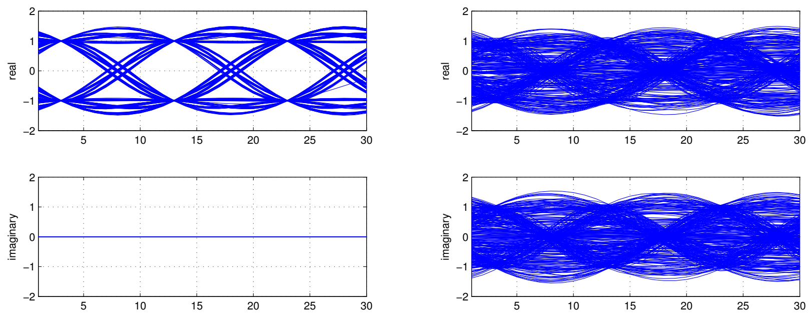


Figure 5.1: Eye diagrams without (left) and with (right) channel impairments. Data are shown for a BPSK constellation.

5.1 Background

5.1.1 Symbol timing

The matched filter outputs are fractionally sampled at L times the symbol rate; that is, there are L samples per symbol period. Our first step is to select which of the L times best corresponds to the proper symbol time. We attempt to discover this symbol time in the presence of channel impairments due to oscillator frequency mismatch, oscillator phase mismatch and ISI – all of which will be addressed subsequent to symbol timing. Here we describe a block-processing version of the constant modulus algorithm (CMA) [17]. Intuition comes from the eye diagram at the left in Figure 5.1: namely, on average, the variability in the magnitude, or "modulus," of the samples is smallest at the proper sampling instant. Further, notice that frequency and phase offsets, as seen in Chapter 2, serve only to spin a PSK constellation in the complex plane, thereby leaving the scatter plot of matched filter outputs lying in an annulus. With poor symbol timing, the width of this annulus spreads to fill much more of the complex plane. This graphical intuition is displayed in Figure 5.3.

Thus, to put this CMA principle into action, we first denote by γ_p the mean of the absolute values of matched filter outputs with

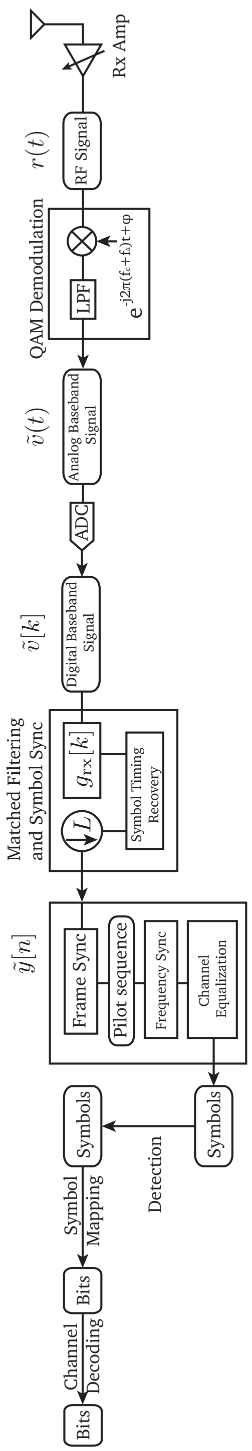


Figure 5.2: Receiver processing chain.

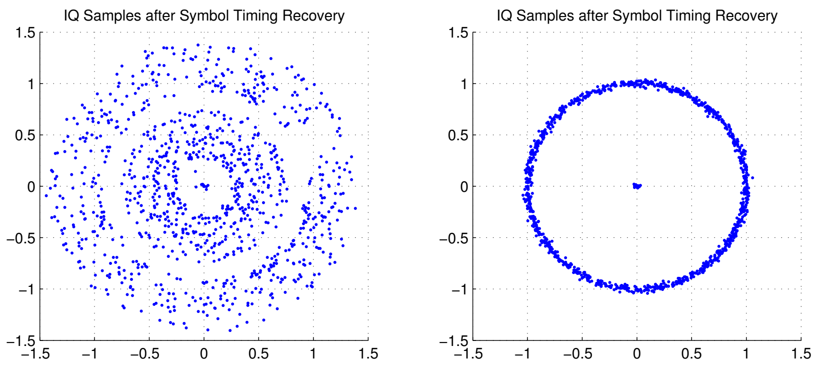


Figure 5.3: Scatter plots of sampled matched filter outputs for QPSK signalling. Left: poor symbol timing. Right: good symbol timing.

offset p samples and subsampled to the symbol rate:

$$\gamma_p = \text{mean}\left\{|\tilde{y}_\uparrow[p+nL]|, \quad n = 1, 2, \ldots, \lfloor N/L \rfloor - 1\right\}. \tag{5.1}$$

Then, minimization of the variance by selection of p can be written

$$\widehat{p} = \min_{p\in\{0,\ldots,L-1\}} \sum_{n=0}^{\lfloor N/L \rfloor - 1} \left(|\tilde{y}_\uparrow[p+nL]| - \gamma_p\right)^2. \tag{5.2}$$

That is, which of the L downsampled subsequences has the smallest variance? See the commands `var` and `min` for simple calculation. Note also the indexing syntax, `(p:L:end)`, for starting at index p, with $0 \le p \le (L-1)$ and counting by L to the end of a list. In the laboratory procedures, you will implement this block-processing version of the CMA. In some commercial systems, an adaptive version is implemented [36].

5.1.2 Frame timing for flat channels

Frame synchronization is typically performed through use of a training sequence of symbols (also called "marker sequence" or "pilots") that is known in advance to both the transmitter and receiver. Given that symbol timing has already been performed, the frame timing step may be computed using the matched filter outputs downsampled to the symbol rate: $\tilde{y}[n] = \tilde{y}_{\uparrow}[nL]$. In the explorations, students will use a training sequence to find the beginning of a received frame.

In a *flat fading* channel model, the transmitted signal, $s(t)$, simply experiences a gain, a phase, and a delay. That is, $\tilde{h}(t) = A\delta(t-\tau)$ is the model of the channel impulse response for some complex-valued gain A and a non-negative delay, τ. The channel model is known as flat fading, because the Fourier spectrum of a delta function is constant versus frequency. (A *frequency-selective fading channel* model is considered in Chapters 8, 10, and 11.) Writing the gain and phase as a complex scalar, $\tilde{H}$, we have

$$\tilde{v}(t) = \tilde{H} \times \tilde{m}(t-\tau) + \tilde{w}(t), \tag{5.3}$$

where $\tilde{w}(t)$ denotes baseband additive white Gaussian noise (AWGN) modeling a disturbance due primarily to thermal noise at the receiver. For the AWGN model, the maximum likelihood (ML) estimate, $\widehat{\tau}$, of the delay is computed using the absolute value of the matched filter output:

$$\begin{aligned} \tilde{y}(t) &= \int \tilde{v}(\lambda)\tilde{m}^*(\lambda - t)d\lambda \\ \widehat{\tau} &= \arg\max_t \; |\tilde{y}(t)|. \end{aligned} \tag{5.4}$$

The matched filter output also gives the ML estimate of the unknown amplitude,

$$\tilde{H}_{est} = \frac{\tilde{y}(\widehat{\tau})}{\int |\tilde{m}(\lambda)|^2 d\lambda}. \tag{5.5}$$

That is, $\tilde{H}_{est}$ is the ratio of the observed peak value to the known noiseless autocorrelation peak from the training signal $\tilde{m}(t)$. Note that the complex-valued scalar $\tilde{H}_{est}$ contains both the gain and the phase modeling the flat fading channel response. For sampled data and ± 1 marker symbols, Equation 5.5 tells us

```
H_est = peak / length(pilots)                (5.6)
```

where `peak` is the complex-valued sample of maximum absolute value, `peak` $= |\tilde{y}(\hat{\tau})|$.

A good marker sequence must have a very sharp autocorrelation peak to provide noise robustness. For a training sequence, here we

Table 5.1: Code to generate Figure 5.4.

```
pilots=[1, 1, 1, 1, 1, -1, -1, 1, 1, -1, 1, -1, 1];
N_tr=length(pilots);
a=pilots;
xc=conv(a,fliplr(a)); % auto-correlation
figure;stem(-(N_tr-1):N_tr-1, xc);
xlabel('lag number');title('autocorrelation')
```

consider a Barker code. A length N_{tr} Barker code $\{a_k\}_{k=0}^{N_{tr}-1}$ is a sequence of binary symbol values, ± 1, such that

$$\left| \sum_{k=0}^{N_{tr}-i-1} a_k a_{i+k} \right| \leq 1 \quad \text{for } i \neq 0. \tag{5.7}$$

For example, a length-13 Barker code is

$$\{a_k\}_{k=0}^{12} = \{1,\ 1,\ 1,\ 1,\ 1,\ -1,\ -1,\ 1,\ 1,\ -1,\ 1,\ -1,\ 1\}. \tag{5.8}$$

The autocorrelation sequence for the length-13 Barker code is shown in Figure 5.4, and the code to generate the plot is given in Table 5.1.

Next, we illustrate timing recovery by embedding the Barker code in noise and computing the cross-correlation between the noisy signal and the known Barker code. The result is seen in Figure 5.5, where the visible peak serves to identify the location of the marker sequence within the noisy signal. The relationship between the sample index of the correlation peak and the first index of marker sequence is considered in the Explorations, step (a).

Extension of frame timing for robustness to a frequency-selective fading channel is considered in Chapters 8, 10, and 11.

5.2 Explorations

In this lesson, students continue development of an acoustic modem by implementing symbol timing recovery and frame timing recovery in the baseband receiver processing chain. The baseband channel model considered is a flat fading channel with additive white Gaussian noise. The processing chain is illustrated in Figures 5.2, 4.1, and 4.3.

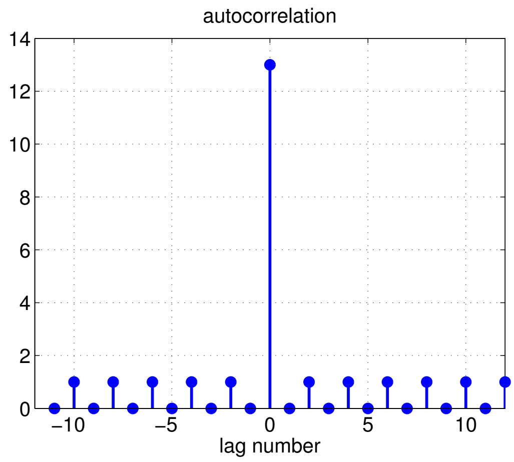


Figure 5.4: Autocorelation for the length-13 Barker code.

For the laboratory exploration, work through the five steps below to implement a baseband modem in simulation. The suggestions below explicitly enumerate items for a brief lab report: an equation for Question 5.1 in step (a), one figure for Question 5.2, a brief paragraph for Question 5.3, one figure for Question 5.4, and code appended for Question 5.5.

Important: to aid with debugging and understanding, insert a graph at each step of the processing chain depicted in Figures 1.1 and 5.2; refer to Table 5.3 for code suggestions to create graphs. Keep commented and tested code from this chapter for your continuing use as you build towards a working acoustic modem.

(a) To prepare for frame synchronization in step (d), consider the following preliminary exercise. Create `pilots`, a sequence of pilot symbols, (or "marker symbols") using a length-13 Barker code. Insert the pilots into a length 100 signal starting at location `start` less than 88. For example,

```
y = zeros(1,100);
tau = 23; %delay >= 0; start=1+tau
y( (1+tau) : (1+tau)+length(pilots)-1) = pilots;
```

Use convolution to compute the cross-correlation between y

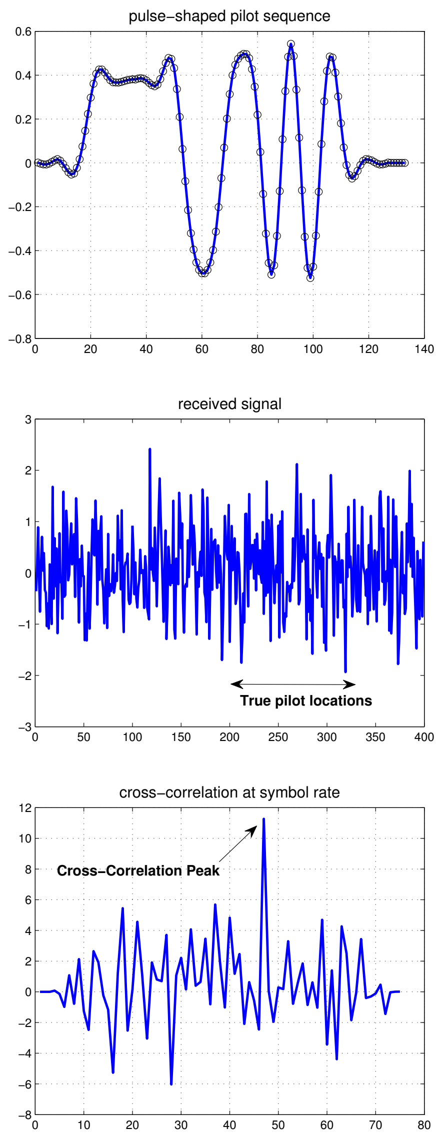


Figure 5.5: Frame timing example for a flat fading channel. Top: marker sequence convolved with pulse waveform. Middle: received waveform before matched filtering. Bottom: correlation of marker sequence with downsampled matched filter outputs. Peak reveals location of the marker sequence.

and `pilots`. Write a general expression to solve for the sample index of `y` at which the *first symbol of the pilot sequence occurs*. The solution is found from the sample index of the peak of the absolute value of the cross-correlation output. Express your answer in terms of the sample index of the peak of the correlation output and the length of the pilot sequence. If the correlation output is named `CorrOutput`, then the peak value and the index at which it occurs can be found via **Q5.1**

```
[value,indx] = max(abs(CorrOutput));
peak = CorrOutput(indx);
```

Test your solution via simulation by changing `tau` and computing the cross-correlation. Visualize the technique by plotting the absolute value of the cross-correlation output.

(b) As a second preparatory step, construct a baseband modem simulator using the snippets of code you have created in previous chapters. Refer to Figure 4.3 and Figure 5.2 for processing roadmaps and re-use your codes from Chapter 4. The purpose in this step is to implement and test the pulse-shaping and matched filtering steps, without any synchronization issues. At the top of your script, assign values for the channel impairments listed in Table 5.2. *Verify that the symbol sequence is accurately reproduced.* Test your baseband modem simulator for some small $N \leq 10$, `H=1`, and by setting the noise power and delay time impairment variables listed in Table 5.2 to zero. To this end, you can directly compute and use the correct timing, as known from your choice of `tau` and the filter delays at the transmitter and receiver, as considered in Section 2.1.5. That is, in this preparatory step, you do not yet need to implement symbol timing or frame timing.

Table 5.2: Variables to define baseband channel impairments.

`H`	complex-valued gain and phase, flat channel model
`noise_var`	variance of complex AWGN
`tau`	channel delay; for simplicity, an integer number of sampling intervals

Suggestions:

- Recommended pulse shaping parameters: `D=5`, `alpha=0.5`, and `L=10`.
- For symbols, create a N random BSPK symbols, as in Chapter 4, and prepend the pilot symbols used in step (a).
- Implement the sample-rate channel time delay, `tau`, by inserting your received message into a longer signal, `v`, similar to the construction of `signal` in step (a) above. Make the signal, `v`, 100 samples longer than the sampled baseband message.
- Add noise to `v` using variance `noise_var/2` in each of the I and Q channels.
- For the baseband transmitter steps, use the `conv` command, the indexing syntax `(1:L:end)`, and the `srrc.m` function introduced in previous chapters. (Ensure that `srrc.m` and any other custom function is present in your working directory or path.)

(c) **[Optional]** Using the baseband processing from step (b), view the changes to the received IQ symbols for channel gain $H \neq 1$ and nonzero noise variance.

(d) Armed with baseband processing from step (b), next consider estimation of the proper symbol timing. Augment your baseband receiver processing chain from step (b) by implementing the CMA symbol timing recovery algorithm described in Equation 5.1 and Equation 5.2. To begin, review the definition of *variance* of a list of numbers and calculation of variance using `var` in MATLAB. Re-use your code from step (b); set $N = 500$. For your tests, a suggested parameter set for pulse shaping is: `D=5`, `alpha=0.5`, and `L=10`.

Table 5.3 provides suggested plotting commands that you can insert into your code to use the power of visualization in debugging and interpreting your results. (Note: you may need to modify variable names to match your code.)

Q5.2 Verify the correctness of your algorithm using scatter plots of IQ samples before and after symbol timing recovery. Test your

Table 5.3: Code suggestions to generate figures.

```
% transmitted signal
figure;
plot(m,'bo');title('Baseband Tx Message')
% eye diagram
eyediagram(y_up,L,length(a),'complex');
% CMA criterion
figure;plot(0:L-1,cma_test);
title('CMA variance criterion')
% upsampled MF outputs; superimpose downsampled values
plot(p_est:length(y_up),real(y_up(p_est:end)));hold on
plot(p_est:L:length(y_up),real(y_up(p_est:L:end)),'ro')
title('Downsampling of MF Outputs')
% video illustrating all choices for downsampling
figure
for kk=1:L
  plot(kk:length(y_up),real(y_up(p_est:end)));hold on
  plot(kk:L:length(y_up),real(y_up(p_est:L:end)),'ro');
  title('Symbol timing from CMA','fontsize',13);hold off
  pause(0.5)% view at two frames per second
end
% correlation output
figure;plot(xc);
title('Cross-correlation for frame timing')
% scatter plot
figure;
plot(real(y_up(start:L:end)), ...
   imag(y_up(start:L:end)),'ko');
grid on; axis image;
title{'IQ samples after symbol timing recovery')
```

symbol timing recovery using several values for the channel delay parameters: `tau` $= 0, 2, 4, 6$ and keeping other channel impairments benign: `H=1` and `noise_var=0`.

Q5.3 After you have verified noiseless performance, experiment by re-running your section of code using the following values for the remaining channel impairments: `H= 2*exp(1j*pi/4)` and `noise_var=0.01`. Generate one figure illustrating performance of the CMA symbol timing recovery; provide a descriptive caption and list simulation values for both the modem parameters and the channel impairment parameters. Provide a brief intuitive description of why the CMA algorithm works.

(e) Augment your baseband receiver processing chain from step (d) to insert frame synchronization explored in step (a). Note the frame synchronization operates on the downsampled matched filter outputs, $\tilde{y}[n]$. Verify operation of your frame synchronization by running the simulation with nonzero `tau` greater than L.

(f) Augment your baseband receiver processing chain from step (e) to insert the flat fading channel equalization from Equation 5.5. Verify operation of your frame synchronization by running the simulation with several choices of gain and phase for `H`.

Q5.4 Generate one figure illustrating the combined performance of the CMA symbol timing recovery, frame timing recovery, and flat fading channel equalization; provide a descriptive caption and list simulation values for both the modem parameters and the channel impairment parameters.

Q5.5 For your report, append code for the software simulator as augmented at step (f). Provide two or three figures illustrating execution of your modem simulator as performed in the Demonstration below; with the figures, provide a descriptive caption and a table listing both the modem parameters and the channel impairment parameters.

5.3 Demonstration

Demonstrate your complex-baseband modem simulator using a mix of parameter settings for the channel impairment variables given in

Table 5.2.

5.4 Summary

In Chapter 5, techniques have been introduced to recover symbol timing and frame timing at the receiver. In addition, frame timing also provided equalization for a flat fading channel response. The exercises provide a complete working baseband modem; in the next chapter, the processing will be extended to operate the modem using bandpass signals at an acoustic carrier frequency. The explorations in this chapter introduced the new commands summarized in Table 5.4.

Table 5.4: Summary of commands introduced in Chapter 5.

Command	Description
`min`	returns the value of the smallest element and the index (or indices)
`max`	returns the value of the largest element and the index (or indices)
`var`	computes the variance of a list of numbers
`fliplr`	reverses the column ordering of an array

CHAPTER 6

Frequency Recovery

In Chapter 6 students complete a software-defined bandpass digital modem. The subcomponents developed in previous lessons for a baseband system are combined with quadrature modulation using an acoustic carrier frequency. *Frequency recovery* is implemented at the receiver to account for the impairment due to a frequency mismatch between the transmitter and receiver. Additionally, channel sounding experiments explore flat versus frequency-selective channel models.

6.1 Background

The complex-baseband equivalent model completed in Chapter 5 is shown in Figure 6.1. In the illustration, the baseband signal is shown upsampled to L samples per symbol, and the model is therefore referred to as "fractionally sampled." In this chapter, the processing is extended to perform analog quadrature amplitude modulation (QAM) and demodulation using an acoustic carrier frequency. Due to the inherent mismatch between transmitter and receiver oscillators, a frequency recovery step is inserted in the receiver processing chain. The bandpass model is depicted in Figure 6.2.

6.1.1 Frequency recovery

For coherent demodulation, the local oscillator must have frequency and phase synchronized with the transmitter. Consider a receiver oscillator given by $e^{j(2\pi(f_c+f_\Delta)t+\phi)}$ with a frequency mismatch f_Δ

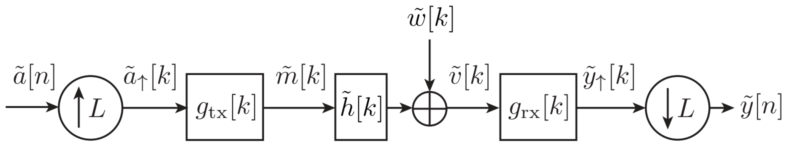


Figure 6.1: Fractionally-sampled complex-baseband model.

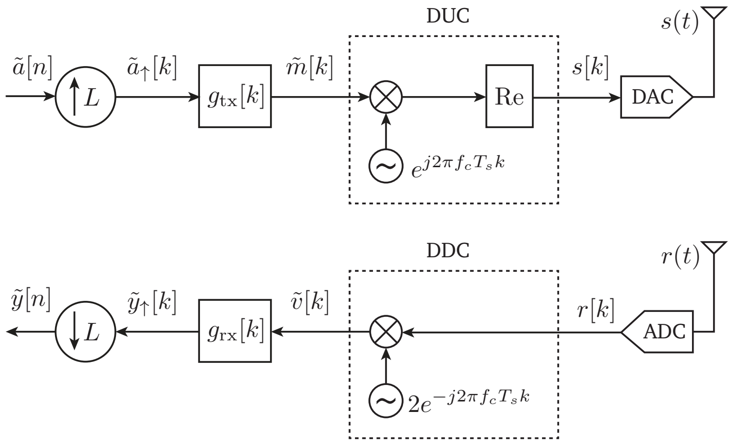


Figure 6.2: Baseband processing and direct digital quadrature modulation to an acoustic carrier frequency. Top: Transmitter. Bottom: Receiver.

Hz and a phase mismatch ϕ radians. Using the convenient complex baseband representation of Figure 2.4 we have that the received message is

$$\tilde{v}(t) = \tilde{m}(t)e^{-j(2\pi f_\Delta t+\phi)}. \tag{6.1}$$

Suppose $\tilde{a}[n]$ is a sequence of known message symbols; thus, after proper receiver filtering, symbol-timing recovery and frame-timing recovery, the received symbols suffer a time-dependent phase error:

$$\tilde{y}[n] = \tilde{a}[n]e^{-j(2\pi f_\Delta nT+\phi)}, \tag{6.2}$$

where T is the symbol period and the index n counts the sampled marker sequence from $n = 0$. The phase offset term $e^{-j\phi}$ may be incorporated into the complex-valued channel gain and therefore is recovered with frame timing, as seen in Section 5.1.2. Thus, we need only recover the frequency offset, f_Δ.

In Chapter 5, a known sequence of pilot symbols was used to recover frame timing, channel gain, and channel phase. Here, we use the same pilot sequence to perform frequency recovery, as well. From Equation 6.2 and with $\phi = 0$, the phase difference between known pilots and received symbols is a linear function of sample index,

$$\text{angle}\left\{\frac{\tilde{a}[n]}{\tilde{y}[n]}\right\} = (2\pi f_\Delta T)\, n, \tag{6.3}$$

where T is the symbol interval. Notice that the quantity $(2\pi f_\Delta T)$ gives the incremental phase error in *radians per symbol* and specifies the slope of the linear function. Thus, a line fit to the unwrapped phase angle of the known markers divided by the received symbols gives an estimate for the frequency recovery, as illustrated in Figure 6.3. In the example, the true frequency offset of -0.05 radians per symbol interval is estimated as -0.0570 radians per symbol using a noisy received marker sequence of 13 pilot symbols.

The MATLAB command `p = polyfit(x,y,d)` finds the coefficients of a polynomial $p(x)$ of degree d that best fits the data, $y(i)$, at sampling instants, $x(i)$, in a least squares sense. For additive white Gaussian noise, the least-squares fit yields the maximum likelihood estimate. The result `p` from the command is a row vector of length `d+1` containing the polynomial coefficients in decreasing order. In particular, for a linear fit, we set `d=1`, provide samples `x=0:N-1`, and obtain `p` equal to $[2\pi f_\Delta T,\, \phi]$. Note that the `unwrap` command should be applied to the phase difference to avoid jumps in the phase were it instead reported on the interval $[-\pi, \pi)$. Further, recall the phase difference between a and b can be obtained via `angle(a ./ b)`. The need for the `unwrap` operation is illustrated in Figure 6.4; in the example there are 25 symbols, $\pi/10$ radians per symbol phase error, and $\phi = \pi/20$. To correct the phase of the data symbols, the linear phase can be added to the symbols for indices beginning with N; that is, to properly correct the phase, the indexing must continue with the same symbol counting used in the line fit.

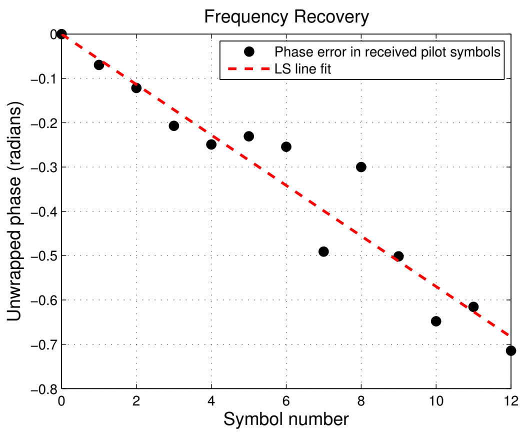


Figure 6.3: Frequency recovery example for a flat fading channel.

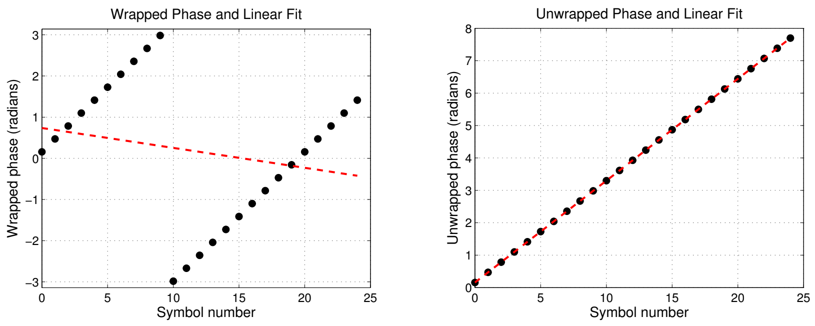


Figure 6.4: Phase difference (dots) and linear fit (dashed line) performed without (left) and with (right) phase unwrapping.

6.1.2 Frequency-selective fading channel model

Here, as in Chapter 5, the remediation of channel impairments assumes a flat fading channel with impulse response given by

$$\tilde{h}(t) = A\delta(t - \tau). \tag{6.4}$$

Thus, the impairments in the baseband signal due to the flat fading channel model are limited to a gain, a phase, and a time delay.

More generally, a *frequency-selective fading channel* model provides a gain and phase that vary as a function of frequency. The most common frequency-selective channel model is a *multi-path* model, in which the received signal is a noisy sum of delayed and attenuated versions of the transmitted signal. For K paths, each with complex-valued gain A_k and path delay τ_k, the so-called K-tap impulse response model is

$$\tilde{h}(t) = \sum_{k=1}^{K} A_k \delta(t - \tau_k). \tag{6.5}$$

The different delayed signal versions result from multiple propagation paths between the transmitter and receiver. At radio frequencies, causes of multi-path include atmospheric ducting and reflections from buildings or mountains. The constructive and destructive interference of the multiple arriving signals causes a frequency-dependent attenuation. Timing recovery and equalization for a frequency-selective fading channel are considered in Chapter 8.

6.1.3 Channel measurement

A frequency-selective fading channel is described by an impulse response of length greater than one, in contrast to flat fading channel characterized by a single complex-valued scalar for gain and phase. In this subsection, we provide background for an optional exercise in which the cross-correlation idea from Chapter 5 is used to estimate a channel impulse response. This measurement is sometimes called *channel sounding.*

For a binary marker sequence $c[n]$ of length N with good autocorrelation properties, the autocorrelation is approximately a scaled Kronecker delta sequence:

$$\frac{1}{N}\sum_{k} c[k]c^{*}[k-n] \approx \delta[n-N]. \tag{6.6}$$

With this observation and the associative property of convolution, we can compute a cross-correlation between a known marker sequence and a noisy received symbol sequence to estimate a symbol-rate baseband representation of the channel impulse response, $\tilde{h}[n]$. Let $\tilde{y}[n] = \tilde{h}[n] \star c[n] + \tilde{w}[n]$ be a noisy received version of the marker sequence, $c[n]$, distorted by convolution with the baseband channel impulse response, $\tilde{h}[n]$. Here $\star$ denotes linear convolution. The cross-correlation with a known marker sequence yields

$$\begin{aligned}
\frac{1}{N}\left\{\tilde{y}[n] \star c^*[-n]\right\} &= \frac{1}{N}\left\{\tilde{h}[n] \star c[n] + \tilde{w}[n]\right\} \star c^*[-n] \\
&= \tilde{h}[n] \star \left\{\frac{1}{N} c[n] \star c^*[-n]\right\} + \frac{1}{N}\left\{\tilde{w}[n] \star c^*[-n]\right\} \\
&\approx \tilde{h}[n] \star \delta[n-N] + \frac{1}{N}\left\{\tilde{w}[n] \star c^*[-n]\right\} \\
&\approx \tilde{h}[n-N]. \qquad (6.7)
\end{aligned}$$

One choice for a long binary marker sequence is a pseudo-random noise (PN) sequence of $+1$ and -1 symbols. Thus, a PN code and cross-correlation processing can provide an estimate of the channel impulse response.

6.2 Explorations

In this lesson, students complete development of a bandpass QAM modem simulator by implementing quadrature modulation to an acoustic carrier frequency and creating a frequency recovery step in the receiver processing chain. The baseband channel model considered is a flat fading channel with additive white Gaussian noise. The processing chain is illustrated in Figure 1.1 and Figure 5.2.

For the laboratory exploration, work through the three steps below to implement a bandpass modem in simulation. Recommendations for preparing a brief, descriptive laboratory report are included in the suggested steps.

Important: to aid with debugging and understanding, insert a graph at each step of the processing chain depicted in Figure 1.1.

(a) Augment and revise your complex-baseband modem simulator from Chapter 5 to implement a bandpass model.

- Let the modem parameters, such as L, D, α, f_c, and f_s, be variables clearly assigned using `load` or set at the beginning of the simulator script, rather than coded as numerical values throughout.
- At the sampled bandpass signal, $s[k]$, insert simulated impairments for a flat fading channel model: a complex-valued channel gain, a delay, and AWGN.
- Code your quadrature demodulator to include a simulated frequency offset, f_Δ.
- First, verify the operation of your bandpass modem simulator for zero frequency offset, a channel gain of 1, and no delay.
- Second, verify the operation of your bandpass modem simulator when channel impairments include a non-zero delay and a complex-valued channel gain not equal to 1. For reporting, show one IQ scatter plot for decoded symbols and list the simulation parameters of the modem, including channel impairments. **Q6.1**

(b) Augment your bandpass receiver processing chain from step (a) by implementing the frequency recovery algorithm described in Section 6.1.1. And, use the estimated frequency offset to correct the phase of the received symbols.

To begin, review the online help for the commands `unwrap` and `polyfit`. Test your frequency recovery algorithm using the following suggested modem parameters:

- acoustic sampling rate of 44100
- carrier frequency of 14.5 kHz
- length 13 Barker sequence for pilot symbols
- $N = 200$ random BPSK symbols
- Upsampling `L` to achieve a symbol interval, T, of approximately 2.5 milliseconds
- Pulse shape: half-width `D=5`; excess bandwidth `alpha=0.5`

Run tests for four values of frequency offset: $2\pi f_\Delta T = 0$, -0.00005, -0.0005, and -0.005. **Q6.2**

For each test, view the results by plotting a scatter plot before and after the frequency recovery step. For reporting, simply provide IQ scatter plots for two of the four cases considered, along with a list of the simulation parameters for the modem, including channel impairments.

Q6.3 (c) Throught the explorations in Chapter 6, you have augmented your acoutic modem software template to include frequency recovery. Submit you commented code.

(d) **[Optional]** Use cross-correlation with a pseudo-random BPSK marker sequence to estimate a channel model, $\tilde{h}[n]$. For your experiment design, the following parameters are recommended:

- Bandpass operation: sampling frequency $f_s = 44100$ sps; carrier frequency $f_c = 5.5\,\text{kHz}$;
- Pulse shape: half-width $D = 5$; upsampling $L = 19$; excess bandwidth $\alpha = 0.25$;
- Marker sequence: $N = 4095$ random BPSK symbols.

Place a speaker and microphone approximately 30 cm apart and approximately 65 cm from a wall (or other large surface). Face the microphone and speaker directly towards the wall, rather than towards each other.

(i) Perform acoustic transmission and recording, as explored in Chapter 1.

(ii) Plot the time and frequency domain views of the transmitted bandpass signal. Include this plot in your report. For your transmitted signal, what is the *fractional bandwidth*, defined as $2W/f_c$, where W is the one-sided bandwidth of the baseband message and f_c is the carrier frequency?

(iii) Compute the cross-correlation of $c[n]$ and $\tilde{y}[n]$ and plot the absolute value. Define $t = 0$ as the first correlation peak. Using 343 m/s as the approximate speed of sound, label the horizontal axis as path length in cm. Compare your cross-correlation peaks with the physical path lengths present in your experimental setup. Include this plot and a brief statement of comparison in your report.

(iv) Using the complex-valued cross-correlation output, plot the frequency response of your measured channel. The command `freqz` may be useful. Interpret the impulse response and frequency response plot of your measured channel. Is the channel accurately described by a single gain and phase?

(e) [**Optional**] Modify your code in step (d): select `L` to implement a signal with two-sided bandwidth of only approximately 290 Hz. Repeat your channel sounding experiment. In this case, is the channel accurately described by a single gain and phase? Why or why not? Do you expect your conclusion to depend on carrier frequency, f_c? Why or why not?

6.3 Demonstration

Demonstrate your bandpass modem simulator implemented in step (b) for a combination of channel impairment variables of your choice.

6.4 Summary

In Chapter 6, a technique has been introduced to recover a frequency offset at the receiver. In addition, flat versus frequency-selective fading channel models have been explored via an optional channel sounding experiment. The explorations introduced new commands summarized in Table 6.1.

Table 6.1: Summary of commands introduced in Chapter 6.

Command	Description
`unwrap`	undo modulo 2π wrapping of a list of phase values
`polyfit`	compute a polynomial fit to data
`freqz`	plot a discrete-time frequency response

CHAPTER 7

Acoustic Modem

In Chapter 7 students employ the software-defined radio components completed in previous chapters to implement an acoustic modem. A series of experiments is performed to quantify the performance of the modem.

7.1 Background

7.1.1 Mobile app

An acoustic quadrature amplitude modulation transmitter is available as a mobile app. The app may be obtained for Apple devices at the iTunes App Store[1], and a version for Android devices is available at Google Play.

The acoustic transmitter app can be operated in two modes: carrier tone and text transmission. In carrier tone mode, the app simply transmits the waveform $\cos(2\pi f_c t)$ for a 400 millisecond duration at a user-specified carrier frequency $f_c < 22.05\,\text{kHz}$. In the text transmission mode, the pilot sequence may be selected as either a 13-symbol Barker code or a 51-symbol pseudo-random sequence. The routine `makepilots.m` is provided in Appendix C to define these pilot sequences. The data payload is fixed at 50 ASCII characters; a text string of fewer than 50 characters entered on the app is augmented with the underscore symbol, _, to achieve a 50 character payload. The user-defined inputs for text transmission are listed in Table 7.1.

[1] App Name: Acoustic Transmitter; App SKU: Acoustic_Transmitter.

The home page and settings page of the app are illustrated in Figure 7.1.

Table 7.1: User-defined inputs for acoustic transmitter app.

Input	Description
text	a text message of 50 or fewer characters
or	
tone	carrier tone only
f_c	acoustic carrier frequency
L	upsampling factor (samples per symbol), resulting in baud rate $44100/L$
D	pulse shape half-width, in symbols
α	pulse shape excess bandwidth
M	constellation size (BPSK or QPSK)
pilots	Barker-13 or pseudo-random 51

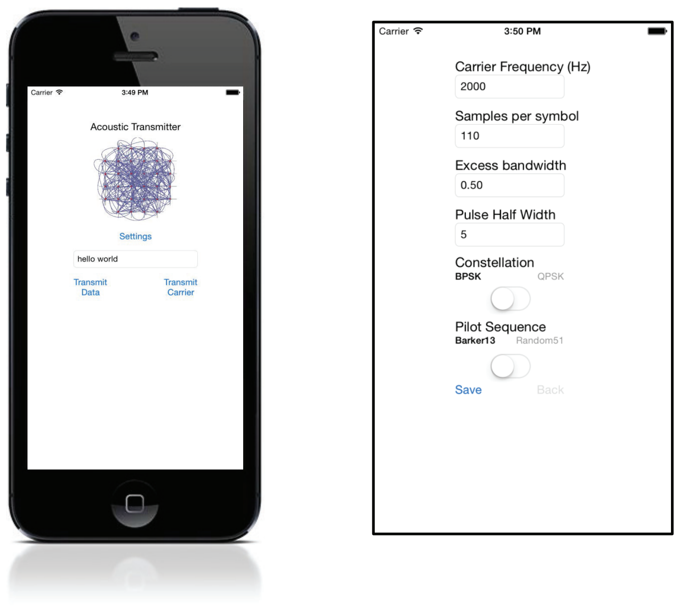
7.1.2 Data packet

A packet is the basic unit of most digital communication systems. Packets may vary in structure depending on the protocols implementing them, but typically contain a *header* and a *payload*. In Chapter 5 a sequence of pilot symbols was introduced as a header, and the data symbols comprised the payload. This simple packet structure is illustrated in Figure 7.2.

7.1.3 Spectral efficiency

Physical layer design for a communication system is the use of constrained resources to maximize system performance. The resources include bandwidth, power, system complexity, and cost. Performance objectives may include, for example, bit rate, error rate, outage probability, delay, battery life, etc. One performance measure is *spectral efficiency*, measured in bits per second per Hertz. For the QAM modem developed in the previous chapters, spectral efficiency is given by

$$\eta = \frac{\log_2 M}{1+\alpha} \text{ b/s/Hz.} \tag{7.1}$$



hello world

Carrier Frequency (Hz)
2000

Samples per symbol
110

Excess bandwidth
0.50

Pulse Half Width
5

Constellation
BPSK

Pilot Sequence
Barker13

Figure 7.1: Communication laboratory mobile app.

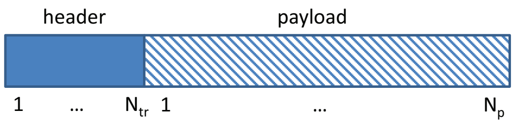


Figure 7.2: A simple packet structure with a header of N_{tr} symbols followed by a data payload of N_p symbols.

7.1.4 Differential PSK modulation

A differentially encoded phase modulation (differential phase shift keying, or DPSK) allows the phase estimate to be obtained from the previous symbol interval. This allows for robustness to, for example, growing phase errors due to a small inaccuracy in frequency offset recovery or due to a slowly time-varying mismatch between the transmitter and receiver oscillators.

Given any M-ary PSK encoding of a bit stream to produce unit-length complex-valued symbols, $\tilde{a}_k, \quad k = 1, 2, \ldots, N$, the corresponding DPSK symbols, $\tilde{b}_k, \quad k = 0, 1, 2, \ldots, N$, can be iteratively obtained:

$$\tilde{b}_k = \tilde{b}_{k-1}\tilde{a}_k, \quad k = 1, 2, \ldots, N, \tag{7.2}$$

where $\tilde{b}_0$ can be any unit-length initial condition. Let θ_k denote the phase of $\tilde{b}_k$, then from Equation 7.2 we have that $\tilde{a}_k = e^{j(\theta_k - \theta_{k-1})}$. Following [27], the demodulator outputs, at symbol times $k-1$ and k, are given by

$$\begin{aligned} \tilde{y}_{k-1} &= \sqrt{E_s}e^{j(\theta_{k-1}-\phi)} + \tilde{n}_{k-1} && (7.3)\\ \tilde{y}_k &= \sqrt{E_s}e^{j(\theta_k-\phi)} + \tilde{n}_k && (7.4)\\ & && (7.5) \end{aligned}$$

where ϕ is the (unknown) carrier phase and n_k is zero-mean additive white Gaussian noise at the matched filter outputs. Then, the decision variable for the phase detector is the difference,

$$\begin{aligned} \tilde{y}_k\tilde{y}_{k-1}^* &= E_s e^{j(\theta_k-\theta_{k-1})} + \\ & \quad \sqrt{E_s}\left\{e^{j(\theta_k-\phi)}\tilde{n}_{k-1}^* + e^{-j(\theta_{k-1}-\phi)}\tilde{n}_k\right\} + \tilde{n}_k\tilde{n}_{k-1}^*. \end{aligned} \tag{7.6}$$

Hence, the mean is independent of the carrier phase, ϕ. And, if we neglect the product term $\tilde{n}_k\tilde{n}_{k-1}^*$ at high SNR, then we observe from the addition of noise sequences in Equation 7.6 that the difference between DPSK and phase-coherent PSK is roughly that the noise variance is twice as large. A more careful analysis reveals that 4-phase DPSK suffers about 2.3 dB SNR loss versus phase-coherent QPSK, and that the performance difference between 2-phase DPSK and BPSK is even less. Thus, DPSK versus coherent PSK provides a design trade-off often used in digital communications: a reduction of implementation complexity can be obtained at the cost of slightly inferior noise performance.

7.2 Explorations

In this chapter, students implement and experimentally verify an acoustic modem. The acoustic modem should be designed to achieve a spectral efficiency $\eta > 1$. For the transmitter, students may use their code from Chapter 3 and Chapter 6 to transmit from one computer to another. Alternatively, the app described in Section 7.1.1 may be used as an implementation of the transmitter on a mobile device.

The laboratory report should contain these brief elements:

(i) A table listing choices for *all* modem parameters: audio sampling frequency, carrier frequency, symbol period, pulse shape parameters, digital modulation format, pilot sequence;

(ii) Also include in the table the following parameters: packet length (in symbols); data rate (bits per second); the experimentally estimated frequency offset, in Hertz, between the transmit and receive devices; bit error rate (BER); the number of bits measured to report the BER; and, the estimated frequency offset, in Hertz, between the transmit and receive devices.

(iii) A short paragraph describing the physical experimental setup for the transmitter and receiver;

(iv) A time and frequency plot (`plottf`) of the received audio-frequency bandpass signal;

(v) Scatter plot of the received symbols before timing recovery and downsampling;

(vi) Scatter plot of the received symbols after timing recovery and downsampling;

(vii) A final scatter plot of the received symbols after frequency recovery.

Students are encouraged to consider four suggestions given below for achieving the design goals; see Appendix A for a brief discussion of recommended software design practices.

- *Use modular code.* Implementation is facilitated by use of several modular scripts. For example, digital modulation functions were completed in Chapter 3, and the simulator from

Chapter 6 can be easily partitioned into one routine for transmission and one for reception. The function `srrc.m` for pulse design and the function `firlpf.m` for low-pass filter design have been used in previous chapters and are provided in Appendix C. Be sure to have all functions available in your working directory or have their directories added to the current search path; see the commands `path` and `addpath` to get, set, or add to the search path.

- Adopt a strategy to set *common parameters* at both the transmitter and receiver. Consistently use variable names, such as `L` or `fcarrier`, in your codes, rather than directly coding a numerical value throughout the code. One method is to create a stand-alone script that simply assigns values to all modem parameters and stores them to a file. Then, at the beginning of the transmit and receive functions the common set of parameters can be loaded from memory. The commands `save` and `load` can be used for this purpose.

- Use a *fixed-length packet* structure; a known packet length facilitates the receiver implementation. This can be accomplished in the conversion of text to symbols. For example, a text message can be either truncated or padded with spaces (ASCII 32) to achieve a fixed number of characters. The packet length will depend on the number of pilot symbols used in the header, the number of 8-bit ASCII characters in a text message, and the number, M, of symbols in the symbol constellation.

- The approximate location of a data packet must be detected within the segment of sound recorded at the receiver. If the detected window were to contain a large percentage of noise-only symbol-rate samples, then the constant modulus algorithm presented in Chapter 5 would not effectively compute the variance of packet symbols. The routine `packetdetect.m` is provided in Appendix C for this purpose and implements a simple energy detector; the routine operates on the fractionally sampled matched filter outputs.

The implementation of an acoustic demodulator and five optional exercises for this chapter are enumerated below.

Q7.1 (a) Implement and experimentally verify an acoustic demodulator with a spectral efficiency greater than 2; decode the first 20 ASCII characters. See items (i)–(vii) above for required elements in a brief laboratory report.

(b) **[Optional]** Modify your acoustic demodulator to decode the first 50 ASCII characters, with the last 30 characters being some fixed value (e.g., a space or underbar). For a transmitter, use either your own code or the mobile app. Plot the angles of the decoded symbols at the receiver. Briefly explain your observations.

(c) **[Optional]** Modify your digital modulation and digital demodulation routines from step (b) to implement and experimentally verify differential QPSK modulation. To this end, modify and replace your routines `char2psk.m` and `psk2char.m` to implement digital modulation and demodulation using 4-phase DPSK. Note that conjugation in Section 7.1.4 can be implemented usign the command `conj`.

(d) **[Optional]** Modify your digital modulation and digital demodulation routines from step (a) to implement and experimentally demonstrate a spectral efficiency greater than 2.

(e) **[Optional]** Modify your digital modulation and digital demodulation routines from step (a) to implement and experimentally demonstrate 32-QAM modulation.

(f) **[Optional]** Derive Equation 7.1.

7.3 Demonstration

Demonstrate your acoustic modem.

7.4 Summary

In Chapter 7, the physical layer communication design steps explored in Chapters 1 through 6 have been integrated to implement and verify a spectrally efficient modem operating at an acoustic carrier frequency. The implementation is designed for a flat-fading channel. The baseband processing translates seamlessly to narrowband communication using a radio-frequency channel, as explored in

Appendix B. The explorations introduced new commands summarized in Table 7.2.

Table 7.2: Summary of commands introduced in Chapter 7.

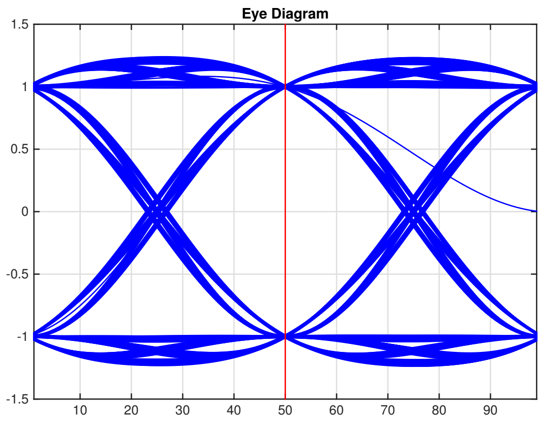
Command	Description
`save`	save workspace variables to file
`load`	load data from memory into a workspace
`path`	get or set search path
`addpath`	add a directory to search path
`packetdetect.m`	function to detect a received packet
`makepilots.m`	function to generate pilot sequence
`conj`	conjugation

CHAPTER 8

Frequency-Selective Fading

The frame synchronization and frequency recovery procedures presented in Chapters 5 and 6 are done without regard to any distortion due to the inter-symbol interference resulting from a *frequency-selective fading channel*. Whereas a flat-fading channel only imparts a gain and phase, a frequency-selective fading channel performs convolution, and thus may impart a gain and phase that varies as a function of frequency. The insertion of the channel's filter response, $\tilde{h}[k]$, in the baseband model implies that the net effect of the pulse shaping filter, $g_{\text{tx}}[k]$, the channel filter, $\tilde{h}[k]$, and the receiver's matched filter $g^*_{\text{tx}}[-k]$ is no longer an ideal Nyquist pulse. For this reason, the frequency-selective fading channel causes inter-symbol interference and is sometimes called an *ISI channel*. The ISI is illustrated, for BPSK signaling, in Figure 8.1.

In this chapter, frame timing and frequency recovery procedures are modified to operate more robustly in the presence of a frequency-selective fading channel. In addition, a linear equalizer is developed to combat the symbol distortion caused by an ISI channel. In Chapters 10 and 11, orthogonal frequency division multiplexing (OFDM) is presented as an alternative "frequency-domain" approach to communication over a frequency-selective fading channel; OFDM essentially reduces a frequency-selective fading channel to multiple, narrow-band, flat fading channels.



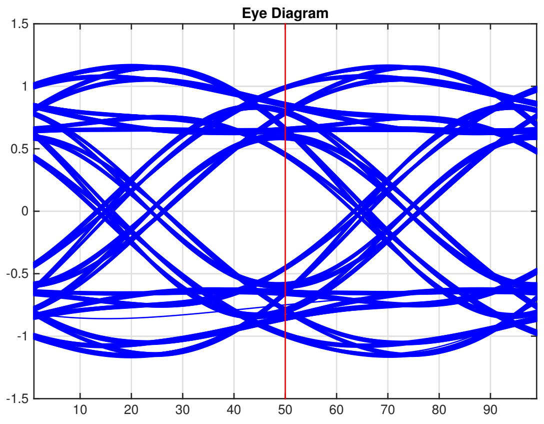


Figure 8.1: BPSK eye diagrams for no ISI (top) and ISI (bottom).

8.1 Background

8.1.1 Frame timing for ISI channels

For frame timing in the presence of a frequency-selective fading channel, we consider the approach proposed by Moose [24]. The method relies on a periodic pilot sequence and is used, for example, in IEEE 802.11a/g [15].

A length K FIR symbol-rate model of the channel impulse response,

$$\{\tilde{h}[n]\}_{n=0}^{K-1},$$

is called a K-tap channel model. The model may physically correspond, for example, to a multi-path channel. Then, the received signal after matched filtering, symbol timing recovery, and downsampling is given by

$$\tilde{y}[n] = e^{j2\pi f_\Delta nT} \sum_{i=0}^{K-1} \tilde{h}[i]\tilde{a}[n-i] + \tilde{w}[n], \tag{8.1}$$

where T is the symbol period and f_Δ is the frequency offset.

The Moose method is a self-referenced synchronization that avoids the dependence on the channel response, $\tilde{h}[n]$, by using a periodic pilot sequence; the length $2N_p$ pilot sequence is constructed by repeating a given marker code of length N_p.

The idea for self-referenced frame timing recovery is to look for a peak in the cross-correlation between the pair of received distorted marker sequences:

$$\widehat{\tau}_{\text{Moose}} = \arg\max_n \frac{\sum_{i=K-1}^{N_p-1} \tilde{y}[n+i+N_p]\tilde{y}^*[n+i]}{\sqrt{\sum_{i=K-1}^{N_p-1} |\tilde{y}[n+i+N_p]|^2}\sqrt{\sum_{i=K-1}^{N_p-1} |\tilde{y}[n+i]|^2}}. \tag{8.2}$$

Because the $(K-1)$ symbols prior to the start of pilots are unknown, the summation in the cross-correlation in Equation 8.2 runs from $(K-1)$ to (N_p-1), rather than 0 to (N_p-1).

8.1.2 Frequency recovery for ISI channels

The periodic pilot sequence is likewise employed in the Moose approach to provide an estimate of the frequency offset that is robust to ISI. Assume that, based on the timing recovery from Section 8.1.1

the received pilot sequence begins at index $n = 0$. Consider again the convolution in Equation 8.1 and rewrite with an advance of N_p samples:

$$\tilde{y}[n] = e^{j2\pi f_\Delta nT} \sum_{i=0}^{K-1} \tilde{h}[i]\tilde{a}[n-i] + \tilde{w}[n] \tag{8.3}$$

$$\tilde{y}[n+N_p] = e^{j2\pi f_\Delta (n+N_p)T} \sum_{i=0}^{K-1} \tilde{h}[i]\tilde{a}[n+N_p-i] + \tilde{w}[n+N_p]. \tag{8.4}$$

Using the repeated structure of the pilot sequence, we know

$$\tilde{a}[n+N_p-i] = \tilde{a}[n-i]. \tag{8.5}$$

From Equations 8.3, 8.4 and 8.5 we have, for indices $K-1 \le n \le N_p - 1$,

$$\begin{aligned} \tilde{y}[n+N_p] &= e^{j2\pi f_\Delta N_p T} e^{j2\pi f_\Delta nT} \sum_{i=0}^{K-1} \tilde{h}[i]\tilde{a}[n-i] + \tilde{w}[n+N_p] \\ &= e^{j2\pi f_\Delta N_p T}\tilde{y}[n] + \left\{\tilde{w}[n+N_p] - \tilde{w}[n]\right\}. \end{aligned} \tag{8.6}$$

Thus we see that $\tilde{y}[n+N_p]$ equals $e^{j2\pi f_\Delta N_p T}\tilde{y}[n]$ plus a zero-mean additive noise. That is, $2\pi f_\Delta N_p T$ is the mean angle of the second half of the received distorted pilots divided by the first half. Again, we omit use of the first $K-1$ samples because of their dependence on unknown symbols prior to the pilots. Thus, we have a sample mean to estimate the frequency offset:

$$\widehat{f}_\Delta = \frac{1/T}{2\pi N_p} \text{angle}\left\{ \frac{1}{(N_p - K)} \sum_{n=K-1}^{N_p-1} \frac{\tilde{y}[n+N_p]}{\tilde{y}[n]} \right\}. \tag{8.7}$$

By the 2π periodicity of the function $e^{j2\pi f_\Delta N_p T}$, the frequency offset f_Δ is unambiguous for

$$|f_\Delta| \le \frac{1}{2N_pT}.$$

A longer pilot sequence (i.e., larger N_p) improves the noise averaging in the estimator, but reduces the unambiguous range for the estimated frequency offset, $\widehat{f}_\Delta$. The Moose approach in Equation 8.7 provides an estimate of frequency offset that is robust to ISI.

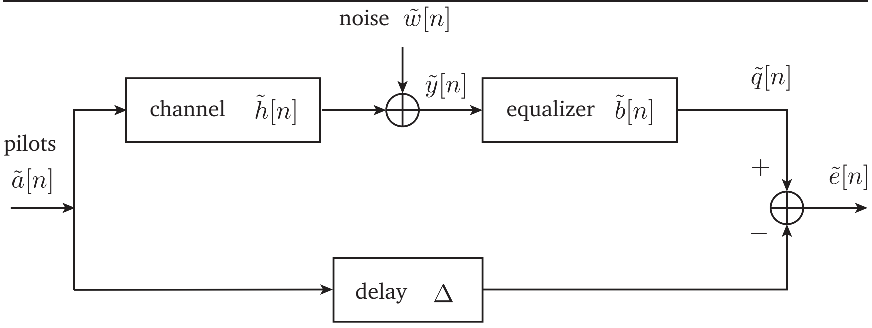


Figure 8.2: Linear equalizer for a frequency-selective fading channel

8.1.3 Linear equalizer for ISI channels

Consider using the known pilot sequence to learn the channel impulse response; a *linear least-squares equalization* approach is shown in the baseband model depicted in Figure 8.2. In the figure, $\tilde{y}[n]$ is the matched filter output downsampled to the symbol rate and post-processed for frequency recovery and frame timing. Without loss of generality, suppose the start of the pilot symbols is indexed by $n = 0$. The sequence $\tilde{q}[n]$ is the output of the equalization filter with impulse response $\{\tilde{b}[0], ..., \tilde{b}[M]\}$. The goal is to choose the filter so that the output matches a delayed version of the known pilot symbol sequence: $\tilde{q}[n] \approx \tilde{a}[n - \Delta]$. The delay is necessary because the equalizer is a causal filter.

The convolution of the FIR equalizer and the matched filter outputs produces $\tilde{q}[n]$,

$$\tilde{q}[n] = \sum_{k=0}^{M} \tilde{b}[k]\tilde{y}[n-k]. \tag{8.8}$$

For a pilot sequence of length $2N_p$, we seek to minimize the sum of squared error between the equalizer output and a delayed copy of the known pilot sequence. Replacing the delay, Δ, in the known pilot sequence, $\tilde{a}[n]$, by an advance in $\tilde{q}[n]$, we have the sum of squared errors, $J_\Delta(\mathbf{b})$, given by

$$J_\Delta(\mathbf{b}) = \sum_{n=0}^{2N_p-1} \left| \sum_{k=0}^{M} \tilde{b}[k]\tilde{y}[n+\Delta-k] - \tilde{a}[n] \right|^2 . \tag{8.9}$$

The least-squares solution is simplified by writing the expression in matrix-vector form:

$$\underbrace{\begin{bmatrix} \tilde{y}[n+\Delta] & \cdots & \tilde{y}[n+\Delta-M] \\ \tilde{y}[n+\Delta+1] & \cdots & \tilde{y}[n+\Delta-M+1] \\ \tilde{y}[n+\Delta+2] & \cdots & \tilde{y}[n+\Delta-M+2] \\ \vdots & \ddots & \vdots \\ \tilde{y}[n+\Delta+2N_p-1] & \cdots & \tilde{y}[n+\Delta+2N_p-M] \end{bmatrix}}_{\mathbf{Y}_\Delta} \underbrace{\begin{bmatrix} \tilde{b}[0] \\ \tilde{b}[1] \\ \vdots \\ \tilde{b}[M] \end{bmatrix}}_{\tilde{\mathbf{b}}} \approx \underbrace{\begin{bmatrix} \tilde{a}[0] \\ \tilde{a}[1] \\ \tilde{a}[2] \\ \vdots \\ \tilde{a}[2N_p-1] \end{bmatrix}}_{\tilde{\mathbf{a}}} \tag{8.10}$$

The equalizer that minimizes the sum of squared errors is given by

$$\tilde{\mathbf{b}}_{\Delta,\star} = \left(\mathbf{Y}_\Delta^H \mathbf{Y}_\Delta\right)^{-1} \mathbf{Y}_\Delta^H \tilde{\mathbf{a}}. \tag{8.11}$$

The delay, Δ, may be found by computing the squared error for each choice $0 \leq \Delta \leq M$ and selecting the value that results in the smallest sum of squared errors. For long pilot sequences, a single matrix inverse can be used to determine Δ, rather than exhaustive search [18]. An alternative of least-squares is a minimum mean squared error (MMSE) solution [31]; the MMSE solution balances noise rejection against the distortion caused by ISI and channel gain.

8.2 Explorations

In this lesson, students modify their acoustic modem simulator from Chapter 6 to implement frequency and timing recovery that is robust to a frequency-selective fading channel.

For the laboratory exploration, work through the three steps below to implement in simulation a bandpass modem for ISI channels. Recommendations for preparing a brief, descriptive laboratory report are included in the suggested steps below.

(a) Augment and revise your bandpass modem simulation code from Chapter 6 to implement the Moose algorithm for frame timing described in Equation 8.2 for an ISI channel.

At the bandpass signal, $s[k]$, insert simulated impairments for an ISI channel model with additive white complex Gaussian noise. Separate the non-zero taps of your fractionally-sampled $\tilde{h}[k]$ by at least L samples; and, limit the extent from first to last nonzero taps to LK or fewer samples.

Verify the operation of your bandpass modem in simulation. **Q8.1**

(b) Augment and revise your bandpass modem simulation code from Chapter 6 to implement the Moose algorithm for frequency recovery described in Equation 8.7 for an ISI channel. Note that the computation in Equation 8.7 can be implemented using these commands: `./`, `mean`, and `angle`.

Verify the operation of your bandpass modem in simulation. **Q8.2**

(c) To begin exploration of a least-squares equalizer, work through an illustrative low-order example. Let $K = 1$ be the order of the equalizer and $2N_p = 4$ be the length of the pilot sequence, $\tilde{a} = \{1, -1, 1, -1\}$. Suppose the true channel is given by the infinite impulse response system, **Q8.3**

$$\tilde{y}[n] = 0.5\tilde{a}[n-1] + 0.25\tilde{y}[n-1].$$

  (a) With initial condition $y[-1] = 1$, use recursion to determine the noise-free sequence $\tilde{y}[n]$ for $n = 0, 1, 2, 3$.

  (b) For $\Delta = 0$ and $\Delta = 1$, compute the equalizer, $\tilde{\mathbf{b}}_{\Delta,\star}$ using Equation 8.10 and Equation 8.11. Also, compute for each Δ the corresponding sum of squared errors; select the Δ yielding the smaller error.

(d) Augment and revise your bandpass modem simulation code from Chapter 6 to implement a linear least-squares equalizer to combat the impairment due to an ISI channel. To construct the matrices in Equation 8.10, the command `toeplitz` may be helpful; further, the least-squares solution in Equation 8.11 is easily computed via `b = pinv(Y)*a`, where `pinv` is an abbreviation for *Moore-Penrose psuedo-inverse.*

Verify the operation of your bandpass modem in simulation. **Q8.4**

(e) **[Optional]** Verify operation of your modem for an acoustic ISI channel. Construct a frequency-selective channel by positioning transmitter and receiver to obtain resolved multi-path interference. This is facilitated by selecting a small symbol interval and low carrier frequency.

8.3 Demonstration

Demonstrate your modem simulator for a five-tap ISI channel and other channel impairment variables of your choice.

8.4 Summary

In Chapter 8, a technique has been introduced to recover frame timing and a frequency offset in the presence of an ISI channel. In addition, a linear equalizer was developed to combat the effect of the channel impairment on data symbols. The explorations introduced new commands summarized in Table 8.1.

Table 8.1: Summary of commands introduced in Chapter 8.

Command	Description
`toeplitz`	construct a Toeplitz matrix
`pinv`	pseudo-inverse

CHAPTER 9

Channel Coding

Fundamental limits for reliable communication over noisy channels were characterized in the pioneering work of Claude Shannon [33]. Shannon gave answers to several fundamental questions: How does one measure the amount of information? Can information be transmitted with arbitrarily high accuracy in the presence of noise? If so, at what rate? What is the minimum rate required to convey an information source? What is the trade-off between reduced rate and approximation error?

In this chapter, we consider Shannon's surprising result that information can indeed be transmitted with arbitrarily high accuracy in the presence of noise, and we characterize the rate. A simple block code is implemented to gain some familiarity with channel coding.

9.1 Background

9.1.1 Channel capacity

Suppose that we encounter a low signal-to-noise ratio (SNR), but would like to achieve a low probability of error. Into the 1940s, the conventional wisdom was to repeat the same symbol multiple times and use a majority vote at the receiver. While this approach does reduce the probability of error, the rate of information bits per symbol reduces linearly with the number of repetitions. Indeed, to drive the error arbitrarily small, the bit rate must go to zero!

Using elementary arguments, Shannon showed that it is indeed possible to communicate at an arbitrarily low probability of error yet maintain a non-zero data rate, C, called the channel capacity.

For additive white Gaussian noise (AWGN), an ideal band-limited channel of two-sided bandwidth B has a capacity C given by

$$C = B \log_2 \left(1 + \frac{P}{BN_0}\right) \text{ bits per second} \tag{9.1}$$

where P watts is the received signal power and $N_0/2$ watts/Hz is the power density of the noise. What is the meaning of capacity? If the information rate R bits/s from the source is less than C, then it is theoretically possible to achieve reliable transmission through the channel by appropriate coding [27]. In contrast, if $R > C$, reliable transmission is not possible using any processing scheme.

For example, consider telephone line modems. To allow for multiplexing of many channels, signals are bandlimited to approximately 3300 Hz. A bandwidth of 3300 Hz and signal-to-noise ratio (SNR) of $P/N_0B = 31\,\text{dB}$ in Eqn. 9.1 yields a capacity of about 34,000 bits per second. This limit is approached by commercial modems, which work at 33.6 kbps, yielding an astounding bandwidth efficiency of over 10 data bits per second per Hertz. For fax modems, the International Telecommunication Union (ITU) V.34 standard is able to fully adapt to the telephone line quality by adjusting the carrier frequency, the data rate, and the transmitted power to fit the communication channel.

The decades since Shannon's pioneering work have witnessed a steady increase in the market demand for data transmission and the steady progress of engineering techniques pushing ever more closely to the fundamental theoretical limits [38].

9.1.2 Channel coding

Error correction codes add structured redundancy to an information sequence in order to detect and correct possible bit errors that occur during transmission. The basic idea of channel coding in digital communication is the use of parity bits, and the concept is very much analogous to spell checking. For example, a reader of English can identify and correct errors in the following text string:

With malice towark none, weth chariti for all

The three errors are identifiable because not all text strings are allowable words. That is, English has redundancy. A reader's guess at the correction may be described as making the fewest changes so that the corrected word is allowable. For example, by changing only one letter, “k,” the illegal string “towark” becomes the allowable word “toward.” Further, by considering longer text strings, a reader may use grammar and context to achieve more sophisticated error checking.

Table 9.1 illustrates application of the structured redundancy idea to binary strings. The left columns list the 16 possible information symbols formed using 4 bits; there is no redundancy. In the right columns, redundancy is introduced by appending three additional bits; hence, structured redundancy is inserted. This example is called a $(7, 4)$ block code because blocks of $k = 4$ bits are mapped to coded blocks of $n = 7$ bits. Block codes are perhaps the simplest of all error correcting codes to both understand and implement.

Table 9.1: An example $(7, 4)$ block code.

Information bits	Code word
0 0 0 0	0 0 0 0 0 0 0
1 0 0 0	1 0 0 0 1 1 0
0 1 0 0	0 1 0 0 1 0 1
1 1 0 0	1 1 0 0 0 1 1
0 0 1 0	0 0 1 0 0 1 1
1 0 1 0	1 0 1 0 1 0 1
0 1 1 0	0 1 1 0 1 1 0
1 1 1 0	1 1 1 0 0 0 0
0 0 0 1	0 0 0 1 1 1 1
1 0 0 1	1 0 0 1 0 0 1
0 1 0 1	0 1 0 1 0 1 0
1 1 0 1	1 1 0 1 1 0 0
0 0 1 1	0 0 1 1 1 0 0
1 0 1 1	1 0 1 1 0 1 0
0 1 1 1	0 1 1 1 0 0 1
1 1 1 1	1 1 1 1 1 1 1

An (n, k) code has *code rate* of k/n because the redundancy reduces the data transmission rate by the factor k/n. The implementation of the $(7, 4)$ code may be achieved by binary matrix multiplication using a matrix of four rows and seven columns. For example, consider the sixth row of Table 9.1.

$$\underbrace{\begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix}}_{k \text{ input bits}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}}_{k \times n \text{ generator matrix}} = \underbrace{\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}}_{n \text{ encoded bits}} \tag{9.2}$$

The arithmetic here, and throughout this chapter, is on the field of binary numbers, whereby addition is *exclusive OR* and multiplication is *logical AND*. Equivalently, the arithmetic is modulo-2. Note that only the last three columns of the *generator matrix*, G, in Equation 9.2 need to be stored and used to generate the three bits appended to the four bit information string. Alternatively, a block code may be efficiently implemented using a linear feedback shift register.

The appended bits may also be interpreted as *parity bits*. Let H^T be a n-by-$(n-k)$ binary matrix satisfying

$$GH^T = 0. \tag{9.3}$$

Because any allowable code word y is a binary sum of rows from G, a code word must satisfy

$$yH^T = 0. \tag{9.4}$$

Thus, H^T is called a *parity check matrix*. From the example above, let x_1, x_2, x_3, x_4 denote the four information bits and let c_1, c_2, c_3 denote the three bits appended by the $(7, 4)$ code. We have three parity check equations:

$$\begin{aligned} x_1 + x_2 + x_4 + c_1 &= 0 \\ x_1 + x_3 + x_4 + c_2 &= 0 \\ x_2 + x_3 + x_4 + c_3 &= 0. \end{aligned} \tag{9.5}$$

For this example, the parity check matrix is given by

$$H^T = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{9.6}$$

Decoding requires making the best guess of the closest allowable code word if the received word does not satisfy the parity checks. A closest allowable word requires a concept of distance. The *Hamming distance* between two binary strings of equal length is defined as the number of positions in which the two strings differ. For example, the two words

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

have a Hamming distance of 2, because the strings differ only in the fifth and sixth positions. For minimum distance decoding, the decoder compares the received code word with the $M = 2^k$ possible allowable transmitted code words and decides in favor of the code word that is closest in Hamming distance. This exhaustive search, while conceptually simple, is computationally inefficient. A more efficient method for this minimum distance decoding uses the parity check matrix, H^T, as seen in Section 9.1.3 below.

The minimum distance between any two allowable code words in the example above is $d = 3$. Therefore, the code can detect up to two errors, but three errors might result in an allowable code word. Additionally, the code permits correction of one error per code word, because only one unique allowable code word has Hamming distance 1 from a code word received with a single error. In general, an (n, k) code of minimum distance d can detect $d - 1$ errors and correct $\lfloor \frac{1}{2}(d-1) \rfloor$ errors, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x.

Block codes are simple to implement. Key characteristics of a block code are the minimum distance and the rate. A larger block size permits a higher minimum distance, but requires more delay. A higher rate code allows for transmission of more data bits per second, but can correct fewer errors.

9.1.3 Syndrome decoding

Syndrome decoding is an efficient alternative to exhaustive search for decoding linear block codes. Consider a received string of bits, $y = c + e$, that is a codeword, c, plus an error string, e. From

Equation 9.4, we apply the parity check matrix, H^T, to learn

$$\begin{aligned} yH^T &= cH^T + eH^T \\ &= 0 + eH^T \\ &= s \end{aligned} \tag{9.7}$$

where $s = eH^T$ is the $(n-k)$-bit *syndrome* of the error sequence, e. If the received string has zero or one error, then the syndrome can be used to return the correct codeword. For example, consider the $(7,4)$ block code described above. With a minimum distance of 3, the code can detect and correct all one-bit errors. For this case, the syndromes of all one-bit errors are given in Table 9.2. If no errors

Table 9.2: Single error syndromes for a $(7,4)$ block code.

Error Pattern $\times H^T$	Syndrome
$[1\ 0\ 0\ 0\ 0\ 0\ 0]H^T$	$[1\ 1\ 0]$
$[0\ 1\ 0\ 0\ 0\ 0\ 0]H^T$	$[1\ 0\ 1]$
$[0\ 0\ 1\ 0\ 0\ 0\ 0]H^T$	$[0\ 1\ 1]$
$[0\ 0\ 0\ 1\ 0\ 0\ 0]H^T$	$[1\ 1\ 1]$
$[0\ 0\ 0\ 0\ 1\ 0\ 0]H^T$	$[1\ 0\ 0]$
$[0\ 0\ 0\ 0\ 0\ 1\ 0]H^T$	$[0\ 1\ 0]$
$[0\ 0\ 0\ 0\ 0\ 0\ 1]H^T$	$[0\ 0\ 1]$

occured, then the syndrome is zero. If one error occurs, then the non-zero syndrome identifies the most likely error pattern, $\hat{e}$; this error vector is then added to y to yield the decoded string of bits. For two or more errors, the syndrome decoding may or may not return the correct codeword. For block codes with minimum distance greater than 3, syndrome decoding can be extended to detect and correct more than one bit error per block of n received bits.

If bit errors occur in bursts, then direct use of a channel code will not allow detection and correction of the cluster of errors. An effective method of dealing with burst errors is to *interleave* (shuffle) the coded data so that the channel with bursts of errors is approximately transformed into a channel with independent errors [35]. A simple approach is the block interleaver, which reads in coded bits row by row, then reads out bits to the modulator column by column.

The 67 years of coding theory since Shannon's seminal paper have witnessed remarkable progress in approximatey achieving channel capacity through use of more sophisticated codes requiring more complex processing for decoding. Clever encoding schemes provide a large Hamming distance between code words, short block lengths (hence low decoding latency), and simple decoding procedures. For example, an audio CD uses a cross interleaved Reed-Solomon code; 4G wireless systems use soft-decision decoding, turbo codes, and trellis-coded modulation.

9.2 Explorations

In this chapter, students implement and experimentally verify a $(7, 4)$ block code. The brief laboratory report should contain answers to the questions enumerated in the margins; subroutines for channel coding and decoding with the $(7, 4)$ block code should be appended.

(a) Block codes are often listed by triples, (n, k, d), where $(n-k)$ is the number of parity bits, k is the number of information bits, and d is the minimum distance between codewords. Compare and contrast the *rates* for three block codes: $(7, 4, 3)$, $(8, 4, 4)$, $(16, 5, 8)$. **Q9.1**

(b) Expand your BPSK simulation from Chapter 3 to include channel coding. Create function calls that perform $(7, 4)$ block coding and decoding. Repeat the bit error rate simulation from the demonstration in Section 3.3, but generate the plot for E_b/N_0 only on the range 0 to 10 dB. For comparison, overlay the theoretical curve for uncoded BPSK data, BER $= Q(\sqrt{2E_b/N_0})$, found in Chapter 3. For your report, present the BER curve and append code for your block coding and decoding functions. **Q9.2**

Although binary arithmetic is computationally efficient, a MATLAB implementation of yG using double precision real variables is simple to compute using modulo-2 arithmetic, `mod(y*G,2)`. For example, re-use random bit generation from Chapter 3:

```
k = 4; N = 250;
bits = round(rand(1,k*N));%random 0's and 1's
% use function from part 9(b)
codedbits = blockcode74(bits);
```

```
% convert bits to BPSK symbols
symbols = 2*codedbits-1;
```

Consider these four suggestions for using the syndrome to correctly decode for 0 or 1 bit error per block of n bits:

(i) Compute the length $(n-k)$ syndrome using the parity check matrix.

(ii) If the syndrome is zero, then return the first k bits as the message.

(iii) If the syndrome is not zero, use the syndrome table to index a single bit error. Flip the detected error and return the first k bits as the message. (Only the first k syndrome vectors in Table 9.2 need be used.)

(iv) In general (but not necessary for this $(7,4,3)$ block code): if there is no match of the syndrome to the first k syndrome vectors, then return the first k bits unchanged.

Q9.3 (c) Block codes are often listed by triples, (n,k,d), where $(n-k)$ is the number of parity bits, k is the number of information bits, and d is the minimum distance between codewords. The probability of error for a (n,k,d) linear block code is bounded by [21]

$$P \leq (2^k-1)Q\left(\sqrt{d\frac{k}{n}\frac{2E_b}{N_0}}\right), \tag{9.8}$$

whereas the probability of error for the uncoded case was found in Chapter 3 to be

$$P = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{9.9}$$

The *effective coding gain* is defined as the additional SNR required for the uncoded data to experience the same error rate as the coded data. By using the approximation $Q(x) \leq \exp(-x^2/2)$, we can find the comparison

$$\begin{aligned} P &\leq \exp\left\{-\frac{E_b}{N_0}\left(\frac{k}{n}d - \frac{k\ln 2}{E_b/N_0}\right)\right\} && \text{coded} \\ P &\leq \exp\left\{-\frac{E_b}{N_0}\right\} && \text{uncoded.} \end{aligned} \tag{9.10}$$

Thus, we conclude that the effective coding gain for an (n, k, d) code is approximately

$$\text{effective coding gain} \quad \approx \quad \frac{k}{n}d - \frac{k \ln 2}{E_b/N_0}. \tag{9.11}$$

Referring to Equation 9.11, plot the effective coding gain, in dB, versus E_b/N_0, for three block codes: $(7, 4, 3)$, $(8, 4, 4)$, $(16, 5, 8)$. Generate the plot for the range $E_b/N_0 \in [5, 20]$ dB. (Note: use $10 \log_{10}$ to express these ratios of power quantities in dB.)

(d) **[Optional]** Shannon's capacity implies a lower bound on E_b/N_0 below which arbitarily reliable communication cannot occur. In this exercise, you are asked to derive that bound [21]. To get started, normalize to $B = 1$ for simplicity and observe that $\ln(1 + x) \leq x$. Then, from Equation 9.1 we have

$$\begin{aligned} C &= \log_2(1 + \text{SNR}) \\ &= \frac{1}{\ln 2} \ln\left(1 + \text{SNR}\right) \\ &\leq \frac{1}{\ln 2} \text{SNR}. \end{aligned}$$

Continuing, use the definitions $\text{SNR} = E_s/N_0$ and $E_s = E_b R$, where E_s is energy per symbol, E_b is energy per bit, and R_b is bit rate. The derivation also requires that bit rate cannot exceed capacity: $R_b \leq C$. Express your answer in dB by computing $10 \log_{10}$ of the power quantity and rounding to two significant digits.

(e) **[Optional]** Reformat your software from Chapters 3 and 9 into separate, modular functions for the following steps: text to data bits; data bits to channel-coded bits; bits to symbols. Similarly, for the receiver processing reformat your software into separate, modular functions for the following steps: symbol detection; symbols to bits; channel decoding; bits to text.

9.3 Demonstration

Demonstrate execution of the codes developed in step (b).

9.4 Summary

In Chapter 9, the concept of channel capacity was introduced as a fundamental bound on the rate at which data can be transmitted through a noisy channel. Block coding was presented as a simple way to insert structured redundancy into a transmitted bit stream, in order to allow detection and correction of errors at the receiver. The explorations introduced a new command summarized in Table 9.3.

Table 9.3: Summary of commands introduced in Chapter 9.

Command	Description
`mod`	modulus after division

CHAPTER 10

Orthogonal Frequency Division Multiplexing Part I

In Chapter 8, we considered approaches to combat channel impairments for a *frequency-selective* fading channel, also known as an inter-symbol interference (ISI) channel. The frequency-selective fading channel is modeled as the convolution of a channel impulse response with the baseband message, in contrast to the simple complex-valued gain that serves to model a *flat* fading channel. In particular, in Chapter 8 an equalization filter was implemented to approximately undo the effects of the ISI channel; the equalization filter can be considered a "time-domain" approach. In contrast, in Chapter 10 and Chapter 11 we present a "frequency-domain" approach that can offer robust performance for a frequency-selective fading channel with lower complexity than the time-domain equalization. Conceptually, the frequency-domain approach is to divide the wide-band ISI channel into multiple non-interfering narrowband sub-channels, then exploit the simplicity of channel equalization when the sub-channel model is merely a complex-valued gain.

The explorations presented in Chapters 10 and 11 provide a tutorial for orthogonal frequency division multiplexing (OFDM) communication [3] and present a sequence of exercises guiding students to the implementation of OFDM in an acoustic modem. OFDM is used, for example, in 4G mobile communication.

In this chapter, we will derive a frequency-domain equalization technique; in the next chapter, we then show how this technique can be further modified to OFDM, where we decompose a multi-tap channel into a set of parallel single-tap channels.

10.1 Background

In this and the next chapter, we focus on the discrete-time baseband equivalent channel model,

$$\tilde{\mathbf{y}} = \tilde{\mathbf{h}} * \tilde{\mathbf{a}} + \tilde{\mathbf{w}}. \tag{10.1}$$

In Equation 10.1, $\tilde{\mathbf{a}}$ is a vector representing a sequence of complex-valued symbols and $\tilde{\mathbf{y}}$ represents the corresponding received symbols, after $\tilde{\mathbf{a}}$ goes through an equivalent channel with impulse response $\tilde{\mathbf{h}}$ and is corrupted with the white Gaussian noise $\tilde{\mathbf{w}}$. In this symbol-rate model, the convolution of the transmitter's pulse shaping filter and receiver's filter is presumed to yield an ideal Nyquist pulse. To ease notation, in this chapter and the next, we ignore the noise term $\tilde{\mathbf{w}}$ and drop the tilde signs[1].

Assume that the channel response $\mathbf{h}$ is a vector with size L, i.e., the channel has L taps. Then, we can rewrite Equation 10.1 as

$$y[n] = \textstyle\sum_{l=0}^{L-1} h[l]a[n-l]. \tag{10.2}$$

Note that, while implementing an acoustic modem in Chapters 1 through 7, the focus was on *narrowband* communication, where the bandwidth of the bandpass transmission signal is small enough (or equivalently, the symbol-rate is low enough) such that the channel can be effectively modeled using a single-tap response, resulting in the flat fading input-output relationship:

$$y[n] = h \times a[n]. \tag{10.3}$$

A nice property of this single-tap channel is that it is *memoryless*, meaning that the n^{th} output symbol, $y[n]$, is a function of only the n^{th} input symbol, $a[n]$. Therefore, to recover the input symbol $a[n]$, we can simply estimate the channel gain h and then divide the output symbol $y[n]$ by h.

Driven by ever-growing traffic demand, most wireless systems operate in the *wideband* regime, meaning that the symbol rate is sufficiently large so that different frequency components of the transmitted signal experience different channel gains. This wideband phenomenon is often referred to as *frequency selectivity* or *frequency*

[1]The tilde sign was employed in Chapter 1 through Chapter 8 to differentiate between real-valued and complex-valued signals.

selective fading. As a result, the single-tap model is not able to fully capture the physical channel effect for a wideband channel in which an output symbol is affected by more than one input symbol. For a wideband channel, the multi-tap convolution model of Equations 8.1 and 10.2 is required.

With a multi-tap channel, ISI occurs: each received symbol is the weighted sum of L input symbols, with the weights determined by the channel gains on each delayed tap, and each input symbol affects L output symbols, which is evident from Equation 10.2. As a result, and different from the single-tap case for a flat fading channel, the channel effect can no longer be equalized by simply dividing the received symbols by the estimated channel gain. The challenge is then to recover the input symbols from the ISI-corrupted output symbols. In contrast to the time-domain equalization filter explored in Chapter 8, in this chapter we present a low-complexity, robust alternative that can be interpreted as a frequency-domain equalization approach. For additional approaches to ISI channels, the reader is referred to maximum-likelihood sequence detection (e.g., the Viterbi algorithm) and direct-sequence spread spectrum (CDMA).

10.1.1 Frequency-domain equalization

Once again, consider the discrete-time wide-band multi-tap channel model

$$\mathbf{y} = \mathbf{h} * \mathbf{a}, \tag{10.4}$$

where $\mathbf{y}$ and $\mathbf{h}$ are complex-valued vectors of received symbols and channel impulse response samples, respectively. The challenge is to recover the symbol sequence $\mathbf{a}$. In other words, we seek to invert the convolution operation. The beauty of the discrete-time Fourier transform is to convert convolution of two sequences into a simple point-wise multiplication of their transforms. That is, let $\mathcal{F}_{DT}$ denote the discrete-time Fourier transform operator; then, the convolution $\mathbf{y} = \mathbf{h} * \mathbf{a}$ is equivalent to $\mathcal{F}_{DT}(y) = \mathcal{F}_{DT}(h) \times \mathcal{F}_{DT}(a)$, and the sequence $\mathbf{a}$ can be recovered via the inverse discrete-time Fourier transform of the ratio:

$$\mathbf{a} = \mathcal{F}_{DT}^{-1}\left(\frac{\mathcal{F}_{DT}(\mathbf{y})}{\mathcal{F}_{DT}(\mathbf{h})}\right). \tag{10.5}$$

For practical implementation, we must work with samples of the discrete-time Fourier transform operator, rather than directly with integration and multiplication of the continuous functions, such as $\mathcal{F}_{DT}(\mathbf{h})$, defined on the interval $[-\pi, \pi)$. Thus, we resort to the discrete Fourier transform (DFT) and employ fast algorithms (FFT) for its computation [16]. However, by virtue of working with frequency samples, the multiplication of frequency-domain samples becomes equivalent to the *circular convolution* (periodic convolution, or cyclic convolution) in the time domain. More precisely, assume that $\mathbf{h}$ and $\mathbf{a}$ are complex vectors with dimension L and N, respectively, with $L < N$. We have

$$\mathrm{DFT}_N(\mathbf{h} \circledast \mathbf{a}) = \mathrm{DFT}_N(\mathbf{h}) \times \mathrm{DFT}_N(\mathbf{a}), \tag{10.6}$$

where $\mathrm{DFT}(\cdot)_N$ is the N-point discrete Fourier transform. The circular convolution is denoted by $\circledast$ and defined as

$$(\mathbf{h} \circledast \mathbf{a})[n] = \textstyle\sum_{l=0}^{L-1} h[l]a[(n-l) \bmod N] \text{ with } 0 \le n \le N-1. \tag{10.7}$$

If the the circular convolution of $\mathbf{h}$ and $\mathbf{a}$ is known at the receiver, then $\mathbf{a}$ can be recovered by

$$\mathbf{a} = \mathrm{IDFT}_N\left(\frac{\mathrm{DFT}_N(\mathbf{h} \circledast \mathbf{a})}{\mathrm{DFT}_N(\mathbf{h})}\right). \tag{10.8}$$

We use uppercase letters to denote the DFT of the vector given by the corrsponding lowercase symbol; the defining summations for the DFT and the inverse DFT (IDFT) are given by

$$X_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n e^{-j2\pi kn/N} \tag{10.9}$$

$$x_n = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X_k e^{j2\pi kn/N}. \tag{10.10}$$

Given the channel model shown in Equation 10.2, the question is how to modify the sequence of transmitted symbols such that the linear convolution induced by the channel can yield the circular convolution as shown in Equation 10.7. To this end, let us first construct a sequence of symbols $\widehat{\mathbf{a}}$ with length $N + P$ by inserting a prefix with length P to $\mathbf{a}$:

$$\widehat{\mathbf{a}} = [\underbrace{a[N-P], \ldots, a[N-1]}_{\text{Cyclic Prefix: last } P \text{ symbols in } \mathbf{a}}, \underbrace{a[0], a[1], \ldots, a[N-1]}_{\mathbf{a}}]. \quad (10.11)$$

Since the prefix consists of the last P symbols in $\mathbf{a}$, it is often called the *cyclic prefix* (CP). After $\widehat{\mathbf{a}}$ is passed through the multi-tap channel with channel response $\mathbf{h}$, we can obtain

$$\widehat{y}[n] = (\widehat{\mathbf{a}} * \mathbf{h})[n] = \sum_{l=0}^{L-1} h[l]\widehat{a}[n-l]. \quad (10.12)$$

If $P \geq L-1$, then for any $0 \leq n \leq N-1$, the above equation yields

$$\begin{aligned}
\widehat{y}[n+P] = (\widehat{\mathbf{a}} * \mathbf{h})[n+P] &= \sum_{l=0}^{L-1} h[l]\widehat{a}[n+P-l] \\
&= \sum_{l=0}^{L-1} h[l] \begin{pmatrix} a[(n+P-l)-P] & \text{if } n+P-l \geq P \\ a[(n+P-l)+(N-P)] & \text{else} \end{pmatrix} \\
&= \sum_{l=0}^{L-1} h[l] \begin{pmatrix} a[n+N-l] & \text{if } n \leq l-1 \\ a[n-l] & \text{if } n \geq l \end{pmatrix} \\
&= \sum_{l=0}^{L-1} h[l]a[(n-l) \bmod N] \\
&= (\mathbf{h} \circledast \mathbf{a})[n], \quad (10.13)
\end{aligned}$$

where the second equality follows from the definition of $\widehat{\mathbf{a}}$.

Equation 10.13 reveals that by prepending a cyclic prefix with length no less than the number of channel taps to the data symbols, and passing the modified symbols to the channel, the channel output will contain a copy of the circular convolution of the channel response and the data symbols (see Figure 10.1 for an illustration). Then, at the receiver, with the knowledge of $\mathbf{h}$, the data symbols can be recovered via Equation 10.8.

Thus, the use of a cyclic prefix is a ploy at the transmitter so that a portion of the received symbol sequence matches the circular convolution, despite the physical channel behaving as a linear convolution. In order to obtain the circular convolution between the length-N data symbols and the length-L channel filter, the number of transmitted symbols (which includes both data symbols and the

$$\begin{array}{ll}
 & [1\ \ 2\ \ 3] \circledast [5\ \ 4\ \ 6\ \ 1\ \ 3\ \ 2] \\
\hline
= & \quad\ \ 1 \text{ x } [5\ \ 4\ \ 6\ \ 1\ \ 3\ \ 2] \\
 & + \ 2 \text{ x } [2\ \ 5\ \ 4\ \ 6\ \ 1\ \ 3] \\
 & + \ 3 \text{ x } [3\ \ 2\ \ 5\ \ 4\ \ 6\ \ 1] \\
\hdashline
= & \qquad\quad [18\ \ 20\ \ 29\ \ 25\ \ 23\ \ 11]
\end{array}$$

$$\begin{array}{ll}
 & [1\ \ 2\ \ 3] * [3\ \ 2\ \ 5\ \ 4\ \ 6\ \ 1\ \ 3\ \ 2] \\
\hline
= & \quad\ \ 1 \text{ x } [3\ \ 2\ \ 5\ \ 4\ \ 6\ \ 1\ \ 3\ \ 2] \\
 & + \ 2 \text{ x } \quad [3\ \ 2\ \ 5\ \ 4\ \ 6\ \ 1\ \ 3\ \ 2] \\
 & + \ 3 \text{ x } \qquad [3\ \ 2\ \ 5\ \ 4\ \ 6\ \ 1\ \ 3\ \ 2] \\
\hdashline
= & \qquad\quad [3\ \ 8\ \ 18\ \ 20\ \ 29\ \ 25\ \ 23\ \ 11\ \ 13\ \ 6]
\end{array}$$

Figure 10.1: An illustration for Equation 10.13 for $\mathbf{a} = [5, 4, 6, 1, 3, 2]$ and $\mathbf{h} = [1, 2, 3]$.

cyclic prefix symbols) must be at least $N + L - 1$, giving rise to a transmission overhead that is the fraction $(L-1)/(N+L-1)$ of the total transmitted symbols.

10.1.2 Frequency-domain interpretation

In order to gain more insight into the frequency-domain equalization technique, let us look at a specific example. Consider the following complex-valued symbol vector with 10 entries:

$$\mathbf{a}^k = [e^{j2\pi\frac{0\cdot k}{10}}, e^{j2\pi\frac{1\cdot k}{10}}, e^{j2\pi\frac{2\cdot k}{10}}, e^{j2\pi\frac{3\cdot k}{N}}, \ldots, e^{j2\pi\frac{9\cdot k}{10}}].$$

Observe that

$$\begin{aligned}
|\mathbf{a}^k| &= [1, 1, 1, 1, \ldots, 1], \\
\text{angle}(\mathbf{a}^k) &= \left[2\pi\frac{0\cdot k}{10}, 2\pi\frac{1\cdot k}{10}, 2\pi\frac{2\cdot k}{10}, 2\pi\frac{3\cdot k}{10}, \ldots, 2\pi\frac{9\cdot k}{10}\right],
\end{aligned}$$

which indicates that the vector $\mathbf{a}^k$ is a sequence of 10 samples taken from 10 equally-spaced positions along the unit circle of the complex plane; the sequence $\mathbf{a}^k$ steps, or "rotates," exactly k positions along the circle between consecutive samples. From Equation 10.9

we further know that $\text{DFT}_{10}(\mathbf{a}^k)$ is vector with all but the k^{th} entry being zero. In other words, the symbol sequence $\mathbf{a}^k$ only has the k^{th} frequency component.

According to Equation 10.6, which we restate below, if we transmit $\mathbf{a}^k$ on the channel with response $\mathbf{h}$, then the received signal also only has the k^{th} frequency component. More precisely, we will have the received signal being $H_k\mathbf{a}^k$, where H_k is the k^{th} element in $\text{DFT}_{10}(\mathbf{h})$.

$$\text{DFT}_N(\mathbf{h} \circledast \mathbf{a}) = \text{DFT}_N(\mathbf{h}) \times \text{DFT}_N(\mathbf{a}).$$

In Figure 10.2, we plot both $\text{DFT}(\mathbf{h})$ and the absolute value of the received the symbols after transmitting $\mathbf{a}^k$ for ten sample times with $k = 8$ through a channel with response $\mathbf{h}$. From the figure, we can verify that the gain in amplitude after transmitting $\mathbf{a}^8$ is $|H_8|$, which conforms with Equation 10.6.

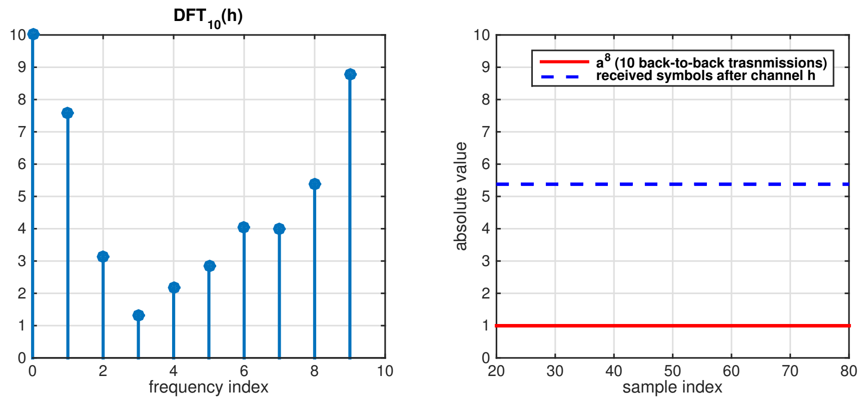


Figure 10.2: Absolute values of the received symbols after transmitting the symbol vector $\mathbf{a}^k$, $k = 8$, for ten time steps through a channel with response $\mathbf{h} = [4 + 1j, 3 - 1j, 2, 1]$.

10.2 Explorations

A lab report should provide answers to the following questions.

(a) Given the definition of DFT in Equation 10.9, verify the correctness of Equation 10.6. **Q10.1**

Q10.2 (b) Let $\mathbf{x}_1 = [1,2,3,4,5,6]$, $\mathbf{x}_2 = [5,2,4,3,1,6]$, and $\mathbf{h} = [1,2,1,1]$.

(i) Find $\mathbf{x}_1 \circledast \mathbf{h}$ and $\mathbf{x}_2 \circledast \mathbf{h}$.

(ii) Find $\mathbf{x}_1 * \mathbf{h}$ and $\mathbf{x}_2 * \mathbf{h}$.

(iii) Construct a vector $\mathbf{x}$ such that $\mathbf{x} * \mathbf{y}$ contains a copy of both $\mathbf{x}_1 \circledast \mathbf{h}$ and $\mathbf{x}_2 \circledast \mathbf{h}$.

(iv) What is the length of the shortest $\mathbf{x}$ that conforms to step (iii)?

Q10.3 (c) For $N = 6$, find $\text{DFT}_N(\mathbf{x})$ for $\mathbf{x} = [1,1,1,1,1,1]$. Repeat for $\mathbf{x} = [1,-1,1,-1,1,-1]$.

Q10.4 (d) Reproduce the result in Figure 10.2 with $k = 0, 1, \cdots, 9$. Explain what you observe.

10.3 Demonstration

Demonstrate the execution of the codes developed in step (d).

10.4 Summary

In Chapter 10, we introduced a frequency-domain channel equalization technique for combating frequency selectivity in wide-band wireless systems. We showed that by prepending a cyclic prefix with length larger than the channel memory onto the transmitted packet, we can equalize the channel gain on a per-frequency-tone basis with low-complexity FFT and IFFT operations. This chapter serves as a stepping stone for implementing OFDM in the next chapter. The explorations introduced new commands summarized in Table 10.1.

Table 10.1: Summary of commands introduced in Chapter 10.

Command	Description
`fft`	fast algorithm to compute the discrete Fourier transform
`ifft`	fast algorithm to compute the inverse discrete Fourier transform

CHAPTER 11

Orthogonal Frequency Division Multiplexing Part II

In the previous chapter, we introduced a method to equalize the effect of a multi-tap channel by prepending a cyclic prefix to the data symbols, converting the received symbols into the frequency domain, equalizing the channel effects on each frequency component, and then converting the modified frequency domain signal back to the time domain. This frequency domain equalization technique is summarized in Equation 11.1:

$$\begin{aligned} &\mathbf{a} \xrightarrow[\text{with length } P]{\text{add cyclic-prefix}} \widehat{\mathbf{a}} \xRightarrow[\text{with response } \mathbf{h}]{L\text{-tap channel}} \widehat{\mathbf{a}} * \mathbf{h} \xrightarrow[\text{received symbols}]{\text{discard first } P} (\mathbf{h} \circledast \mathbf{a}) \\ &(\mathbf{h} \circledast \mathbf{a}) \xrightarrow{N\text{-point DFT}} \mathbf{H} \cdot \mathbf{A} \xrightarrow[N\text{-point IDFT}]{\text{Divide } \mathbf{H}} \mathbf{a} \end{aligned} \tag{11.1}$$

Observe two characteristics of this technique. First, most of the computational complexity resides at the receiver side: the receiver needs to perform DFT and IDFT, while the transmitter just needs to prepend a cyclic prefix to the data symbols. Second, the receiver needs to know the channel impulse response. Therefore, two natural questions arise: (i) Can we turn the multi-tap channel into a set of parallel single-tap sub-channels so that the channel effect on each sub-channel can be estimated and equalized independently? (ii) Can we modify this technique so that the complexity is shared evenly between the sender and the receiver? We aim to address these two questions in this chapter.

11.1 Background

11.1.1 Orthogonal frequency division multiplexing

Let us simplify the notation for the procedures shown in Equation 11.1 as the following.

$$\mathbf{a} \xRightarrow{\text{equivalent channel}} \mathbf{H} \cdot \mathbf{A}$$

Note that after $\mathbf{a}$ is sent through the equivalent channel, the receiver obtains the *point-wise product* of the N-point channel frequency response and the N-point DFT of $\mathbf{a}$. This point-wise product is the key to realizing that it is possible to convert the multi-tap channel into N single-tap channels, with each channel corresponding to a term in the point-wise product. The idea is the following: instead of sending $\mathbf{a}$, let us send the N-point IDFT version of $\mathbf{a}$, denoted as $\mathbf{b}$, through the equivalent channel, which results in the procedures in Equation 11.2.

$$\mathbf{a} \xrightarrow{N\text{-point IDFT}} \mathbf{b} \xRightarrow{\text{equivalent channel}} \mathbf{H} \cdot \mathbf{B} = \mathbf{H} \cdot \mathbf{a} \tag{11.2}$$

The last equation holds because $\mathbf{b} = \text{IDFT}_N(\mathbf{a})$, which leads to $\mathbf{B} = \text{DFT}_N(\mathbf{b}) = \text{DFT}_N\left(\text{IDFT}_N(\mathbf{a})\right) = \mathbf{a}$. By substituting the equivalent channel with the procedures in Equation 11.1, we have the complete procedure:

$$\begin{gathered}\mathbf{a} \xrightarrow{N\text{-point IDFT}} \mathbf{b} \xrightarrow[\text{with length } P]{\text{add cyclic-prefix}} \widehat{\mathbf{b}} \xRightarrow[\text{with response } \mathbf{h}]{L\text{-tap channel}} \widehat{\mathbf{b}} * \mathbf{h} \\ \widehat{\mathbf{b}} * \mathbf{h} \xrightarrow[\text{received symbols}]{\text{discard first } P} (\mathbf{h} \circledast \mathbf{b}) \xrightarrow{N\text{-point DFT}} \mathbf{H} \cdot \mathbf{a}\end{gathered} \tag{11.3}$$

This small modification has two important implications. First, the multi-tap channel is successfully decomposed into N single-tap channels. Specifically, for any data symbol $a[n]$ in $\mathbf{a}$, the receiver will obtain a scaled version of it as $H[n]a[n]$, where $H[n]$ is the n^{th} element of the N-point DFT of the channel response $\mathbf{h}$. Second, the computational burden is evenly shared between the transmitter and the receiver: the transmitter must compute one IDFT, and the receiver must compute one DFT. The complexity of DFT and IDFT, using the FFT and IFFT algorithms, is $\mathcal{O}\left(N \log_2 N\right)$.

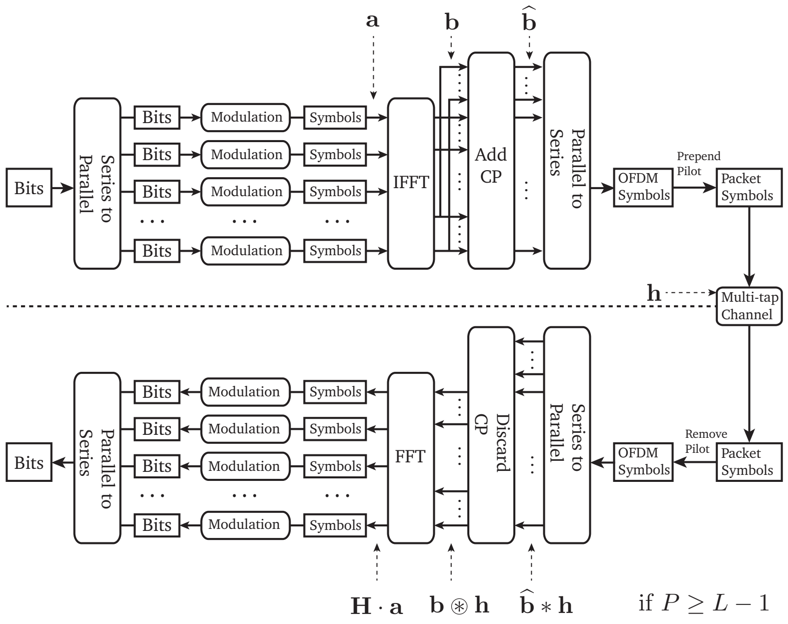


Figure 11.1: Block diagram of an OFDM system.

To gain insight into the first implication, let us consider the case when a series of N-dimensional symbol vectors, denoted as $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \ldots,$ is sent. After the procedures in Equation 11.3, the receiver will obtain

$$\begin{array}{llll} H[0]a_1[0] & H[0]a_2[0] & H[0]a_3[0] & \ldots \\ H[1]a_1[1] & H[1]a_2[1] & H[1]a_3[1] & \ldots \\ \vdots & \vdots & \vdots & \ldots \\ H[N-1]a_1[N-1] & H[N-1]a_2[N-1] & H[N-1]a_3[N-1] & \ldots \end{array}$$

From the above result, we can observe that the n^{th} symbol in each symbol vector, $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \cdots$, experiences the same one-tap channel gain $H[n]$. In other words, the procedure in Equation 11.3 effectively converts the multi-tap channel into N-single tap channels, where for any symbol vector $\mathbf{a}$, the n^{th} symbol experiences a channel with gain $H[n]$.

Because the transmitted symbol vector, $\mathbf{b}$, is the IDFT of the data symbol vector, $\mathbf{a}$, we know that the data symbols in $\mathbf{a}$ are the

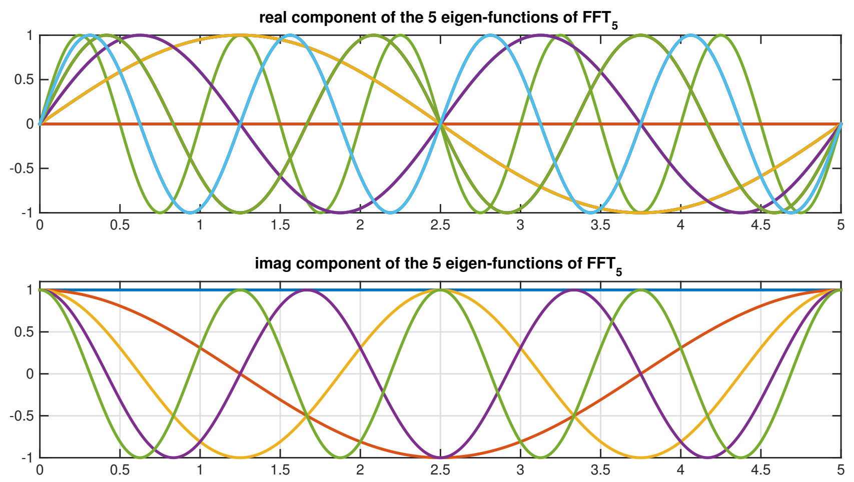


Figure 11.2: The time domain waves of the five frequency subcarriers in DFT_5.

frequency domain representation of the transmitted symbols. In this sense, OFDM transmission may be interpreted as directly modulating data as a gain and phase on each of N orthogonal carrier tones or "subcarrier frequencies," and the total transmitted waveform is the sum of these quadrature amplitude modulated subcarrier signals. The channel gain on the n^{th} frequency subcarrier is equal to $H[n]$. To illustrate, we plot in Figure 11.2 the five time-domain signals (eigen-functions) that correspond to the five frequency subcarriers if IFFT_5 and FFT_5 are used in the OFDM scheme described in Figure 11.1. It is easy to verify that these five waves are orthogonal. For any input vector $[a[0], a[1], a[2], a[3], a[4]]$ to the IFFT_5 block, the five values $a[0], a[1], a[2], a[3]$, and $a[4]$ are modulated directly on these five orthogonal waves.

The complete OFDM system is illustrated in Figure 11.1. In Figure 11.3, we highlight the process in the first subcarrier. In an OFDM system, the symbol vectors are transmitted in blocks of many symbols concatenated with no delay in between. Since each OFDM symbol vector corresponds to a group of data symbols modulated on orthogonal frequencies, the back-to-back transmission results in

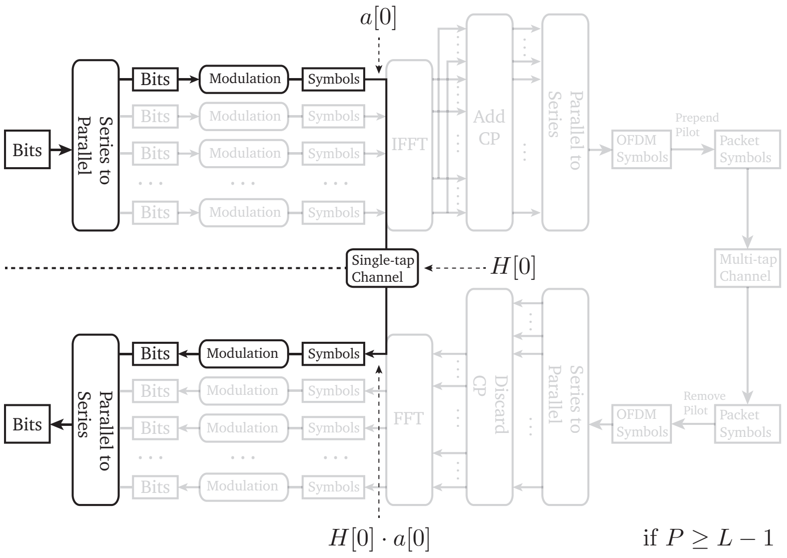


Figure 11.3: Block diagram of a single sub-channel.

a time-frequency resource grid, with each grid entry corresponding to a single complex-valued data symbol. This time-frequency grid is depicted in Figure 11.4. Each row in the time-frequency resource grid represents a frequency subcarrier, while each column represents the time duration for a single OFDM symbol vector.

Note that the cyclic prefix at the beginning of each OFDM symbol vector not only ensures that the data symbols are modulated onto orthogonal subcarrier frequencies (in that it provides a way to implement circular convolution through the channel), but also serves as a guard interval in time that prevents the transmission of the previous OFDM symbol vector from interfering with the current OFDM symbol vector.

Denote the symbol sampling period as T_{Sym} second and the time for a complete OFDM symbol vector as T_{OFDM} second. Because one symbol is transmitted per subcarrier per T_{OFDM} second, we know that the bandwidth of a subcarrier is $1/T_{\text{OFDM}}$ Hz. On the other hand, the total bandwidth is $1/T_{\text{Sym}}$ Hz. Therefore, the total number of subcarriers, including those devoted to the cyclic prefix symbols, is roughly $1/T_{\text{Sym}}/(1/T_{\text{OFDM}}) = T_{\text{OFDM}}/T_{\text{Sym}}$, which conforms

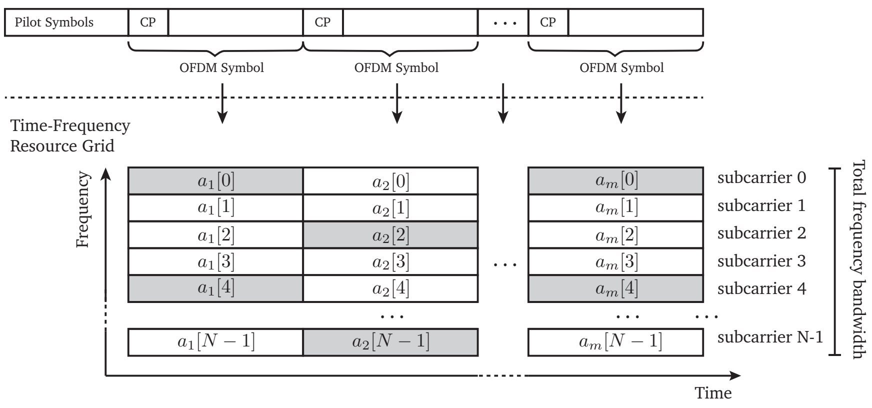

Pilot Symbols
CP
CP
CP
OFDM Symbol
OFDM Symbol
OFDM Symbol
Time-Frequency Resource Grid
Frequency
$a_1[0]$ $a_1[1]$ $a_1[2]$ $a_1[3]$ $a_1[4]$ $a_1[N-1]$
$a_2[0]$ $a_2[1]$ $a_2[2]$ $a_2[3]$ $a_2[4]$ $a_2[N-1]$
$a_m[0]$ $a_m[1]$ $a_m[2]$ $a_m[3]$ $a_m[4]$ $a_m[N-1]$
subcarrier 0
subcarrier 1
subcarrier 2
subcarrier 3
subcarrier 4
subcarrier N-1
Total frequency bandwidth
Time

Figure 11.4: OFDM time-frequency resource grid. The shaded grid entries correspond to a possible selection of reference symbols, which are used by the receiver as known pilot symobls for channel estimation. The channel gain for the remaining grid positions can be estimated by interpolation.

with our time-domain observation. For a fixed OFDM symbol length T_{OFDM} second, the number of subcarriers may be varied based on the total bandwidth available for transmission, leading to a spectrally flexible system design.

11.1.2 Channel estimation

To estimate the channel gain on different subcarriers, one can send both data symbols and reference symbols in the time-frequency grid. Prior to the start of the transmission, the transmitter and the receiver agree on the values and locations of the reference symbols so that the channel gain can be measured at the reference symbol locations. The channel gain at the data symbol locations can then by estimated by interpolating the complex-valued gains of the reference symbols locations. Figure 11.4 shows a possible selection of reference symbol positions, which are marked as shaded rectangles in the time-frequency grid. The reference symbol locations are typically designed to stagger across both time and frequency, providing robustness against channel fading in both time and frequency.

One can also eliminate the need for channel estimation by applying to each subcarrier a differential modulation/demodulation technique (such as DBPSK, DQPSK from Section 7.1.4), which relies on the relative phase to encode/decode information bits. The price to pay for this simplicity, however, is the overhead caused by the transmission of the initial reference symbol on each subcarrier and a slight increase in the symbol error rate.

11.2 Explorations

A lab report should provide answers to the following questions.

(a) Verify that the OFDM transceiver architecture described in Figure 11.1 is captured by the following equation **Q11.1**

$$\mathbf{a} \xrightarrow{N\text{-point IDFT}} \mathbf{b} \xrightarrow[\text{with length } P]{\text{add cyclic-prefix}} \widehat{\mathbf{b}} \overset{L\text{-tap channel}}{\underset{\text{with response } \mathbf{h}}{\Longrightarrow}} \widehat{\mathbf{b}} * \mathbf{h}$$

$$\widehat{\mathbf{b}} * \mathbf{h} \xrightarrow[\text{received symbols}]{\text{discard first } P} (\mathbf{h} \circledast \mathbf{b}) \xrightarrow{N\text{-point DFT}} \mathbf{H} \cdot \mathbf{a}$$

(b) Write and test subroutines for each step in the above equation. **Q11.2**

(c) Write a function that combines the subroutines from step (b) to create a baseband simulator for OFDM. The example .m file header below provides defining requirements for the function. **Q11.3**

```
[Output_symbol_matrix]=...
    OFDM_Transceiver(Input_symbol_stream,M,N,h,P)
% --------- Input ---------
% Input_symbol_stream is a row vector of length M*N
% N is the IFFT and FFT size.
% P is the cyclic prefix length
% h is the channel impulse response in vector form
% --------- Output ---------
% Output is a M by N matrix, with the n-th row
% being the output of the n-th subcarrier.
% --------- Requirement ---------
% 'conv' command is invoked only once.
% Calculation of the FFT of h is not allowed.
```

(d) Test your function with the following input: **Q11.4**

```
M=100;
N=1024;
P=10;
h=[1,2,2*1j,-1j];
Input_symbol_stream=ones(1,M*N);
```

Explain what you observe in the output.

(e) **[Optional]** Use the baseband simulator from step (d) and the acoustic modem from Chapter 7 as a foundation from which to construct an acoustic transmitter and receiver using orthogonal frequency division multiplexing. At the transmitter, the waveform $\widehat{b}$ will serve as input to the pulse shaping filter. At the receiver, pilot symbols will be used to estimate a complex-valued channel gain for each time-frequency grid point; for a channel that is static across time, a simple strategy is to place a known pilot symbol at every location in the first column of the grid shown in Figure 11.4. In an attempt to create an ISI channel, rather than a narrow-band flat fading channel, select a low-carrier frequency and short symbol interval. The resulting bandpass acoustic signal should haves a large *fractional bandwidth*, defined as the bandwidth divided by the carrier frequency.

11.3 Demonstration

Demonstrate the execution of the codes developed in step (c).

11.4 Summary

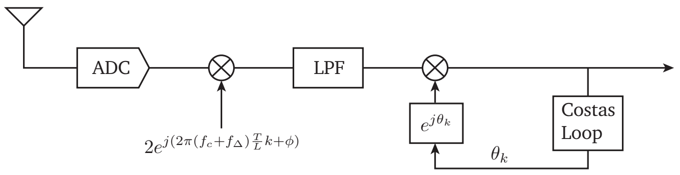
In Chapter 11, the orthogonal frequency division multiplexing scheme was derived. ODFM is the fundamental building block in almost all wide-band wireless systems, and the scheme effectively converts a wide-band multi-tap channel into multiple orthogonal single-tap sub-channels. Explorations guided students through implementation of an OFDM transceiver.

CHAPTER 12

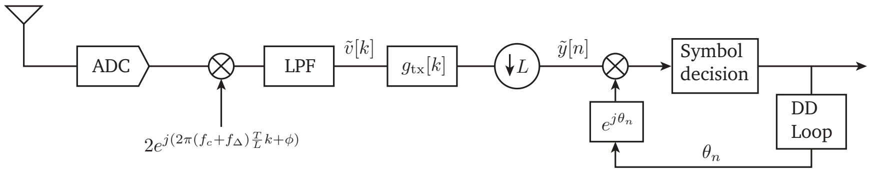
Adaptive Processing

In previous chapters, decoding of a received signal was accomplished via block processing; the long stream of input samples was stored to memory and accessible for decoding any particular symbol embedded in the stream. Channel impairments, such as frequency and phase offsets, were assumed static throughout the duration of the signal. In contrast, this chapter introduces *adaptive* processing techniques that sequentially process input samples and can provide time-varying estimates of channel impairments. In comparison to block processing, adaptive techniques can lessen memory requirements, reduce processing latency, and, importantly, track time-varying channel conditions.

In order to present representative examples of adaptive processing, this chapter guides students to implement feedback loops for carrier phase recovery. First, for the special case of a real-valued baseband message, the *Costas loop* is considered. Second, *decision-directed* phase recovery is presented, whereby the error between a received value and the nearest symbol is used to track the carrier phase. In effect, the decoding decisions are used as surrogates for the true symbols, and the decisions direct the adaptation of the unknown channel parameters. Remarkably, the strategy is very effective for non-zero error rates less than 0.01.



(a) Costas Loop



(b) Decision-directed phase recovery

Figure 12.1: Feedback loops for adaptive carrier phase recovery: (a) Costas loop; (b) decision-directed phase recovery.

12.1 Background

12.1.1 Carrier phase errors

As explored in Chapter 2 and Chapter 5, imperfections in the receiver oscillator create a phase mismatch relative to the transmitter. For phase offset, ϕ, and frequency offset, f_Δ, the time-varying phase error in the fractionally-sampled demodulated signal, $\tilde{v}[k]$, is $\phi + 2\pi f_\Delta \frac{T}{L} k$, modulo 2π. Here, T is the symbol interval, L is the upsampling factor, and k is the time index for the fractionally-sampled signal. Thus, a small frequency offset, f_Δ, causes a phase error that grows over time; similarly, *frequency drift* in the demodulation reference signal results in a time-varying offset.

The processing structures considered in this chapter for addressing phase errors are illustrated in Figure 12.1. In Figure 12.1(a), the Costas loop operates on the fractionally-sampled demodulated signal, $\tilde{v}[k]$, to track phase errors due to f_Δ and ϕ. In Figure 12.1(b), the decision-directed phase tracker operates on the symbol decisions, $\widehat{a}[n]$, to correct the phase of the symbol-rate matched filter outputs, $\tilde{y}[n]$. In both cases, *feedback* is employed, whereby output values determine future estimates of the time-varying phase correction, θ_k or θ_n.

12.1.2 Costas loop

The Costas loop [4] is a feedback mechanism for generating the correct carrier phase in a double-sideband suppressed carrier signal. The basic principle is that for a real-valued message signal, all energy in a QAM receiver should be in the in-phase channel, with zero signal energy appearing in the quadrature channel. While the approach originally employed a voltage-controlled oscillator (VCO) to generate the reference carrier signal at the receiver, here we present a software-based, sampled data version. Let the basedband message be a real-valued signal, $m[k]$. From Equation 2.12, demodulation at the receiver with a phase offset $\phi_k = 2\pi f_\Delta \frac{T}{L}k + \phi$ results in the IQ signal, $\tilde{v}[k]$

$$\tilde{v}[k] = m[k]\cos(\phi_k) - jm[k]\sin(\phi_k). \tag{12.1}$$

We seek a phase correction, θ_k, so that zero signal energy appears in the quadrature channel. To this end, we define a cost function, $J_C(\theta_k)$, and adaptively adjust θ_k to minimize the cost. Specifically, let the cost be the mean squared value of the quadrature channel samples. Define $\tilde{v}[k] = I_k + jQ_k$ to be the demoduated samples at the receiver; then, the mean squared value of the quadrature channel, using phase correction θ_k, is given by

$$\begin{aligned} J_C(\theta_k) &= \mathrm{E}\left\{ |\,\mathrm{Im}\left[\tilde{v}[k]e^{j\theta_k}\right]|^2 \right\} \\ &= \mathrm{E}\left\{ |\,\mathrm{Im}\left[(I_k + jQ_k)(\cos\theta_k + j\sin\theta_k)\right]|^2 \right\} \\ &= \mathrm{E}\left\{ |I_k \sin\theta_k + Q_k\cos\theta_k|^2 \right\}. \end{aligned} \tag{12.2}$$

Above, $\mathrm{E}\{\cdot\}$ denotes *expectation*, i.e., statistical average. We adopt a gradient descent method, taking a small step size, μ, in the downhill direction of the cost, as given by the partial derivative.

$$\frac{\partial}{\partial\theta} J_C(\theta) = \mathrm{E}\left\{ \frac{\partial}{\partial\theta} |I_k\sin\theta_k + Q_k\cos\theta_k|^2 \right\} \tag{12.3}$$

$$= \mathrm{E}\left\{ 2(I_k\sin\theta_k + Q_k\cos\theta_k)(I_k\cos\theta_k - Q_k\sin\theta_k) \right\}. \tag{12.4}$$

By virtue of the averaging effect of many small steps, we omit the expectation operator, E, to arrive at the following gradient descent update equation

$$\theta_{k+1} = \theta_k - 2\mu\,\mathrm{Im}\{\tilde{v}[k]e^{j\theta_k}\}\,\mathrm{Re}\{\tilde{v}[k]e^{j\theta_k}\}. \tag{12.5}$$

The same update equation can equivalently be derived by using gradient ascent to maximize the average energy in the in-phase channel.

Rather than using gradient descent, an alternative update strategy may be derived by postulating an error signal and using a *tracking loop*, depicted in Figure 12.2. The frequency and phase offsets can be considered a phase disturbance,

$$d_k = 2\pi f_\Delta \frac{T}{L} k + \phi. \tag{12.6}$$

Thus, we seek a tracking loop to provide zero steady-state error for a disturbance that contains a ramp, $\{2\pi f_\Delta T/L\}k$, and a step, ϕ. To this end, we use a Type 2 controller with two poles at $z = 1$ [11],

$$C(z) = \frac{(K_1 + K_2)z - K_1}{(z-1)^2}. \tag{12.7}$$

To specify an error signal in the tracking loop, observe from Equation 12.5

$$\begin{aligned}
& \mathrm{Im}\{\tilde{v}[k]e^{j\theta_k}\}\,\mathrm{Re}\{\tilde{v}[k]e^{j\theta_k}\} \\
= \; & \mathrm{Im}\left\{m[k]e^{-jd_k}e^{j\theta_k}\right\}\mathrm{Re}\left\{m[k]e^{-jd_k}e^{j\theta_k}\right\} \\
= \; & |m[k]|^2 \sin(\theta_k - d_k)\cos(\theta_k - d_k) \\
= \; & |m[k]|^2 \sin(2[\theta_k - d_k]). \qquad (12.8)
\end{aligned}$$

Thus, noting that $\sin(2[\theta_k - d_k]) \approx 2(\theta_k - d_k)$ for small phase errors, we define the error signal to be

$$e[k] = \frac{1}{|\tilde{v}[k]|^2}\mathrm{Im}\{\tilde{v}[k]e^{j\theta_k}\}\,\mathrm{Re}\{\tilde{v}[k]e^{j\theta_k}\}. \tag{12.9}$$

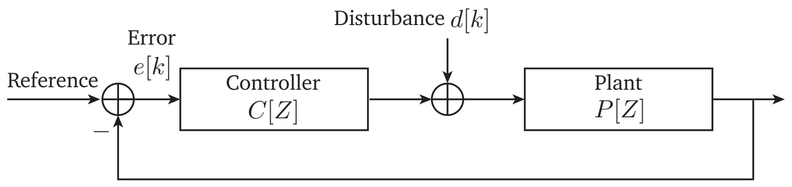


Figure 12.2: Block diagram for tracking interpretation of adaptive carrier phase recovery.

For discrete-time implementation of the second-order loop filter, we introduce a state variable, x_k, to obtain the update equations:

$$\theta_{k+1} = \theta_k + K_1 e_k + x_k \tag{12.10}$$
$$x_{k+1} = x_k + K_2 e_k. \tag{12.11}$$

The controller gains should be selected with ratio K_1/K_2 approximately 50 to 100 for a suitable transient response [37, p. 253].

For small phase errors, we adopt the linearization

$$\sin(2[\theta_k - \phi_k]) \approx 2(\theta_k - \phi_k), \tag{12.12}$$

yielding $P(z) = -2$ in Figure 12.2. Then, the closed-loop transfer function from the disturbance, d_k, to the tracking error, $e[k]$, is found using z-transforms:

$$\frac{E(z)}{D(z)} = \frac{-P(z)}{1+P(z)C(z)} = \frac{2(z-1)^2}{z^2 - 2(K_a + K_2 + 1)z + (2K_1 + 1)}. \tag{12.13}$$

12.1.3 Decision-directed phase tracking

Next, we consider carrier phase tracking using a decision-directed approach, working with symbol rate complex-valued samples of the matched filter output, $\tilde{y}[n]$. Define a cost function to be the mean squared error between the phase corrected received sample, $\tilde{y}[n]e^{j\theta_n}$, and the true transmitted symbol, $\tilde{a}[n]$. In the decision-directed approach, the symbol decisions, $\widehat{a}[n]$, are used as stand-in values for the true data symbols. Thus, the cost is

$$J_D(\theta_k) = \mathrm{E}\left\{|\widehat{a}[n] - e^{j\theta_n}\tilde{y}[n]|^2\right\}.$$

To perform gradient descent, we need the partial derivative of this cost with respect to the phase correction, θ_k.

$$\begin{aligned}
\frac{\partial}{\partial\theta} J_D(\theta) &= \mathrm{E}\left\{\frac{\partial}{\partial\theta}\left\{\left(\widehat{a}[n] - e^{j\theta_n}\tilde{y}[n]\right)^* \left(\widehat{a}[n] - e^{j\theta_n}\tilde{y}[n]\right)\right\}\right\} \\
&= \mathrm{E}\left\{j\left(\widehat{a}[n]\tilde{y}^*[n]e^{-j\theta} - \widehat{a}^*[n]\tilde{y}[n]e^{j\theta}\right)\right\} \\
&= -2\,\mathrm{E}\left\{\mathrm{Im}\left(\widehat{a}[n]\tilde{y}^*[n]e^{-j\theta}\right)\right\}.
\end{aligned} \tag{12.14}$$

Here, the superscript * denotes complex conjugation. Thus, the gradient descent update, with step size μ, is given by

$$\theta_{k+1} = \theta_k + 2\mu\,\mathrm{Im}\left\{\widehat{a}[n]\tilde{y}^*[n]e^{-j\theta}\right\}. \tag{12.15}$$

As in the previous subsection, we can use a tracking loop in place of gradient descent. To define an error signal, we observe

$$\begin{aligned}\mathrm{Im}\left(\widehat{a}[n]\tilde{y}^*[n]e^{-j\theta}\right) &= \mathrm{Im}\left(|\widehat{a}[n]|\,|\tilde{y}[n]|e^{j(\alpha_n-\beta_n)}\right)\\ &= |\widehat{a}[n]|\,|\tilde{y}[n]|\sin(\alpha_n-\beta_n), \qquad (12.16)\end{aligned}$$

where α_n is the phase angle of the symbol $\widehat{a}[n]$ and β_n is the phase angle of the phase-corrected sample, $\tilde{y}[n]e^{j\theta_n}$. Thus, we have the relation

$$\sin(\alpha_n-\beta_n) = \frac{\mathrm{Im}\left\{\widehat{a}[n]\tilde{y}^*[n]e^{-j\theta}\right\}}{|\widehat{a}[n]|\,|\tilde{y}[n]|} \qquad (12.17)$$

For small angle errors, we invoke the approximation $\sin\alpha = \alpha$ to define an error signal as

$$e_k = \sin(\alpha_n-\beta_n) = \frac{\mathrm{Im}\left\{\widehat{a}[n]\tilde{y}^*[n]e^{-j\theta}\right\}}{|\widehat{a}[n]|\,|\tilde{y}[n]|}. \qquad (12.18)$$

Armed with this error signal in the decision-directed phase recovery framework, the tracking loop update equations again take the form of Equations 12.10–12.11; these update equations can be written as a second-order difference equation:

$$\theta_{k+1} = 2\theta_k - \theta_{k-1} + K_1 e_k + (K_2 - K_1)e_{k-1}. \qquad (12.19)$$

12.2 Explorations

In this chapter, students implement and experimentally verify adaptive carrier phase recovery algorithms. A lab report should provide answers to the questions enumerated in the page margins and present commented code, as a subroutine, for the adaptive computations.

Q12.1 (a) Derive Equation 12.5 from Equation 12.4.

Q12.2 (b) Create code to implement decision-directed carrier phase tracking, using the gradient descent approach in Equation 12.15. Apply the adaptive technique as post-processing to receiver outputs obtained in the simulator constructed at Chapter 6. Due to mis-estimation of the carrier frequency, the received symbol phase will drift and eventually cause errors in a long decoded text string. To view the action of the tracker in your modem simulator, display the decoded text message both before and after the decision-directed tracker; also, plot both the symbol angle error before tracking and the tracker output signal, θ_k, from Equation 12.15. An example snippet of code for plotting is given below:

```
figure;
plot(angle(symbols./symbols_est)*180/pi,'r');
hold on;grid;ylabel('degrees');
plot(angle(exp(1i*thetaGD))*180/pi);
legend('OpenLoop Error','\theta_{grad}[k]',...
    'location','best')
```

Choose a text string length greater than 100. Because different pseudo-random noise trials will cause different frequency estimation errors, run your simulator several times to observe errors in the text string before and after phase tracking. Experiment with several values for step size, μ.

(c) Artificially insert into your modem receiver software a step error at some time point in the demodulation carrier phase. Apply the gradient descent carrier phase tracking as implemented in step (b). To view the action of the tracker in your modem simulator, plot both the symbol angle error before tracking and the tracker output signal, θ_k, from Equation 12.15. Try for several choices of step size, μ. What do you observe? **Q12.3**

(d) **[Optional]** Implement decision-directed carrier phase tracking, using the second-order tracker given in Equation 12.19. Apply the adaptive technique as post-processing to receiver outputs obtained in the simulator developed for Chapter 6.

For numerical experimentation, start with these suggested parameters:

sampling frequency, f_s (sps)	44100
carrier frequency (Hz)	12000
frequency offset, f_Δ (Hz)	15
upsampling, L	110
pilot sequence	Barker 13
modulation	BPSK
number of text characters	> 100
passband noise variance	0.001
loop gains	$K_1 = 0.04$, $K2 = 0.008$

Display the decoded text message both before and after the decision-directed tracker. Due to mis-estimation of the carrier frequency, the received symbol phase will drift and eventually cause errors in the decoded text string. To view the action of the tracker in your modem simulator, plot the angle error before tracking, and the symbol-rate tracker signals, $e[n]$ and $\theta[n]$. For example,

```
figure;
plot(angle(symbols./symbols_est)*180/pi,'r');
hold on;grid;ylabel('degrees');
plot(angle(exp(1i*theta))*180/pi);
plot(PhError*180/pi,'g--');
legend('OpenLoop Error','\theta[k]',...
    'e[k]','location','best')
```

Because different pseudo-random noise trials will cause different frequency estimation errors, run your simulator several times to observe errors in the text string before and after phase tracking. Is the adaptive procedure able to track the correct phase for long data packets, despite small errors in frequency offset estimation? Experiment with several values for filter gains, K_1 and K_2.

As alternative to tracking phase drift due to frequency estimation errors, the tracking may be tested by using zero noise at the receiver and artificially inserting into your modem receiver software a carrier frequency offset.

(e) **[Optional]** From step (d), experiment with different values of K_1 and K_2. Choose gains that cause the tracking signal to diverge. What are the poles of the resulting transfer function in Equation 12.13?

(f) **[Optional]** Modify your decision-directed phase tracker in step (d) to operate for QPSK modulation.

(g) **[Optional]** Artificially insert into your modem receiver software a step error at some time point in the demodulation carrier phase. Apply the decision-directed carrier phase tracking with second-order feedback loop, as implemented in step (d). Make a plot of tracking error versus symbol index, n. Try for several choices of controller gains, K_1 and K_2. What do you observe?

(h) **[Optional]** Derive the Costas loop, Equation 12.5, by using gradient ascent to maximize the average energy in the in-phase channel.

(i) **[Optional]** Use the z-transform to derive the controller transfer function, $C(z)$, from Figure 12.2 and the update equations given in Equations 12.10 - 12.11.

(j) **[Optional]** Using z-transforms, derive Equation 12.19 from Equations 12.10 - 12.11.

(k) **[Optional]** Derive Equation 12.13, the closed-loop transfer function from the disturbance, d_k, to the tracking error, $e[k]$. How would you determine stability for a given pair of controller gains, K_1 and K_2? What is the DC gain? For your choices of K_1 and K_2, find and plot the step and ramp responses.

12.3 Demonstration

In Chapters 6 and 7, a small error in frequency estimation at the receiver could lead to an accumulated phase error that destroyed decoding performance in a long data packet. Demonstrate any one of your adaptive carrier phase recovery algorithms as a means for correctly decoding a long data packet in the presence of a small frequency offset or frequency drift.

12.4 Summary

In Chapter 12, adaptive processing has been introduced as a closed-loop procedure for tracking carrier phase at the receiver. For real-valued baseband messages, the Costas loop was derived, using both gradient descent and a second-order tracking controller for update equations. For general symbol constellations, a decision-directed phase tracking procedure was introduced, again using both gradient descent and a second-order controller to derive update equations.

The adaptive and decision-directed approaches introduced here are representative of a wealth of procedures developed over the past four decades (see, e.g., [27, 10, 22, 18]). For example, decision-directed techniques have been developed for the channel equalization challenge introduced in Chapter 8.

APPENDIX A

Software Design Suggestions

Students and instructors alike are encouraged to embrace the development of engineering skills, and software design skills in particular, that are part of a laboratory experience. To this end, we highlight several suggestions as students progress towards implementing a working acoustic modem.

- Work from a structured outline; follow the graphical roadmaps provided by Figures 1.1, 4.1, and 4.3. Give a descriptive caption to each section of code, and segment via the %% section indicator.

- Use descriptive variable names. For debugging and readability, generally avoid re-use of variable names during code development. Memory efficiency can be sacrificed for clarity during development.

- Set modem parameter values in a stand-alone section at the top of your code; or, load parameters from a separate `.mat` file, as discussed in Section 7.2. Do not hardcode nor reset parameters throughout a script. A good rule of thumb is to ensure that no number appears in your code, except possibly for the parameter assignment commands at the very beginning of a script.

- Clear all variables and figures at the beginning of the script (`clear all;close all`) to avoid debugging difficulties that may arise from re-use of variable values inherited from previous executions of the script.

- At the conclusion of each step along the roadmap, generate a new graph to visualize the result of the step. The visualization can be an indispensible aid in debugging and understanding. The command `figure` initiates a new figure window.

- Develop, test, and debug steps along the roadmap one at a time. Steps may be re-used via cut-and-paste of in-line code, or may be re-used as function calls. Efficient progress through the lessons presented in Chapters 1 through 7 will depend on re-use of debugged code as students proceed through staged development of an acoustic modem. Unused sections can be bypassed by use of comment symbol, %. A code section may be created using the double percent marker, %%, and a section may be individually executed using the menu bar selection, "Run section," at the top right (or via Ctrl + Enter). Additionally, breakpoints can provide a useful debugging tool.

- Use complex-valued arithmetic to conveniently implement quadrature modulation and to maintain the I and Q samples linked in a simple format.

APPENDIX B

RF Experiments

The explorations in Chapters 1 through 7 culminate in students implementing a half-duplex modem operating at an acoustic carrier frequency. The same baseband processing can be used to implement a digital communication system operating at a radio frequency carrier. This appendix provides suggestions and links to additional information for students and instructors interested in implementing a radio frequency modem. Although radio frequency propagation has been understood since the 1880s, wireless broadcast can be an ethereal experience, especially when acomplished with the most rudimentary of materials.

B.1 Low-cost DIY Modulation

For do-it-yourself (DIY) construction of a transmitter and receiver from simple materials, both AM and FM approaches provide solutions accessible to students in middle school [30] and beyond.

B.1.1 AM

Mathematically, an envelope detector for large-carrier AM reception is described by

$$v(t) = \frac{\pi}{2}\mathrm{LPF}\{\,|r(t)|\,\} - A \approx m(t). \tag{B.1}$$

The envelope detector is a noncoherent receiver in that it does not use knowledge of the phase of the carrier signal. The gain $\frac{\pi}{2}$ compensates

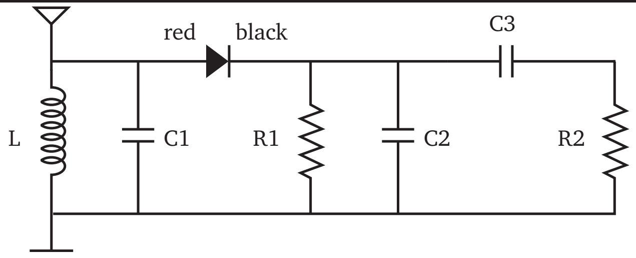


Figure B.1: Circuit model for an envelope detector.

for the loss incurred when lowpass filtering the rectified signal,

$$\frac{\int_0^{\frac{1}{4f_c}} \cos(2\pi f_c t)dt}{\frac{1}{4f_c}} = \frac{2}{\pi}. \tag{B.2}$$

A circuit model of Equation B.1 using passive devices is shown in Figure B.1. The absolute value, $|\cdot|$, is easily approximated using a diode (or, for example, piece of carbon with a blued razor blade). An RC filter provides the lowpass filtering; and, an additional capacitor provides the DC block to achieve the level shift of $-A$. Further, a resonant LC circuit provides tuning of the antenna to the carrier frequency, f_c. The demodulated signal is the voltage $m(t)$; a piezo-electric crystal in parallel with R3 can provide audible output. By converting approximately one trillionth Watt of power from the transmitted waveform into acoustic energy, the circuit is able to produce sound pressures perceptible by the human ear. Many DIY guides are available, e.g. [9, 40, 6, 26].

QA.1 As an exercise, the reader is invited to derive the resonant frequency of a parallel LC circuit. Specifically, find the transfer function, $H(j\omega)$, from $x(t)$ to $y(t)$ for the circuit shown in Figure B.2. (In the figure, the resistance R approximates an antenna given by a long insulated wire.)

An AM transmitter can be constructed from an audio transformer and crystal oscillator [9]. The baseband BPSK signal used in the acoustic modem may serve as single-channel audio input; more generally, a complex-valued QPSK or QAM baseband signal modulated to an acoustic carrier frequency, as in Chapter 7, may serve as a real-valued intermediate frequency signal prior to RF amplitude modulation.

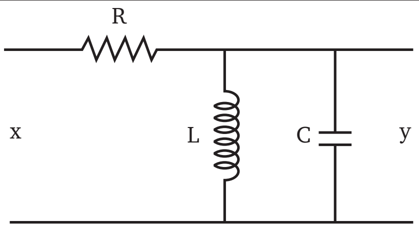


Figure B.2: RLC circuit model.

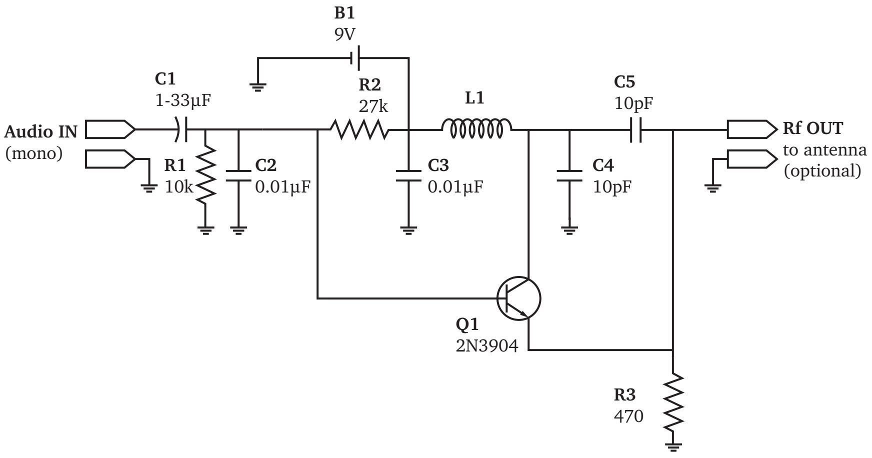


Figure B.3: Circuit schematic for a single-transistor FM transmitter.

B.1.2 FM

The basic regenerative circuit for FM modulation and demodulation was introduced by E. H. Armstrong in 1915 [1, 19]. A super-regenerative FM transmitter can be implemented using a single transistor design [28, 20]. The circuit schematic is given in Figure B.3 [28]; note that capacitor C4 is a variable capacitor for tuning. Many other designs can be found from DIY sources [34].

Similarly, a single-transistor design may be used to implement a super-regenerative FM receiver. A circuit schematic [2, 7, 5] is given in Figure B.4; note that IC1 is an LM386 audio amplifier. Many alternatives, e.g. [23], can be found online.

Rather than build an FM modulator and demodulator, students

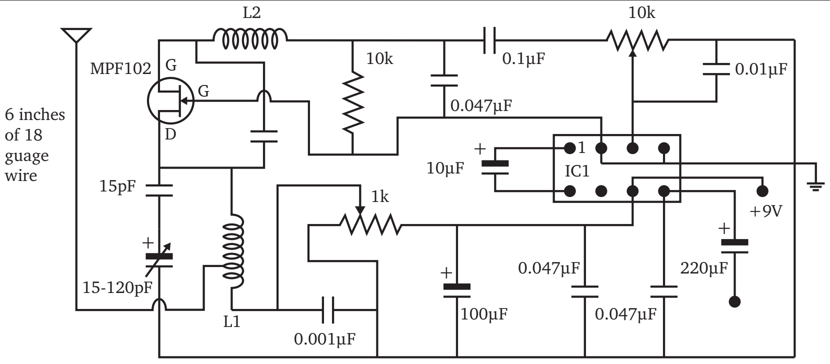


Figure B.4: Circuit schematic for single-transistor FM receiver.

can construct a modem using commercially available integrated circuits. The example shown in Figure B.5 employs ES series modules from Linx Technologies operating at 916 MHz.

In addition, monaural audio input and output can be easily taken from a cordless phone and base to provide a low-cost FM wireless solution. Retailers sell phones for under US$15, and used phones can be found at thrift stores and consignment shops. In 1994, digital cordless phones were introduced operating in the 900 MHz frequency range. Telephones sold in the US use the 900 MHz, 1.9 GHz, 2.4 GHz, or 5.8 GHz bands; phones operating at 900 MHz are typically inexpensive analog models.

Nonlinearities in low-cost FM transmitters and receivers are poorly

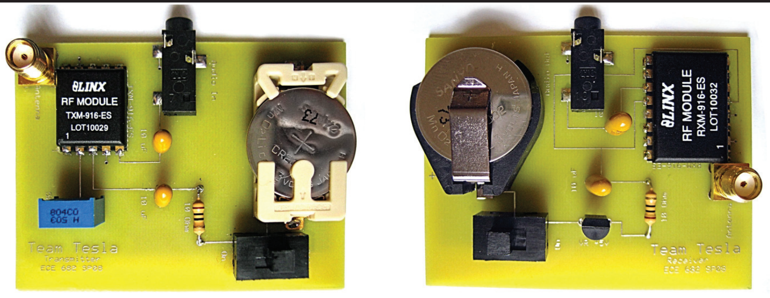


Figure B.5: FM transmitter (left) and receiver (right) constructed from low-cost RF modules.

suited for use with general QAM constellations; but, PSK digital modulation can perform well.

B.2 Commerical SDR Solutions

Several software-define radios are commercially available. Costs range from several hundred to several thousand US dollars per unit.

The **PicoZed SDR** Z7035/AD9361 System-on-Module (SOM) is a credit-card size form-factor PicoZed SDR solution that provides frequency-agile wideband 2×2 receive and transmit paths from RF to baseband. The unit was introduced in 2015 by Avnet in conjunction with Xilinx, Analog Devices, and Mathworks. A bandwidth to 56 MHz can be obtained with a tunable carrier from 70 to 6000 MHz.

The Wireless Open-Access Research Platform (**WARP**) provides a scalable and programmable hardware for software-agile physical layer prototyping. The WARP v3 shown in Figure B.6 integrates a high performance FPGA, two flexible RF interfaces, and multiple peripherals to facilitate rapid prototyping of custom wireless designs [25]. Reference designs and support materials are available online [39]. In addition, teaching materials for use in a ten week laboratory course are available, including slides and lecture videos [8]. The course introduces basics of implementing digital communication systems on real-time hardware and uses the Xilinx System Generator environment.

For radio operation at 2 to 50 MHz, **GenesisRadio** sells transceiver kits and associated SDR software for under US$400.

Ambitious students may wish to contruct their own software-agile transceiver, for example using the **Maxim 2830** evaluation kit. The MAX2830 is a direct conversion, zero-IF, RF transceiver designed specifically for the 2.4GHz to 2.5GHz ISM band. The chip integrates all circuitry required to implement the RF transceiver function, providing an RF power amplifier (PA), an Rx/Tx and antenna diversity switch, RF-to-baseband receive path, baseband-to-RF transmit path, voltage-controlled oscillator (VCO), frequency synthesizer, crystal oscillator, and baseband/control interface.

Digital video broadcast USB dongles based on the **Realtek RTL 2832U** can be used as an inexpensive SDR receiver, since the chip allows transferring the raw I/Q samples to a host computer. The

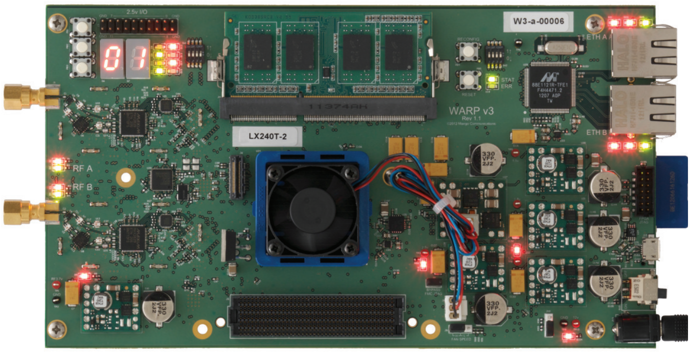


Figure B.6: WARP provides a scalable and programmable hardware platform for prototyping software-agile communication systems. (Photograph used with permission.)

RTL2832U outputs 8-bit I/Q-samples at approximately 2.5 MS/s. The frequency range depends on the tuner employed [29]. The GNU Radio collection of tools can be used to build custom radio devices [14].

APPENDIX C

Functions

This appendix provides code for eight custom functions referenced in the text and listed in Table C.1. Scripts and functions are available for download at OpenStax CNX[1] and Mathworks MATLAB Courseware[2].

Table C.1: Custom functions.

Function	Description
`plottf.m`	Make time and frequency plots
`firlpf.m`	Design low-pass filter design
`char2psk.m`	Template for PSK digital modulation
`psk2char.m`	Template for PSK digital demodulation
`srrc.m`	Design square-root raised cosine filter
`eyediagram.m`	Make eye diagram
`makepilots.m`	Make pilot sequence
`packetdetect.m`	Detect data packet

[1] `http://cnx.org`
[2] `http://www.mathworks.com/academia/courseware`

plottf

```
%PLOTTF Plot sampled signal in time and frequency domains
%   PLOTTF(x,Ts) plots the time-domain samples in vector x, assuming
%   that Ts is the sampling interval in seconds, and also plots the
%   Riemann-sum approximation of the Fourier transform between the
%   frequencies of -1/(2Ts) and 1/(2Ts) Hertz.
%
%   PLOTTF(x,Ts,'t') plots only the time-domain signal.
%
%   PLOTTF(x,Ts,'f') plots only the frequency-domain signal.
%
%   In all cases, PLOTTF returns handles to the graphical objects.

% P. Schniter. Used with permission.

function hh = plottf(x,Ts,str)

plot_type = 0;
if nargin==3,
  if str=='f',
    plot_type = 1;
  elseif str=='t',
    plot_type = 2;
  end;
end;

N=length(x); % discrete signal length
t=Ts*(0:N-1); % time vector
if 2*floor(N/2)==N,
  f=(-N/2:N/2-1)/(Ts*N); % frequency vector
else
  f=(-(N-1)/2:(N-1)/2)/(Ts*N); % frequency vector
end;
X=Ts*fft(x); % do DFT
Xs=fftshift(X); % % shift it for plotting

if plot_type==1,
  hh = plot(f,abs(Xs)); % plot magnitude spectrum
  xlabel('frequency [Hz]');
  ylabel('magnitude');% label the axes
elseif plot_type==2,
  if isreal(x),
    hh = plot(t,x);% plot the real waveform
    xlabel('time [sec]');
    ylabel('amplitude'); % label the axes
  else
    hh = plot3(t,real(x),imag(x)); % plot the complex waveform
    xlabel('time [sec]');
    ylabel('real'); zlabel('imag'); % label the axes
  end;
else
  subplot(2,1,1);
  if isreal(x),
    hh = plot(t,x); % plot the real waveform
    xlabel('time [sec]');
    ylabel('amplitude');  % label the axes
  else
    hh = plot3(t,real(x),imag(x)); % plot the complex waveform
    xlabel('time [sec]');
    ylabel('real'); zlabel('imag'); % label the axes
  end;
  subplot(2,1,2);
  hh = [hh,plot(f,abs(Xs))]; % plot magnitude spectrum
  xlabel('frequency [Hz]');
  ylabel('magnitude') % label the axes
  subplot(2,1,1);
end;
```

firlpf

```
function [h]=firlpf(Lh,fpass,fstop,fsample)
%FIRLPF  Least-squares design of linear-phase low-pass filter.
%
% [h] = firlpf(Lh,fpass,fstop,fsample) designs Type I, length Lh,
% linear-phase low-pass filter via least-squares design.
% Lh must be odd.
%
% Lh        filter length (odd)
% fpass     edge of passband (Hz)
% fstop     edge of stopband (Hz)
% fsample   sampling rate (Hz)
% h         designed impulse response

% Digital Communication Laboratory
% Autumn 2014

%% argument check
if(nargin < 4)
    error('Too few input arguments for FIRLPF.')
end
if(fpass >= fstop)
    error('Passband edge must be less than stopband edge')
end
if(fsample <= 2*fstop)
    error('Sampling rate is sub-Nyquist')
end
if( mod(Lh,2) ~= 1);
    Lh=Lh+1;
    warning('Lh incremented for Type I FIR filter.');
end
%% normalized frequency (radians per sample)
fpass = fpass / (fsample/2) * pi;
fstop = fstop / (fsample/2) * pi;
%% apply orthogonality principle for LS solution
indx=(1:(Lh-1)/2);
RHS = [fpass sin(fpass*indx)./indx].';
% diagonal of matrix
tmp=zeros(1,length(indx)+1);
tmp(1) = pi - (fstop - fpass);
tmp(2:end)=tmp(1)/2+(sin(2*indx*fpass)-sin(2*indx*fstop))./(4*indx);
LHS = diag(tmp);
% off-diagonals
for row=0:(Lh-1)/2;
   for col=row+1:(Lh-1)/2;
       LHS(row+1,col+1) = ...
         (sin((col+row)*fpass)-sin((col+row)*fstop))/2/(col+row)+...
         (sin((col-row)*fpass)-sin((col-row)*fstop))/2/(col-row);
       LHS(col+1,row+1) = LHS(row+1,col+1);%symmetry
   end
end
g = LHS\RHS;g=g.';%solve linear system
h = [g(end:-1:2)/2 g(1) g(2:1:end)/2];%linear phase impulse response
%end of function
```

char2psk template

```
%% template for char2psk.m
function [a,bits] = char2psk(str,M);
% char2psk.m
% Function to map a charater string to PSK symbols
% (See also psk2char.m)
%
% Inputs:
% str       string variable with text
% M        size of PSK constellation (2 or 4)
%           or, user can select 'bpsk' or 'qpsk'
% Outputs:
% a         output list of PSK symbols (complex-valued)
% bits      output list of bits (logical)

% Digital Communications Laboratory
% Autumn 2014

%% error checks
if(nargin ~= 2)
    error('Error: char2psk.m requires two input arguments.')
end
%% text to symbols
% first, string to decimal to binary
str_binary = dec2bin(double(str),8);
% convert array row-by-row to one long string
bits = reshape(str_binary.',1,8*length(str));
% binary to symbols
switch M
    case {2,'bpsk','BPSK'}
        M = 2;
        a = (2*bin2dec(bits.')-1).';
    case {4,'qpsk','QPSK'}
        M = 4;
        %
        %create your QPSK code here
        %
end
bits=(bits=='1');% convert char to logical
% end of function
```

psk2char template

```
%% template for psk2char.m
function [str,bits] = psk2char(y,M)
% psk2char.m
% Function to decode M-PSK symbols to a string of ASCII text.
% (See also char2psk.m)
%
% Inputs:
% M      size of PSK constellation (2 or 4; or 'bpsk or 'qpsk')
% y      list of PSK IQ points (complex numbers)
% Outputs:
% strbin string variable with ASCII text
% bits   list of 0/1 bits (as logical variables)
%
% Digital Communication Laboratory
% Autumn 2014

%% error checks
if(nargin ~= 2)
    error('Error: psk2char.m requires two input arguments')
end
if (isnumeric(y) ~= 1 || isempty(y))
    error('Error: the input y must be numeric.')
end

%% symbol slicing
% first, symbol string to string of integer labels: 0,1,...,log2(M)
% labels are integers; use double data type
  switch M
     case {2,'bpsk','BPSK'}
        M = 2;
        labels = (real(y) >= 0);
     case {4,'qpsk','QPSK'}
        M = 4;
        %
        %create code here; see Figure 3.1(b) for labeling quadrants
        %
  end
% second, labels into string of bits
  N = length(labels)*log2(M)/8;%number of characters
  tmp = dec2bin(labels,log2(M));%label to binary
  strbin = reshape(tmp.',8,N).';%array, labels to 8 bits/character
% third, convert from char to logical:
  bits = logical(bin2dec(reshape(tmp.',1,N*8).').');
% fourth, convert binary to decimal ASCII to character
  str=char(bin2dec(strbin)).';%
% end of function
```

srrc

```
function g = srrc(D,alpha,L)
% SRRC Fractionally-sampled square-root raised cosine pulse.
%
% [g] = srrc(D,alpha,L) creates a fractionally-sampled square-root
% raised cosine pulse
%
%    D       one-half the length of srrc pulse in symbol durations
%    alpha   excess bandwith (valude between 0 and 1);
%            alpha=0 gives a sinc pulse
%    L       samples per symbol (oversampling factor);
%            L must be a positive integer
%    g       samples of the srrc pulse

% Digital Communication Laboratory
% Autumn 2014

%% argument check
if(nargin < 3)
    error('Too few input arguments for SRRC')
end
if( min( [D,alpha,L] ) < 0 )
    error('Inputs must be non-negative for SRRC.')
end
if( round(L) ~= L )
    error('Input L must be a non-negative integer for SRRC.')
end

%% compute samples
k = -D:(1/L):D; % k is t/T
g = (sin(pi*(1-alpha)*k)+(4*alpha*k).*cos(pi*(1+alpha)*k)) ./ ...
  ((pi*k) .* (1-(4*alpha*k).^2))/sqrt(L);
%% fill in for denominator zeros
g(k==0) =  (1 + (4/pi-1)*alpha)/sqrt(L);
g(abs(abs(4*alpha*k)-1) < sqrt(eps)) = ...
   alpha/sqrt(2*L)*( (1+2/pi)*sin(pi/4/alpha) + ...
  (1-2/pi)*cos(pi/4/alpha));
%end of function
```

eyediagram

```
function [] = eyediagram(y_up,L,N,IQ)
% eyediagram.m
% Function to plot an eye diagram
%
% y_up    upsampled matched filter outputs
% L       downsampling factor (integer)
% N       number of symbols
% IQ      string, 'complex', if both I and Q are desired
%
% Note that first sample of y_up is assumed to occur at a symbol time.

% Digital Communication Laboratory
% Autumn 2014

%% error checks
if(nargin < 2)
    error('Error: eyediagram.m requires two input arguments')
end
if(nargin == 3), IQ='real'; end
if(strcmp(IQ,'complex'))
    IQ=1;%plot both I and Q
else
    IQ=0;
end
%N-2 segments for N symbols

%% extract symbol period length segments
% start of first full interval
start = 1 +ceil(L/2);
Y_up = reshape(y_up(start:start+(N-2)*L-1),L,N-2); %extract segments

%% plots
if(IQ == 0)
    % plot eye diagram, I channel only
    figure;
    plot(-floor(L/2)/L+(0:L-1)/L,real(Y_up));% superimpose them
    title('Eye Diagram','fontsize',13);
    xlabel('relative symbol index','fontsize',13);
else
    % plot eye diagram, I & Q channels
    figure;
    subplot(211);plot(-floor(L/2)/L+(0:L-1)/L,real(Y_up));
    title('Eye Diagram','fontsize',13);
    subplot(212);plot(-floor(L/2)/L+(0:L-1)/L,imag(Y_up));
    xlabel('relative symbol index','fontsize',13);
end
```

makepilots

```
function [pilots] = makepilots(opt)
%MAKEPILOTS  generate pilot sequence
%
% [pilots] = makepilots(opt) generates a sequence of pilot symbols
%
% opt indicator to select pilot sequence:
%  Barker: 'barker13', 'B', 'b', 'Barker', or 'barker' (default)
%  PN: 'pn51', 'PN', 'pn', or 'pseudorandom'
%  split Barker: 'splitbarker' or 'splitBarker'
%
% The length-13 Barker code and length-51 pseudo-random code
% generated here are employed in the acoustic transmitter mobile
% app available at iTunes Apps or Google Play.

% Digital Communication Laboratory
% Autumn 2014

if(nargin == 0), opt='barker13'; end % default
switch opt
    case {'barker13', 'B', 'b', 'Barker', 'barker'}
        pilots=[+1 +1 +1 +1 +1 -1 -1 +1 +1 -1 +1 -1 +1];
    case {'pn51', 'PN', 'pn', 'pseudorandom'}
        pilots = [+1,+1,-1,+1,+1,-1,-1,+1,+1,+1, ...
                   -1,+1,+1,-1,+1,-1,-1,+1,+1,+1, ...
                   +1,-1,+1,+1,+1,+1,+1,-1,+1,-1,...
                   +1,-1,-1,-1,-1,+1,+1,-1,+1,-1, ...
                   -1,-1,+1,+1,-1,-1,-1,+1,+1,+1,-1];
    case {'splitbarker', 'splitBarker'}
        pilots=[+1 +1 +1 +1 +1 -1 -1 +1 +1 -1 +1 -1 +1 ...
                +1 +1 +1 +1 +1 -1 -1 +1 +1 -1 +1 -1 +1];
    otherwise
        warning('Unexpected pilot type. Default barker13.')
        pilots=[+1 +1 +1 +1 +1 -1 -1 +1 +1 -1 +1 -1 +1];
end
```

packetdetect

```
function [ rx_Trim ] = packetdetect( yup , L,  pkt_len)
%PACKETDETECT  energy detector to approximately locate a data packet;
%              trims a received upsampled signal
%
% [rx_Trim] = packetdetect(yup,L,pkt_len)
%
% yup       fractionally sampled matched filter output
% L         upsampling factor
% pkt_len   packet length (including both header and payload)

% Digital Communication Laboratory
% Autumn 2014

Mask=ones(1,pkt_len*L);
Energy_Corr=conv(abs(yup).^2,Mask);
Energy_Corr=Energy_Corr(length(Mask):end);
[~,Max_Energy_Index]=max(Energy_Corr);
End_Packet_Index=min(length(yup),Max_Energy_Index+(pkt_len+3)*L);
rx_Trim=yup(Max_Energy_Index-3*L-round(max(10,L)*rand(1,1)) : ...
    End_Packet_Index);
%for educational use:
%  randomized offset to test frame timing;
%  presumes at least 4L noise samples precede packet
end
```

Glossary

Notation

The indexing conventions $f(t)$, $f[k]$, and $f[n]$ are adopted to suggest a continuous-time signal representation, a fractionally-sampled baseband signal representation, and a symbol-rate sampled signal representation, respectively. A tilde, as in $\tilde{m}$, denotes a complex-valued quantity carrying both in-phase and quadrature components.

Symbol	**Quantity**
f_c	carrier frequency
f_s	sampling frequency
T_s	sampling period
T	symbol period
M	constellation size
$g_{\text{tx}}(t)$, $g_{\text{tx}}[k]$	pulse shaping filter
$G_{\text{tx}}(f)$	Fourier transform of $g_{\text{tx}}(t)$
D	pulse half-width, in symbol intervals
L	upsampling factor
α	excess bandwidth; a pulse shaping parameter
W	one-sided baseband bandwidth (Hertz)
$p(t)$	convolution of pulse shape and matched filter
$P(f)$	Fourier transform of $p(t)$
N_{tr}	length of pilot, or training, symbol sequence

Symbol	**Quantity**
$\tilde{a}[n]$	symbol sequence, including pilots
$a_\uparrow[k]$	upsampled symbol sequence
$\widehat{a}[n]$	decoded symbol sequence
$\tilde{m}[k]$, $\tilde{m}(t)$	baseband message at transmitter
$m_I(t)$	in-phase (real) component of message $m(t)$
$m_Q(t)$	quadrature (imaginary) component of $m(t)$
$s(t)$, $s[k]$	bandpass transmission
f_Δ	frequency offset
ϕ	phase offset
N_0	noise power spectral density, Watts per Hertz
E_b	energy per bit
E_s	energy per symbol
K	order of multi-path channel model
$\delta[k]$	Kronecker delta sequence
$\delta(t)$	Dirac delta function
j	imaginary unit, $j = \sqrt{-1}$
$\mathrm{Re}\{\cdot\}$	real part of
$\mathrm{Im}\{\cdot\}$	imaginary part of
$r(t)$, $r[k]$	bandpass signal at receiver
$\tilde{v}(t)$, $\tilde{v}[k]$	baseband message at receiver
$v_I(t)$	in-phase (real) component of $\tilde{v}(t)$
$v_Q(t)$	quadrature (imaginary) component of $\tilde{v}(t)$
B_p	pass-band edge frequency (Hertz)
B_s	stop-band edge frequency (Hertz)
$\tilde{y}_\uparrow[k]$	fractionally sampled matched filter output
$\tilde{y}[n]$	symbol-rate matched filter outputs at receiver

Acronyms

ADC	analog-to-digital converter
AM	amplitude modulation
ASCII	American standard code for information interchange
AWGN	additive white Gaussian noise
BER	bit error rate
BPSK	binary phase-shift keying
CDMA	code-division multiple access
CMA	constant modulus algorithm
CP	cyclic prefix
DAC	digital-to-analog converter
DDC	digital downconverter
DFT, IDFT	discrete Fourier transform, inverse DFT
DDS	direct digital synthesis
DPSK	differential phase-shift keying
DSP	digital signal processor
DUC	digital upconverter
FFT, IFFT	fast Fourier transform, inverse FFT
FIR	finite impulse response
I	in-phase
I/O	input/output
IF	intermediate frequency
ISI	inter-symbol interference
PA	power amplifier
PSK	phase-shift keying
LNA	low-noise amplifier
LPF	low-pass filter
NCO	numerically controlled oscillator
OFDM	orthogonal frequency division multiplexing
PHY	physical layer
Q	quadrature
QAM	quadrature amplitude modulation
QPSK	quadrature phase-shift keying
RF	radio frequency
SDR	software-defined radio
SNR	signal to noise ratio
sps	samples per second
SRRC	square-root raised cosine pulse
UHF	ultra-high frequency (300 – 1,000 MHz)
VCO	voltage-controlled oscillator

Bibliography

[1] Edwin H. Armstrong. Some recent developments in the audion receiver. *Proc. Inst. Radio Engineers (IRE)*, 3:215–247, 1915. 143

[2] Patrick Cambre. A one transistor super-regenerative FM receiver. http://web.archive.org/web/20090121123846/http://braincambre500.freeservers.com/rss(1)(1)(1).htm. Archived 2009-01-21; accessed 2015-06-09. 143

[3] R. W. Chang. Synthesis of band-limited orthogonal signals for multi-channel data transmission. *Bell System Technical Journal*, 45(10):1775–1796, 1966. 113

[4] J.P. Costas. Synchronous communications. *Proceedings of the IRE*, 44(12):1713–1718, Dec 1956. 131

[5] Martijn de Milliano. Simple FM radio. www.millibyte.nl/index.php?page=fm-radio, 2002. Accessed: 2015-06-09. 143

[6] Steven Dufresne. Crystal radios. www.rimstar.org/equip/crystal_radios.htm and www.youtube.com/watch?v=VqdcU9ULAlA. Accessed: 2015-06-09. 142

[7] electronicsNmore. One transistor FM receiver. www.youtube.com/watch?v=Wdi0X0YQSxM, 2013. Accessed: 2015-06-09. 143

[8] Evan Everett and Michael Wu. Eec 433 architeture for wireless communications. www.warpproject.org/trac/wiki/Rice_ELEC_433. Accessed: 2015-06-12). 145

[9] Simon Quellen Field. A simple AM transmitter. http://sci-toys.com/scitoys/scitoys/radio/am_transmitter.html. Accessed: 2015-06-09. 142

[10] Michael Fitz. *Fundamentals of Communications Systems*. McGraw-Hill, New York, NY, 2007. viii, 138

[11] Gene F. Franklin, David J. Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 4th edition, 2001. 132

[12] L. E. Frenzel. The elusive software-defined radio. `http://electronicdesign.com/communications/elusive-software-defined-radio`, 2006. 2

[13] Robert G. Gallager. *Principles of Digital Communication*. Cambridge University Press, New York, NY, 2008. viii

[14] GNUradio. GNURadio: The Free & Open Software Radio Ecosystem. `www.gnuradio.org/redmine/projects/gnuradio/wiki`. Accessed: 2015-06-09. 146

[15] Robert W. Heath, Jr. *Digital Communications: Physical Layer Exploration Lab using the NI USRP*. NTS Press, Allendale, NJ, 2012. ix, 97

[16] M. Heideman, D.H. Johnson, and C.S. Burrus. Gauss and the history of the fast Fourier transform. *IEEE ASSP Magazine*, 1 (4):14–21, October 1984. 116

[17] R. Johnson, Jr., 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*, 86(10): 1927–1950, Oct 1998. 64

[18] C. Richard Johnson Jr. and William A. Sethares. *Telecommunications Breakdown: Concepts of Communications Transmited via Software-Defined Radio*. Prentice Hall, Upper Saddle River, NJ, 2004. viii, ix, 55, 100, 138

[19] Charles Kitchin. High peformance regenerative receiver design. *QEX*, pages 24–36, 1998. Available online `www.arrl.org/files/file/Technology/tis/info/pdf/9811qex026.pdf`. 143

[20] Tetsuo Kogawa. Making the simplest FM transmitter. `www.translocal.jp/radio/micro/simplestTX01.pdf`, 2007. Accessed: 2015-06-09. 143

[21] C. Emre Koksal. *ECE 5000, Introduction to Analog and Digital Communication.* unpublished lecture notes, Columbus, OH, 2015. 110, 111

[22] Upamanyu Madhow. *Fundamentals of Communications Systems.* Cambridge University Press, Cambridge, UK, 2008. viii, 138

[23] Andrew R. Mitz. Build a one transistor FM radio. `www.somerset.net/arm/fm_only_one_transistor_radio.html`, 2013. Accessed: 2015-06-09. 143

[24] P. Moose. A technique for orthogonal frequency division multiplexing frequency offset correction. *IEEE Trans. Commun.*, 42 (10):2908–2914, October 1994. 97

[25] Patrick Murphy. WARP v3 Kit. Mango Communications `http://mangocomm.com/products/kits/warp-v3-kit`. Accessed: 2015-07-13). 145

[26] Bre Pettis. How to make a foxhole radio. `www.youtube.com/watch?v=skKmwT0EccE`, 2007. Accessed: 2015-06-09. 142

[27] John Proakis. *Digital Communications.* McGraw-Hill, New York, NY, 2001. viii, 38, 90, 104, 138

[28] Sean Michael Ragan. Make-zine: Super-simple iPod FM transmitter. `http://makezine.com/projects/super-simple-fm-transmitter`, 2013. Accessed: 2015-06-09. 143

[29] rtl-sdr. rtl-sdr. `sdr.osmocom.org/trac/wiki/rtl-sdr`. Accessed: 2015-06-09. 146

[30] H. C. Sandifer. Telegraph plant by *Amateur Work* readers. *Amateur Work, Illustrated*, page 223, June 1904. Available online, `www.earlyradiohistory.us/1904ama.htm`. 141

[31] Philip Schniter. *Introduction to Analog and Digital Communications [Kindle edition].* Amazon Digital Services, Inc., Seattle, WA, 2011. ASIN B005JU4GTM. ii, viii, ix, 100

[32] SDR Forum. SDRF cognitive radio definitions,. `http://www.sdrforum.org/pages/documentLibrary/documents/SDRF-06-R-0011-V1_0_0.pdf`, 2007. Accessed: 2007-11-15. 2

[33] Claude Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948. 103

[34] Art Swan. Art Swan's electronic circuits. `www.angelfire.com/art2/artswan`. Accessed: 2015-06-09. 143

[35] O.Y. Takeshita. On maximum contention-free interleavers and permutation polynomials over integer rings. *IEEE Transactions on Information Theory*, 52(3):1249–1253, March 2006. 108

[36] J. Treichler and M. Larimore. New processing techniques based on the constant modulus adaptive algorithm. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 33(2):420–431, Apr 1985. 66

[37] Steven A. Tretter. *Communication System Design Using DSP with Laboratory Experiments for the TMS320C6713 DSK*. Springer, New York, NY, 2008. ix, 133

[38] S. Verdu. Guest editorial. *IEEE Transactions on Information Theory*, 44(6):2042–2044, Oct 1998. 104

[39] WARP. Warp project. `www.warpproject.org`. Accessed: 2015-06-09. 145

[40] Charles Wenzel. Crystal radio circuits. `www.techlib.com/electronics/crystal.html`. Accessed: 2015-06-09. 142

Index