Advanced Theory of Communications (LAB)

Computer Exercise I: Sampling Theory

Objectives: In this experiment you will investigate Sampling Theorem.

A-Prelab Assignment: Given signal $x(t) = \text{sinc}(t)$:

1. Find out the Fourier transform of $x(t)$, $X(f)$, sketch them.
2. Find out the Nyquist sampling frequency of $x(t)$.
3. Given sampling rate f_s, write down the expression of the Fourier transform of $x_s(t)$, $X_s(f)$ in terms of $X(f)$.
4. Let sampling frequency f_s = 1Hz. Sketch the sampled signal $x_s(t) = x(kT_s)$ and the Fourier transform of $x_s(t)$, $X_s(f)$.
5. Let sampling frequency f_s = 2Hz. Repeat 4.
6. Let sampling frequency f_s = 0.5Hz. Repeat 4.
7. Let sampling frequency f_s = 1.5Hz. Repeat 4.
8. Let sampling frequency f_s = 2/3Hz. Repeat 4.
9. Design a Matlab function to calculate the Fourier transform of a sampled signal $x_s(t)$, $X_s(f) = \sum_k x(kT_s)\exp(-j2\pi fkT_s)$. This is necessary in your experiments.

NOTE: In Matlab and this experiment, sinc(t) is defined as sinc(t) = sin(πt)/(πt). Under this definition: sinc(2Wt) → 1/(2W) rect(f/2W).

B. Procedure:

1. Design Matlab programs to illustrate items 4-8 in Prelab. You need to plot all the graphs.
2. Compare your results with your sketches in your Prelab assignment and explain them.

Computer Exercise II: Binary Signaling Formats

Objectives: In this experiment you will investigate how binary information is serially coded for transmission at baseband frequencies. In particular, you will study:

- line coding methods which are currently used in data communication applications;
- power spectral density functions associated with various line codes;
- causes of signal distortion in a data communications channel;
- effects of intersymbol interference (ISI) and channel noise by observing the eye pattern.
-

PART A

Binary 1's and 0's such as in pulse-code modulation (PCM) systems, may be represented in various serial bit signaling formats called line codes. In this section you will study signaling formats and their properties.

A1- Given the binary sequence $x = \{1, 0, 1, 0, 1, 1\}$ sketch the waveforms representing the sequence x using the following line codes:
a. Unipolar NRZ / b. Polar NRZ / c. Unipolar RZ / d. Bipolar RZ / e. Manchester.
Assume unit pulse amplitude and use binary data rate R_b = 1 kbps.

A2- Determine and sketch the power spectral density (PSD) functions corresponding to the above line codes. Let $f_1 > 0$ be the location of the first spectral null in the PSD function. If the transmission bandwidth B_T of a line code is determined by f_1, determine B_T for the line codes in A1 as a function of R_b.

A3- For the above set of line codes determine which will generate a waveform with no dc component regardless of binary sequence represented. Why is the absence of a dc component of any practical significance for the transmission of waveforms?

A4- Generate a 1,000 sample binary sequence. Display the PSD function of each line code used in part A1. Let f_{p1} be the first spectral peak, f_{n1} be the first spectral null, f_{p2} be the second spectral peak, and f_{n2} be the second spectral null such that all f 's are > 0. Record your observations in Table 1. Location of the first spectral null determines transmission bandwidth B_T .

Table 1

R_b	f_{p1}	f_{n1}	f_{p2}	f_{n2}	B_T
Unipolar NRZ					
Polar NRZ					
Unipolar RZ					
Polar RZ					
Manchester					

A5- To illustrate the dependence of the PSD function on the underlying binary data rate, use the Polar NRZ line code and vary R_b as 5 kbps; 10 kbps; and 20 kbps. Observe the location of spectral peaks and nulls and relate them to R_b.
A6- For a baseband data communications channel with usable bandwidth of 10 kHz, what is the maximum binary data rate for each of the line codes examined in part A1.

PART B: Channel Characteristics

In this part you will simulate characteristics of a communications channel.

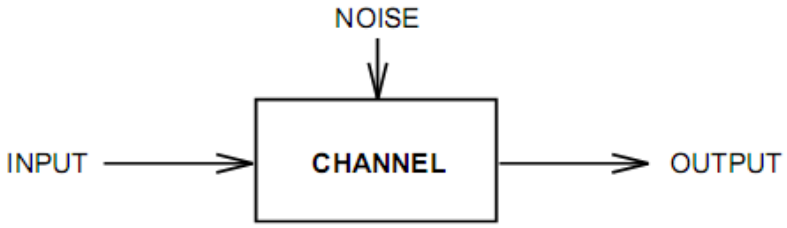


Fig. 1 Channel model.

B.1 Create a 10 sample binary sequence *x* and generate a waveform representing *x* in polar NRZ signaling format. Use R_b = 1 kbps. From your observation in part A, determine the transmission bandwidth B_T of *x*.

B.2 Consider a baseband data transmission channel with unity gain and additive white Gaussian noise (AWGN) where the noise power is $\sigma_0^2 = 10^{-2}$ W and the channel bandwidth is 4.9 kHz channel. Display the channel input and output waveforms. Estimate the transmitted sequence i.e. $\hat{x}$ from the display of the channel output waveform. Compare your estimate with the original sequence *x*.

B.3 Effect of channel noise on the transmitted waveform: Gradually increase the channel noise power while keeping the channel bandwidth at 4.9 kHz and observe changes in the channel output (let $\sigma^2 = (0.1; 0.5; 1; 2; 10)\sigma_0^2$). At what noise power level, does the channel output waveform become indistinguishable from noise?

B.4 You can also observe effects of increasing channel noise power by looking at the PSD of the channel output waveform. Since the channel noise is additive and uncorrelated with the channel input, determine an expression that will describe the PSD of the channel output in terms of the input and noise PSD functions.

B.5 Effects of channel bandwidth on transmitted waveform: Distortion observed in the time display of the channel output is due to finite bandwidth of the channel and due to noise. To study distortion due to channel bandwidth only, set noise power to zero and regenerate the channel output waveform.

B.6 Investigate effects of channel bandwidth on the output waveform by setting the bandwidth to (0.5; 1; 2; 3; 5) KHz. Observe the delay in the output waveform due to filtering characteristics of the channel. Plot the input and output waveforms. Determine the appropriate sampling instants for the decoding of the waveform for the case BW = 500 Hz.

PART C: Eye Diagram

Effects of channel filtering and noise can be best seen by observing the output waveform in the form of an "eye diagram". The eye diagram is generated with multiple sweeps where each sweep is triggered by a clock signal and the sweep width is slightly larger than the binary data period $T_b = 1=R_b$. In this simulation the eye diagram is based on a sweep width of $2T_b$.

C1- Generation of Eye Diagram:

$$x = [\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ ];$$

The eye diagram for the waveform *x* represents what you should expect to see for an undistorted signal. To observe how the eye diagram is generated and to observe effects of signal distortion as the signal x is transmitted over a finite bandwidth channel with no noise component:
If the second argument to the function eye diagram is negative, you have to hit the Return key for the next trace to be displayed. This will assist you to understand how the eye diagram is generated.

C2- Key parameters to be measured with an eye diagram are shown below. Interpretation of the eye pattern as depicted in Figure 2:

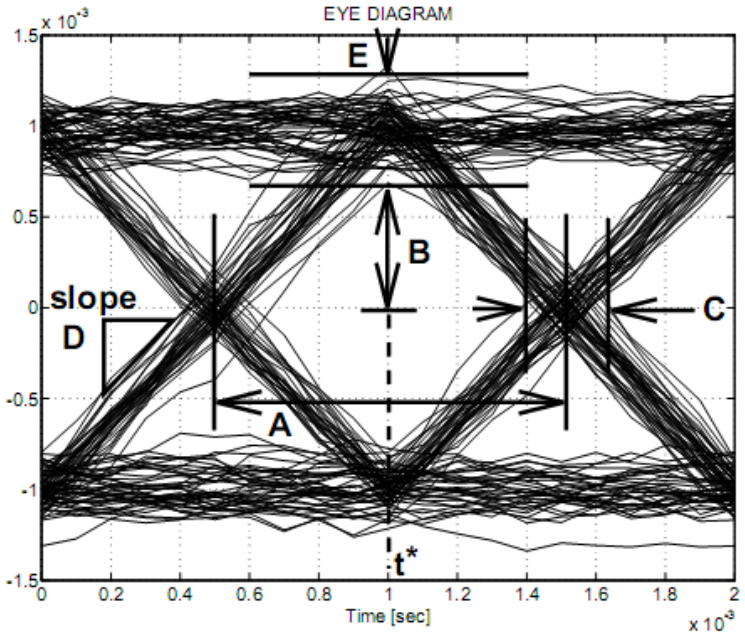


Fig. 2 Interpretation of the eye pattern.

A: time interval over which the waveform can be sampled;
B: margin over noise;
C: distortion of zero crossings;
D: slope: sensitivity to timing error;
E: maximum distortion;
t^* : the optimum sampling instant measured with respect to the time origin. If the binary data period is T_b, then the waveform will be sampled at t^*; $t^* + Tb$; $t^* + 2Tb$; … for signal detection.

Generate the eye diagram from a polar NRZ waveform at the channel output for values of noise variance σ^2 and channel bandwidth BW shown in Table 2. Record t^*, A and B for each set of σ^2 and BW.

Table 2

Polar NRZ Line Code				
σ^2	BW	t^*	A	B
0.01	3000			
	2000			
	1000			
0.02	4000			
0.08				
0.1				

C3- Repeat step C2 for bipolar RZ line code and record your results in a table as Table 2.

C4- When you compare the eye diagrams from C2 and C3 for $\sigma^2 = 0.01$ and BW = 1000, for which line code do you observe a "reasonable" eye diagram? Explain the difference in terms of the respective line code properties.

Computer Exercise III: Digital Data Transmission

The objectives of this laboratory are:

1. To implement baseband, amplitude shift keying, and phase reversal keying for binary communication systems.
2. To investigate the operation of these systems.
3. To measure the effects of noise on the probability of error in the received data.

Digital Modulation Techniques

There are three ways in which the bandwidth of the channel carrier may be altered simply. It is worth emphasizing that these methods are chosen because they are practically simple, not because they are theoretically desirable. These are the altering of the amplitude, frequency and phase of the carrier sine wave. These techniques give rise to amplitude-shift-keying (ASK), frequency-shift-keying (FSK) and phase-shift-keying (PSK), respectively.

ASK describes the technique the carrier wave is multiplied by the digital signal *f(t)*. Mathematically, the modulated carrier signal is *s(t)*:

$$s(t) = f(t)\sin(2\pi f_C t + \phi)$$

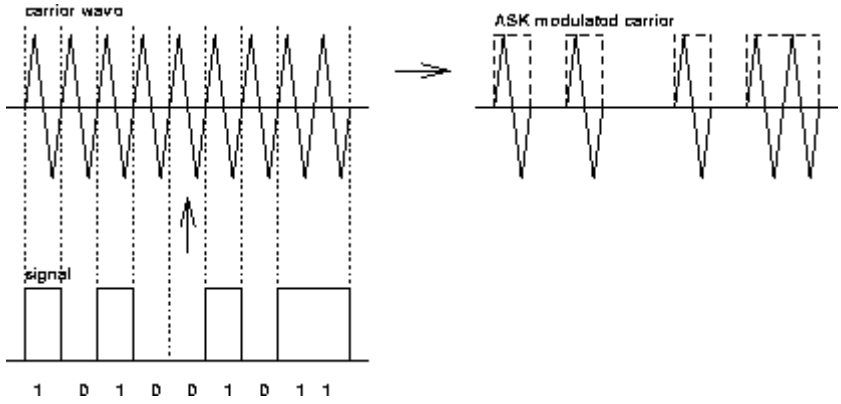


Figure 1: Amplitude shift keying

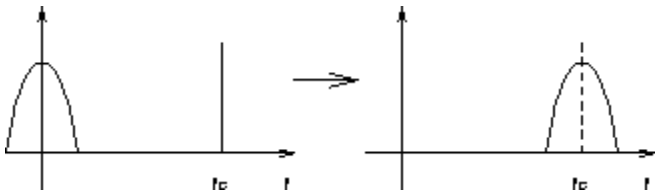


Figure 2: Amplitude shift keying -- frequency domain

The pulses representing the sample values of a PCM waveform can be transmitted on a RF carrier by use of amplitude, phase or frequency modulation. For digital amplitude modulation, it is known as Amplitude Shift Keying. Here, the carrier amplitude is determined by the data bit for that interval. We note that the transmitter for such a system simply consists of an oscillator that is gated on and off; accordingly, ASK is often referred to as *on-off keying*. The oscillator runs continuously as the on-off gating is carried out.

The circuit is connected according to the schematic shown. A silicon diode and a 50-ohm terminator are also used. The VCO generates a 20 kHz carrier signal and receiver reference. The summer and variable power supply changes the threshold of the decision circuit.

Threshold Adjustment:

The output of the integrator has values between zero volts and "A" volts. The threshold voltage is selected to be +A/2 volts. In other words, the output of the decision summer should be –A/2 ("0") and +A/2 ("1"). This can be easily done if the output of the summer with the DC supply is viewed on the scope. The Dc level is adjusted until the square wave waveform is centered about 0 volts. The limiter converts this to ±2.5V.

PSK describes the modulation technique that alters the phase of the carrier. Mathematically:

$$s(t) = \sin(2\pi f_C + \phi(t))$$

Phase Shift Keying

The pulses representing the sample values of PCM waveform when transmitted using phase modulation, Phase Shift Keying occurs. Here, the data bit establishes the phase of the carrier.

Removing the diode from the ASK modulator results in phase shift keying modulation (PSK). The threshold voltage is returned to 0V. The PSK signal is observed using the oscilloscope as it is generated and detected.

Laboratory Procedure

A) Baseband Transmission:

The circuit is built according to the schematic shown in figure 3A. The data rate (fs) is set for 5 kHz. The Frequency counter is set to DC coupling and the auto level is set to OFF. The module is monitored using the oscilloscope.

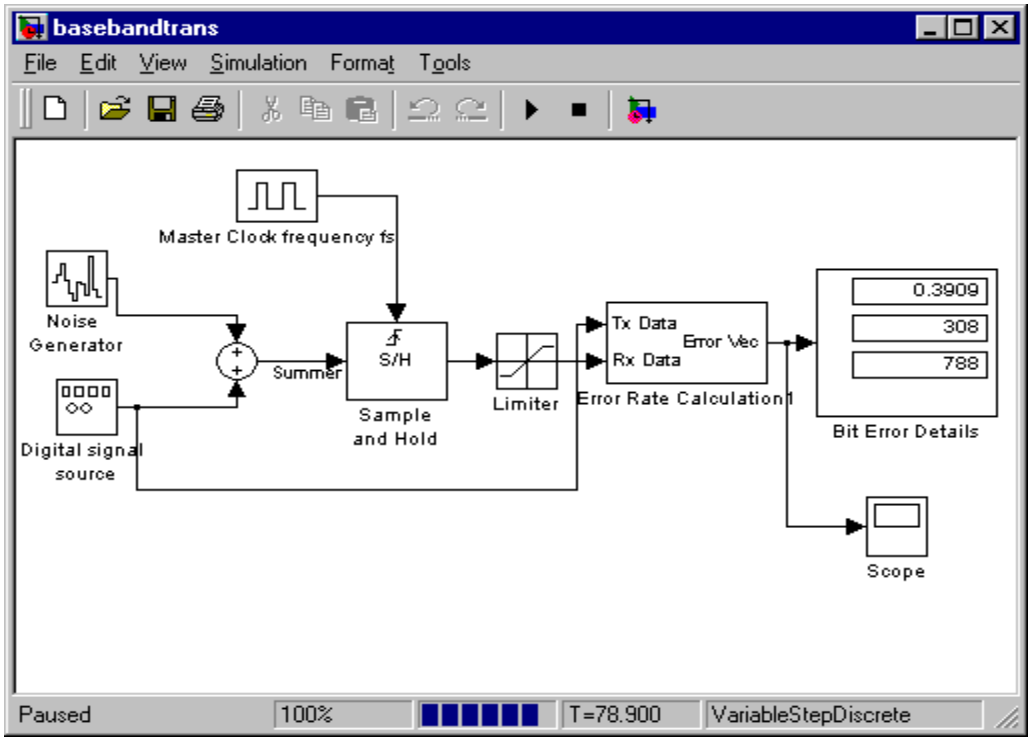


Figure 3 A (a) Base Band transmission

The functions of the different modules are explained below:

Digital Signal Source:

The signal sources produce a series of "0"s (about –2.5 V) and "1"s (+2.5 V) at the data rate set by the Master Clock. The time period of each bit is T = 1/fs. This signal is useful in visually examining the detection process. This signal is useful in accurately measuring errors in detected data due to a communications channel.

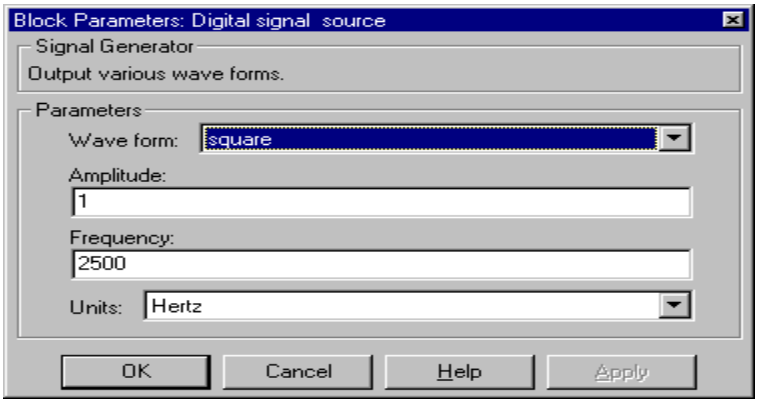


Figure 5 A (b) Block parameters digital signal source

Summer:

The summer represents the communications channel in which random noise is added from the signal analyzer's SOURCE OUT output. The spectrum level of the random noise, No, can be measured by connecting the SOURCE OUT to the INPUT of the signal analyzer, and setting the SOURCE to RANDOM NOISE with 0dB attenuation. The spectrum is averaged by setting 50 averages. By selecting RMS and pressing the START button, the spectrum will be averaged with 50 samples. This average was close to uniform at a level of about 150mV. The square of this level, gives the noise power. No, The relative power spectral density in W/Hz, is determined by dividing the noise power by the bandwidth.

No = 1.24exp(-4). Watts/ Hz. {measured value}

Integrator:

This is an integrate and dump circuit with the dump discharging the integrating capacitor and occurring at the end of the data period. This provides an optimal detection filter for the signal. The OUT connector provides a direct view of the integration. While an integration is occurring, the previous integration value is stored and is available at the sample and hold connector. The integrator is basically a low pass filter in frequency domain.

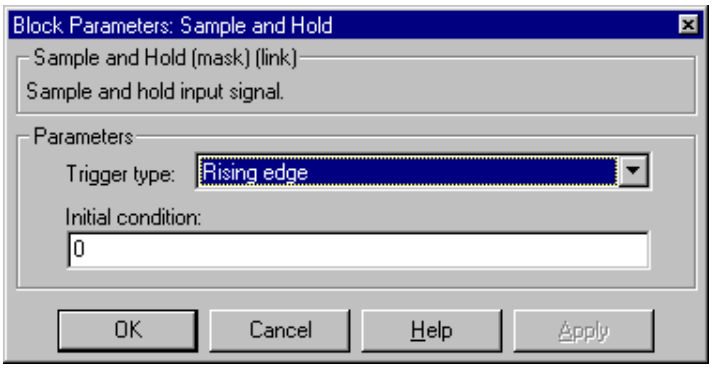


Figure 3 A (c) Block parameters Sample and hold

Limiter:

This is the decision-making device. If the output of the integrator is grater than zero, the limiter output is about +2.5 V. The choice of zero as a threshold voltage is valid since the signal source produces equally likely 0's and 1's.

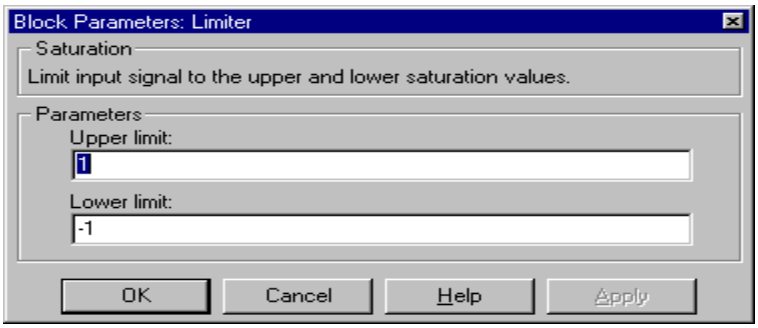


Figure 3 A (d) Block parameters limiter

Error Detector:

The processed signal is compared with a sample of the signal source. For each data bit that does not match, a 10s pulse appears at the ERROR connector. Measuring the average number of pulses per second gives a Bit Error Rate (BER). The received data may be delayed during processing, so a delay is needed in the error detector to compensate. For processing delays <1 bit, the Bit Delay Switch is set to "0". For delays >1 bit and < 2bits, the Bit Delay Switch is set to "1".

Next, the effects of changing the noise level were observed. By changing the attenuation level in the spectrum analyzer, different levels of noise were generated. We observed how the bits became more masked by the noise as the noise increased (i.e., SNR decreases)

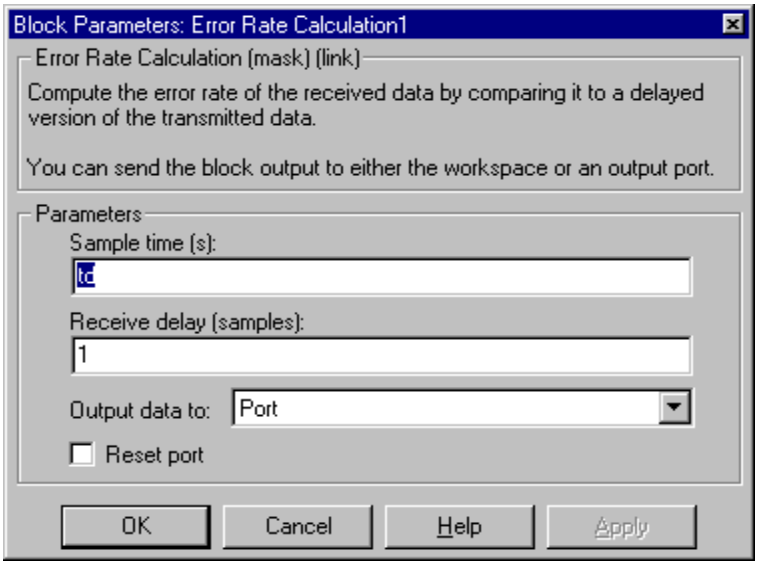


Figure 3 A (e) Block parameters Error rate calculations

The output looks like this when it is mixed with the input signal

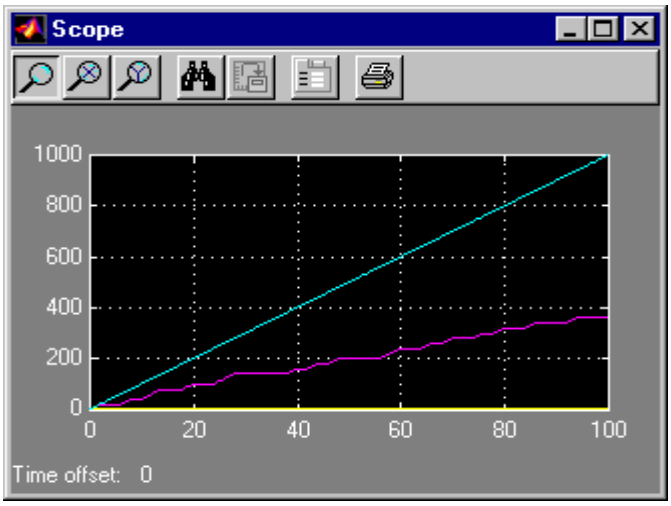


Figure 3 A (f) Scope output

B) The effect of following changes on BER was observed:

1) Changing The Noise Level

If we changed the noise level as the attenuation decreased that is as the noise power increased the bits were masked more and more by noise. Therefore as the noise level increased the chances of error also increased.

For Attenuation of 10 dB

BER for bit delay (1) = 0.0 KHz

BER for bit delay (0) = 5.0 KHz

2) Changing The Data Rate

Data Rate of 8KHz attenuation 0 dB

BER for bit delay (1) = 0 Hz

BER for bit delay (0) = 8 KHz

3) Bypassing the Integrator

BER bit delay of (1) = 5 KHz

BER bit delay of (0) = 0 KHz

C) The probability of error (PE) may be estimated by an average BER and dividing it by the data rate. Therefore PE = BER / fs; Eb = 7.27×10^{-5}

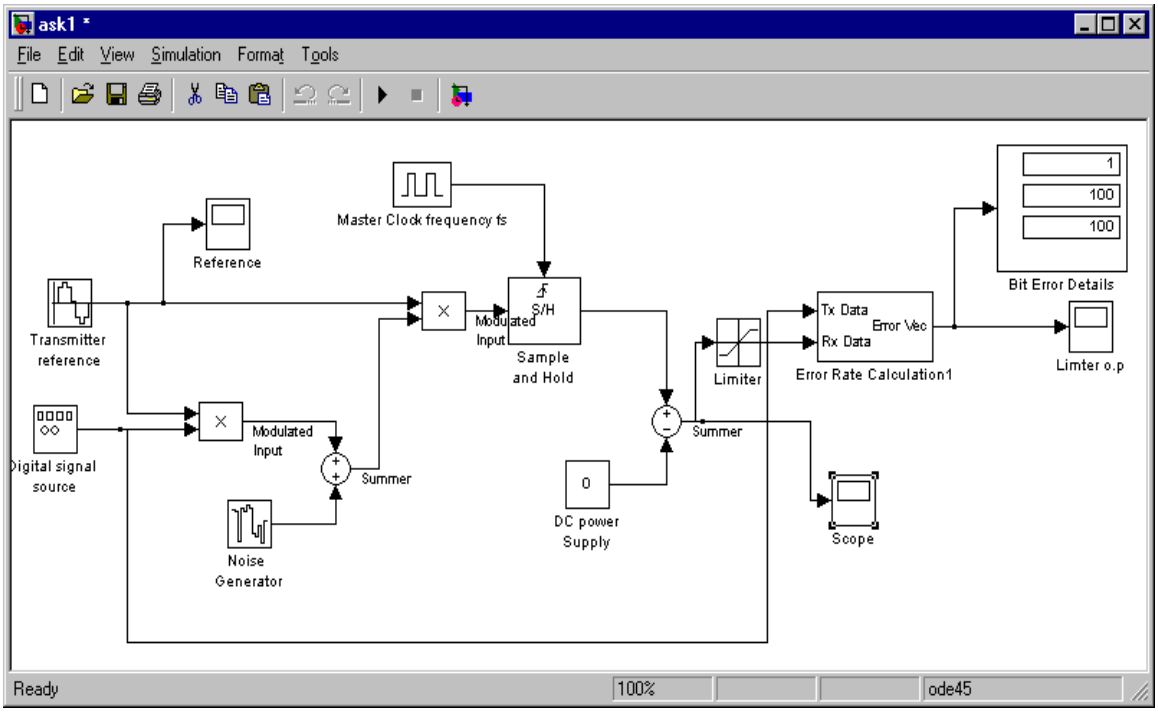


Figure 3 B (a) ASK BER measurement

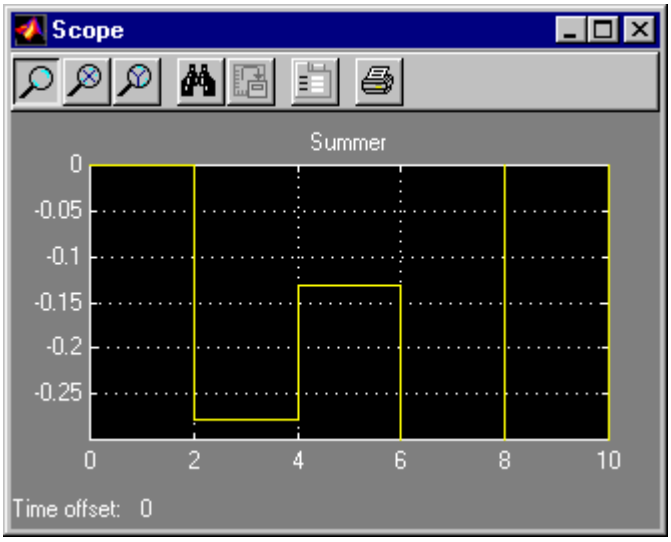


Figure 3 B (a) ASK Ouptut of the summer

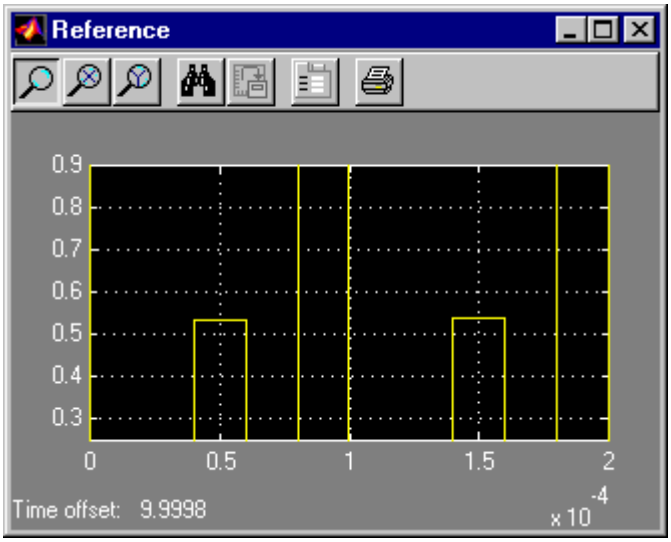


Figure 3 B (c) ASK reference signal

The output of the limiter looks like this

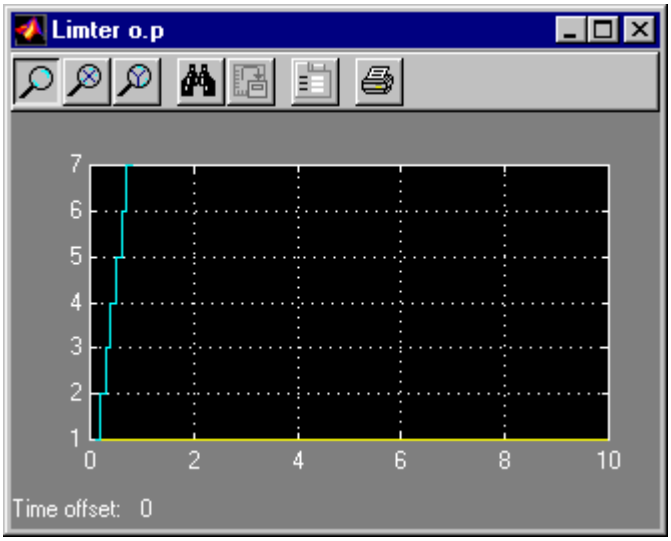


Figure 3 B (a) Limiter output

The noise generator is a white noise for continious systems, band limited using zero-order-hold.

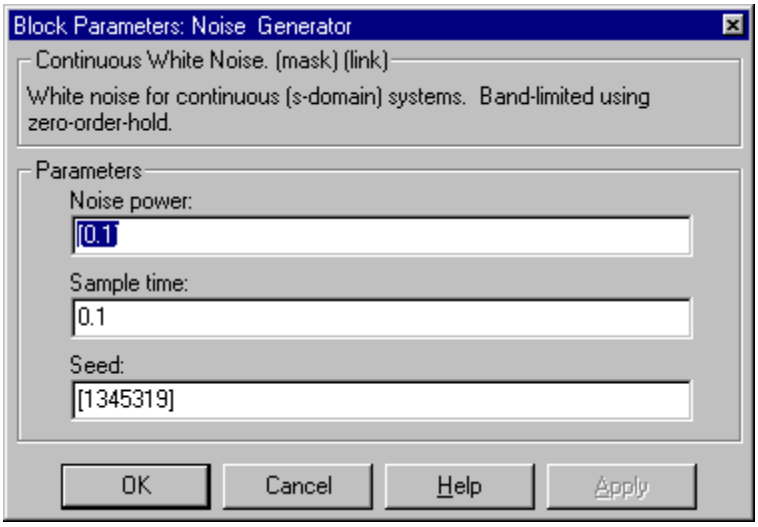


Figure 3 B (a) ASK block parameters Noise generator

II. ASK Transmission (on-off Keying)

The circuit is built as shown in figure below. The input sine wave is multiplied by a pulse generator, which can be used as a digital data bit.

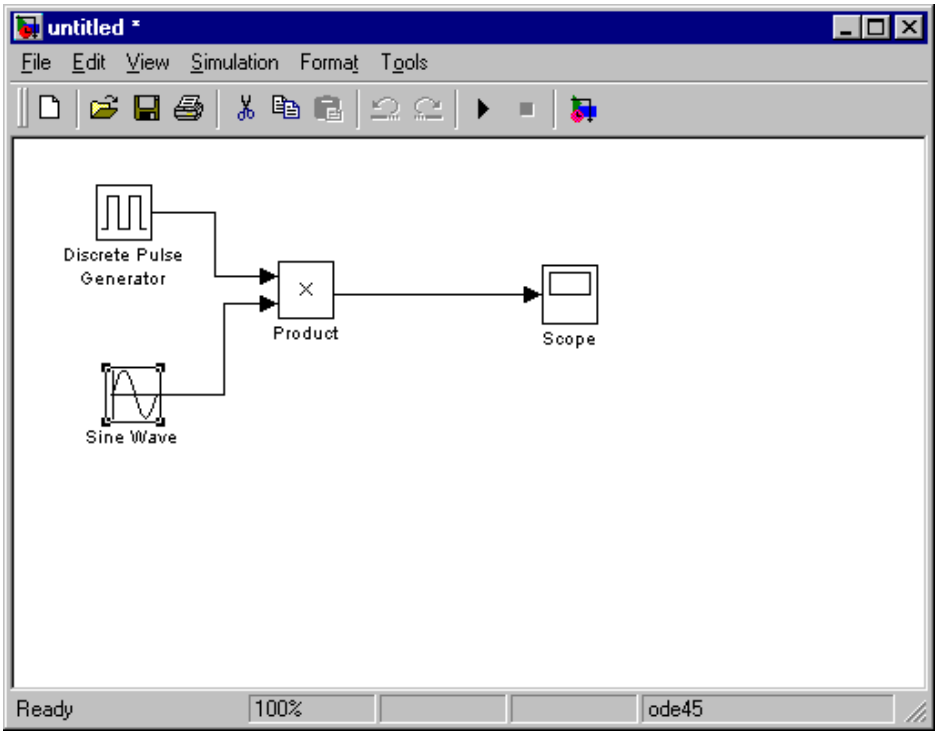


Figure 3 B –II (a) ASK On-off keying implementation

We can see that the carrier modulates the data bits and the output waveform is shown

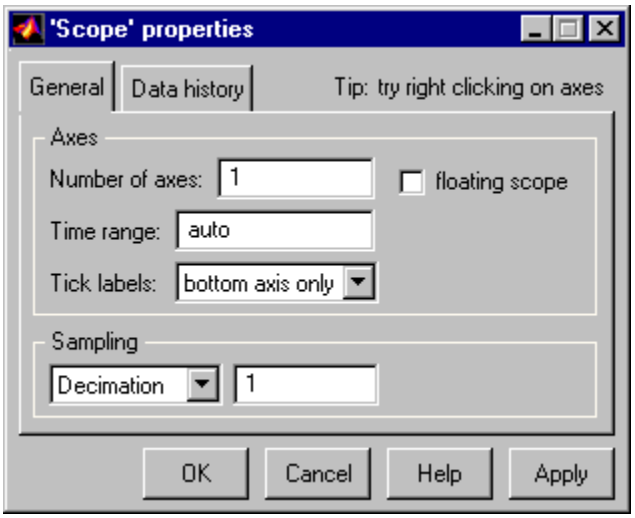


Figure 3 B –II (b) Scope properties

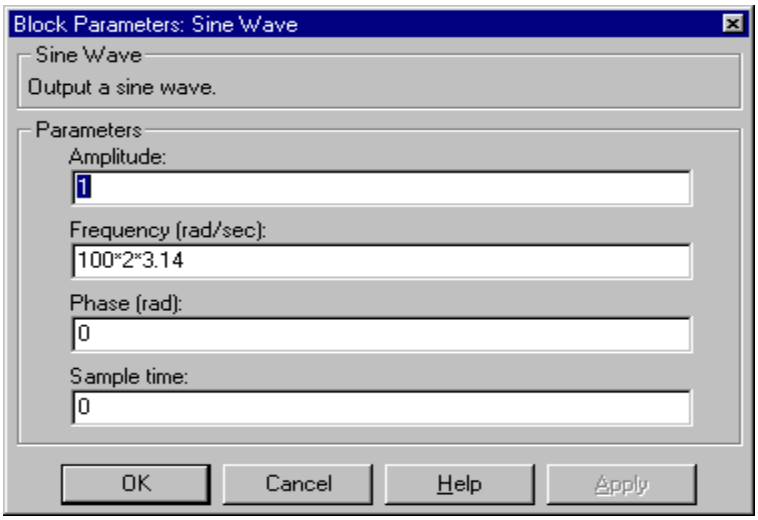
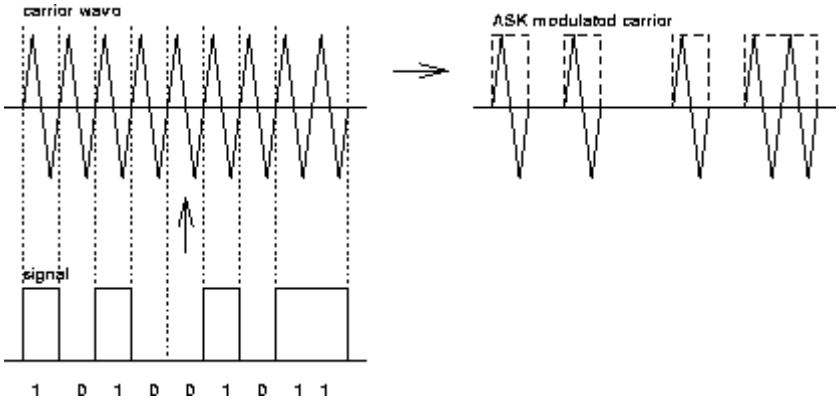


Figure 3 B –II (c) block properties of the signal generator



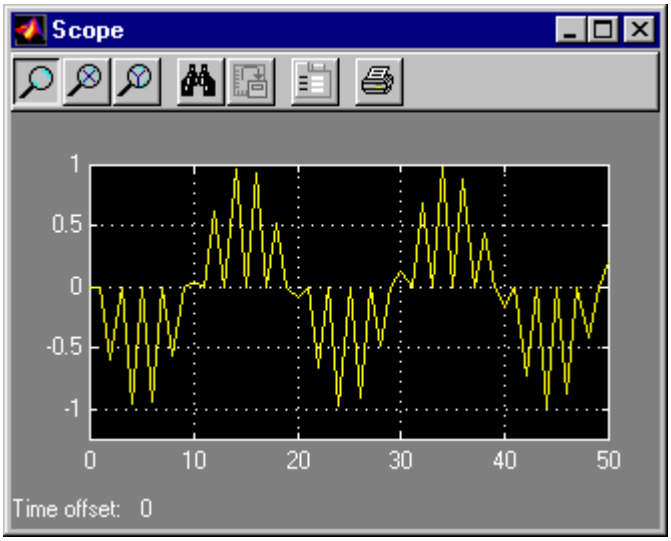


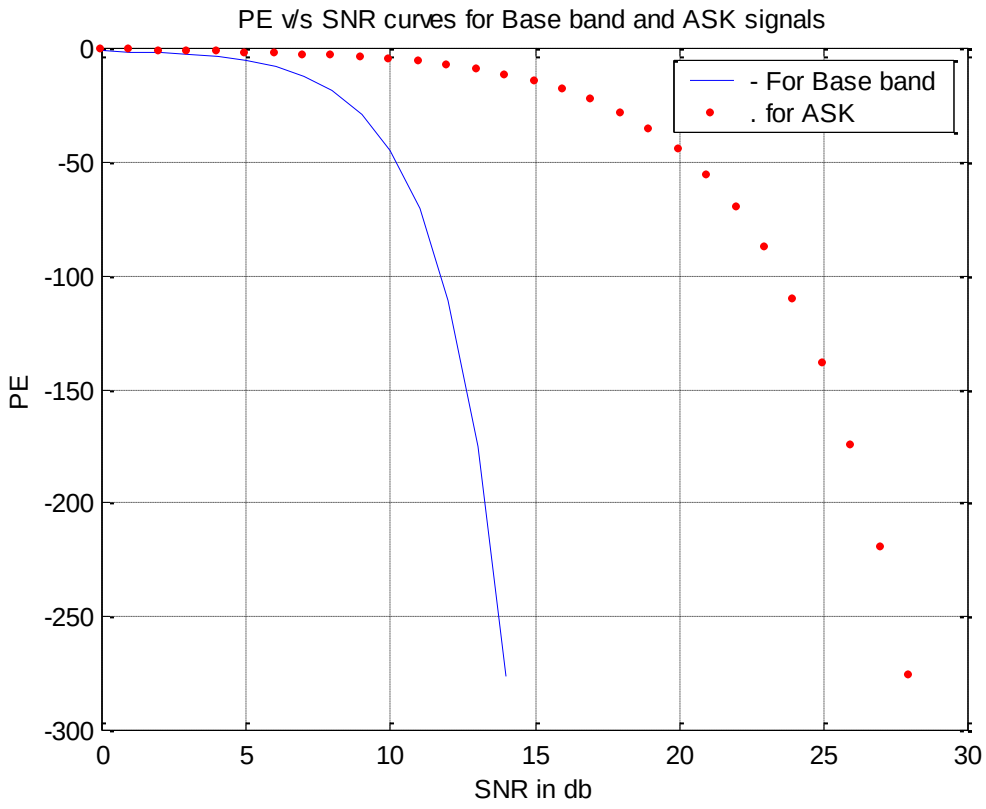
Figure 3 B –II (d) Output of the scope the ASK on-off signal

Prelab Assignment

Plot of PE vs SNR curves for Base Band and ASK signals is shown below

```
clear all; close all;
x = 0:1:30;
PEbb = 0.5*erfc(sqrt(power(10,0.2*x)));
PEask = 0.5*erfc(sqrt(power(10,0.1*x)));
```

```
plot(x,log10(PEbb)); hold on
plot(x,log10(PEask),'r.');
grid on; zoom on;
title('PE v/s SNR curves for Base band and ASK signals');
xlabel('SNR in db'); ylabel('PE');
legend('- For Base band','. for ASK');
```



Computer Exercise IV: Digital Modulation

Objectives: In this experiment you will apply concepts of baseband digital transmission and analog continuous wave modulation to the study of band-pass digital transmission. You will examine:

- generation of digital modulated waveforms;
- coherent (synchronous) and noncoherent (envelope) detection of modulated signals;
- system performance in the presence of corrupting noise.

I- Prelab Assignment:

1. Consider the binary sequence b = [1 0 0 1 0]. Let the bit rate R_b be 1 kbps and let the peak amplitude of all digital modulated waveforms be set to 1 V.
(a) Sketch the ASK waveform representing the binary sequence b using a carrier frequency of 5 kHz.
(b) Sketch the PSK waveform representing the binary sequence b using a carrier frequency of 5 kHz.
(c) Let the mark and space frequencies used by an FSK modulator be set to 3 and 6 kHz, respectively. Sketch the resulting FSK waveform representing the binary sequence b.
2. Sketch the power spectral density function for each of the modulated signals in Question 1.
3. If an ASK signal is applied to the input of a coherent detector shown in Fig. 4.1, sketch the waveforms at the output of each block.

II. Procedure:
In this experiment, the binary data rate R_b is 1 kbps and peak modulated signal amplitude is 1 V. The bit period $T_b = 1/R_b$ is represented by 100 samples.

A. Generation of Modulated Signals

***Amplitude-Shift Keying (ASK)**

A.1 Generate a binary sequence with the first 5 bits [1 0 0 1 0].

A.2 To generate the ASK signal, sa, with a carrier frequency of 8 kHz:
- generate a unipolar NRZ signal xu, from the sequence b;
- mix xu with the output of an oscillator operating at 8 kHz.

A.3 Display the first 5 bits of xu and sa in the binary sequence b. Compare the two waveforms. Also display and record the respective PSD functions over the frequency interval [0, 20kHz].

***Phase-Shift Keying (PSK)**

A.4 To generate the PSK signal sp, with a carrier frequency of 8 kHz:

- generate a polar NRZ signal xp, from the sequence b;
- mix xp with the output of an oscillator operating at 8 kHz.

A.5 Display the first 5 samples of the waveforms xp and sp. What is the phase difference between sp and the carrier $\sin(2\pi f_c t)$ during the first and second bit periods?

A.6 Display the PSD functions of xp and sp over the frequency interval [0, 20 kHz]. Record main characteristics of each PSD function.

***Frequency-Shift Keying (FSK)**

A.7 To generate the continuous phase FSK signal sf, with mark and space frequencies of 4 and 8 kHz, respectively:

- generate a polar NRZ signal from the sequence b;
- mix xp apply the polar waveform to the input of a voltage controlled oscillator (VCO). In this experiment the VCO has the free-running frequency set to 6 kHz and has frequency sensitivity of -2 kHz/V.

A.8 Display waveforms xf and sf for $0 < t < 5\ T_b$. Display and record the PSD function of the FSK signal. How can you generate an FSK signal from two ASK signals? For a system where efficient bandwidth utilization is required, which modulation scheme would you prefer?

B. Coherent and Noncoherent Detection

***Coherent Detection**

B.1 A coherent detector for ASK and PSK signals is depicted in Fig. 4.1.

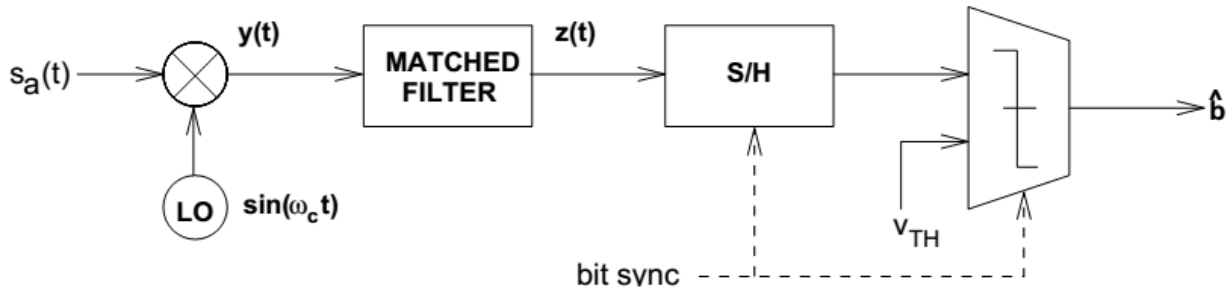


Fig. 4.1 Coherent Detector

To demodulate the ASK signal sa, first multiply sa by a locally generated carrier which has the same frequency and phase as the carrier used in generating sa. Display the waveform ya at the output of the multiplier for the first five bit periods. Also display and record the corresponding PSD function over the interval fr.

B.2 Apply ya to a matched filter and record its output for $0 < t < 5\ T_b$ .Determine the impulse response of the matched filter. Note that za is similar to the output of the matched filter for a unipolar NRZ signal. Why?

B.3 The major difficulty in implementing a coherent detector is carrier synchronization. In order to achieve optimum performance, the local oscillator should have the same phase and frequency as the incoming carrier. Phase or frequency deviation will result in degradation of detection performance.

To observe the effect of phase error, demodulate sa using a local oscillator whose output is $\sin(2\pi f_c + \varphi)$. Here, φ is the phase error measured with respect to the carrier. Record the peak signal amplitude at the matched filter output for each phase error shown in Table 4.1.

Table 4.1

Phase Error	Peak Amplitude [V]
0°	
20°	
60°	
80°	
120°	

Recall that the BER resulting from the detection of a signal in the presence of noise, is a function of peak signal amplitude at the receiver filter output. Determine from the results displayed in Table 4.1 which phase error will result in smallest BER.

B.4 Demodulate sa with 60° and 120° phase errors. Decode the matched filter output to recover the first five bits of the sequence b. Record each decoded sequence and comment on the difference.

B.5 To observe the effect of frequency deviation in demodulating an ASK signal, demodulate sa with a local oscillator set to 7,900 Hz. Display and compare the demodulated signals ya and ya1.

Could the original binary sequence be recovered from ya1? Consider a second case where the local oscillator frequency is set to 7,985 Hz. Demodulate sa and generate the matched filter output ya2. Determine the frequency of the envelope of the matched filter output.

Now consider an ASK signal $s_a(t)$ with carrier frequency of fc . If $s_a(t)$ is demodulated by multiplying with the output of a local oscillator set to f_o, such that $f_o \neq f_c$, the envelope of detector matched filter output is modulated by a sinusoid. Determine the frequency of this modulating signal as a function of f_c and f_o.

C. System Performance Under Noise

Coherent Detection

C.1 Generate an ASK signal b representing a 500-sample binary sequence with first five bits '1 0 0 1 0'

C.2 Apply sa to a channel with unity gain, channel noise σ_n^2= 1 W, and of sufficient bandwidth such that no distortion is introduced to the signal. Display the ASK signal sa and the channel output y for $0 < t < 5T_b$.

C.3 Use a coherent detector to demodulate y. Display the eye diagram of the matched filter output zm. From the eye diagram, determine optimum sampling instants and the threshold value.
Apply zm to the decision circuit, and record the resulting probability of bit error.

Compute the theoretical probability of bit error for the case considered above. Recall that the PSD function of the channel noise is

$$S_n(f) = \frac{N_o}{2} = \frac{\sigma_n^2}{2 \times System\ Bandwidth}$$

The system bandwidth in this experiment is 50 kHz.

.

Computer Exercise V: Matched Filter and Bit Error Rate (BER)

Objectives: In this experiment you will investigate the signal detection process by studying elements of a receiver and of the decoding process. In particular you will:

- investigate the characteristics of matched filters;
- study performance of various receiver structures based on different receiver filters by measuring probability of bit error;
- use the eye diagram as an investigative tool to set parameters of the detection process.

I- Prelab Assignment:

A: A matched filter is to be designed to detect the rectangular pulse

$$r(t) = rect\left(\frac{t - T_b/2}{T_b}\right) \quad , \quad T_b = 1 \text{ msec}$$

(a) Determine the impulse response of the matched filter.
(b) Determine the output of the matched filter if *r(t)* is the input.
(c) Repeat parts (a) and (b) for a triangular pulse of 10 msec duration.

B: Let y (t) = x (t) + n(t), represent the waveform at the output of a channel. x (t) is a polar NRZ waveform with unit pulse amplitude and binary data rate R_b of 1 kbps. n(t) is a white noise process with PSD function:

$$S_n(f) = N_o/2 = 0.5 \times 10^{-4} \text{ W/Hz.}$$

If y (t) is applied to a matched-filter receiver:
(a) Determine the rms value of n(t) and the peak signal amplitude at the output of the matched filter.
(b) Determine E_b , the average energy of x (t) in a bit period.
(c) Determine the probability of bit error $P_e = Q\left(\sqrt{2E_b / N_o}\right)$.

C- If y (t) in Question B is applied to a RC-filter with frequency response:

$$H_{rc}(f) = \frac{1}{1 + j2\pi fRC}$$

with $RC = 1/(2000\pi)$,
(a) Determine the peak signal amplitude and rms value of the noise at the filter output;
(b) Determine the probability of bit error P_e, if x(t) were to be detected by a receiver based on the RC-filter.

II- Procedure:

A. Characteristics of Matched Filters

A.1 Generate a rectangular pulse r with unit pulse amplitude and 1 msec pulse duration.

A.2 Display *r* and the impulse response of a matched filter based on *r*.
A.3 Observe the matched filter output (r_m) if *r* is applied to its input. Then, determine the time when the filter output reaches its maximum value. How is this time related to the waveform *r*?

A.4 Repeat parts A.1–A.3 for a triangular pulse with 10 msec pulse width and unit peak amplitude. If the triangular pulse width is changed to 1 msec, determine the peak amplitude of the matched filter output?

A.5 Repeat parts A.1–A.3 for a Manchester pulse with 10 msec pulse width and unit peak amplitude. Predict the matched filter impulse response and matched filter output. Verify your predictions using MATLAB functions.

A.6 Generate a polar NRZ waveform (x5) that represents the 5-sample binary sequence [1 0 0 1 0]. The binary data rate R_b is 1 kbps and the pulse amplitude *A* is 1 V. Record the waveform x5.

A.7 Apply x5 to a matched filter. Record output. Construct the waveform at a matched filter output if the input is a unipolar NRZ waveform that represents the binary sequence [1 0 0 1 0].

B. Signal Detection

B.1 Generate a 10-sample binary sequence (b10) and a waveform (x10) that represents this binary sequence in polar NRZ signalling format.

B.2 Apply x10 to a channel with 4.9 kHz bandwidth and AWGN where the noise power is 2W. Display the channel output waveform y10. Then, decode the binary sequence from the waveform $\widehat{y10}$.

B.3 Apply y10 to a matched filter. Display the output waveform z10.

B.4 Let T_b be the binary data period. Sample the output of the matched filter at kT_b, k =1, . . . , 10 and apply the following decision rule:

$$b_k = \begin{cases} 0, \ if \ \ sample\ value \ > \ 0; \\ 1, \ if \ \ sample\ value \ < \ 0; \end{cases}$$

where $\hat{b}_k$ is the estimated value of the *k*-th element of the binary sequence b10. Apply this decision rule on the matched filter output z10. Compare your decoded sequence with the original sequence b10. Comment on whether it is easier to decode the transmitted binary sequence directly from the channel output y10 or from the matched filter output z10. If sampling instants other than those specified above are used, the probability of making a decoding error will be larger. Why?

C. Matched-Filter Receiver

C.1 Generate a 2,000-sample binary sequence b and a polar NRZ waveform x based on b.

Apply x to a channel with 4.9 kHz bandwidth and channel noise power of 0.5 W. Let y be the channel output waveform.

C.2 Apply y to a matched filter. Display the eye diagram of the matched filter output z. From the eye diagram, determine the optimum sampling instants and threshold value v_th for the detector to decode the transmitted binary sequence b. Sampling instants for the matched filter output are measured with respect to the time origin. For example, if the binary data period is T_b and the sampling instant parameter is set to t_i , then the detector will sample the signal at t_i , $t_i + Tb$, $t_i + 2\ Tb$, . . . etc. Use v_th and sampling instant in the detector which will operate on the matched filter output. Record the resulting probability of bit error P_e (BER) in Table 3.1.

Table 3.1

σ_n^2[W]	P_e empirical	P_e theoretical
0.5		
1		
1.5		
2		

C.3 Repeat C.1–C.2 for channel noise power of 1, 1.5, and 2 W without displaying the eye diagram of the matched filter output z. Record P_e results in Table 3.1.
Remark: In Experiment 2 you have observed that the optimum sampling instants and the threshold value are independent of channel noise power. Therefore, you can use the optimum sampling instants determined in part C.2 to decode the matched filter output for different channel noise power levels.

C.4 If different sampling instants other than the optimum values are used, the resulting BER will be larger. You can observe this by decoding the binary sequence using values for the sampling instant parameter that are 0.9 and 0.5 times the optimal value used in part C.3. Also, evaluate theoretical probability of bit error values for all cases considered above and record in Table 3.1. Note that the PSD function of a white noise process can be as:

$$S_n(f) = \frac{N_o}{2} = \frac{\sigma_n^2}{2 \times System\ Bandwidth}$$

where the system bandwidth in this experiment is 4.9 kHz.

D. Low-Pass Filter Receiver

D.1 Apply a rectangular pulse to a first-order RC-filter of 1 kHz bandwidth. Display the filter output and measure the peak amplitude A_r .

D.2 Generate 2,000 samples from a zero-mean white noise sequence of 0.5 W power. Apply the noise sequence to the RC-filter. Record the rms value of the output noise power σ_n^2. From the results in parts D.1 and D.2, determine the ratio A_r, where A_r is the peak signal amplitude measured

in D.1 and σ_n is the rms value of the output noise. If y in part C.1 is applied to a receiver which uses the above RC-filter, determine the resulting BER.

D.3 Regenerate y from part C.1. Apply y to the RC-filter. Display the eye diagram of the output waveform z_lpf.

D.4 From the eye diagram, determine the optimum sampling instant and threshold value. Decode the binary sequence form z_lpf. Compare the resulting BER with the BER evaluated in step C.2.

D.5 Repeat part D.4 for the channel noise power of 1, 1.5, and 2 W. Record results in Table 3.2.

Table 3.2

σ_n^2[W]	P_e @ BW = 1.0 kHz	P_e @ BW = 0.5 kHz
0.5		
1		
1.5		
2		

D.6 Repeat parts D.3 – D.5 for a first-order RC-filter with 500 Hz bandwidth. Record the resulting BER values in Table 3.2. Explain why the BER resulting from a low-pass filter of 500 Hz bandwidth is smaller than the BER resulting from a low-pass filter of 1 kHz bandwidth. Will the BER be further decreased if a low-pass filter of 100 Hz bandwidth is used?

Computer Exercise VI : Equalization

In this assignment, you will design various receivers for a nontrivial (real) ISI channel, and compare their performance with theoretical bounds.
The channel of interest has the following length L = 11 impulse response:

$$h[n] = [\ -0.8\ \ 0.6\ 0.002\ 1.05\ \ -0.43\ \ 0.19\ \ 0.512\ \ -0.005\ \ -0.2\ 0.3\ \ -0.01\ 0.085];$$

for $n \geq 0$. For the below equalizers, you can use a receiver delay of $\Delta = (N_f + L)/2$. This will in general be quite close to optimum, but you are certainly encouraged to experiment to find the optimum Δ for each equalizer. For this channel using MPSK/MQAM modulation, after some normalizing, $E_{av} = 1$ and $\sigma^2 = 0.1$. Also, it is assumed that sampling is at the symbol rate (i.e. there is no "oversampling").

(a) Plot h[n] and $|H(\omega)|$. (b) What is SNR for this system?

(c) Design equalizers for the following scenarios. Also, for each case plot the equalizer response in both the time and frequency domains and determine the attained SNR. Further, plot the scatter plots at the transmitter and the receiver-before and after equalization.

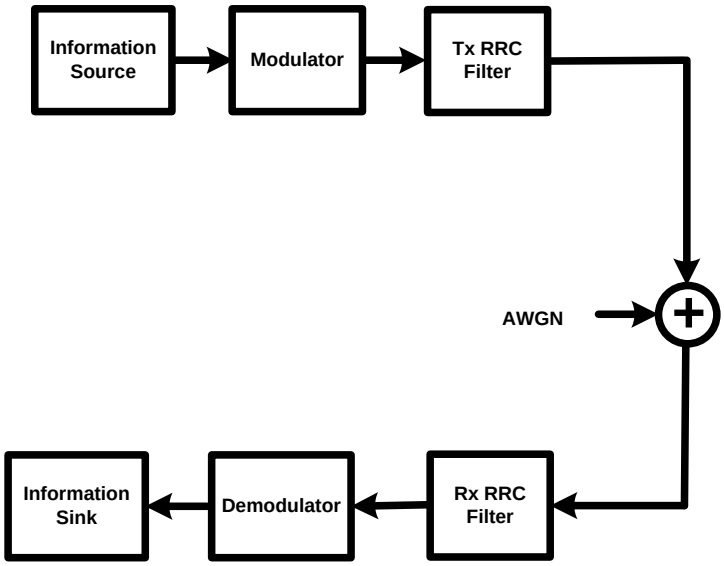
(i) A ZF-LE i.e. w_{ZFE} with $N_f = 24$. You can use either an FIR approach or a pseudoinverse approach to minimize the convolution of w_{LS} and h. The scaling will be slightly different (by a factor of $||h||$) for each, but this will not affect the output SINR.
(ii) The MMSE-LE i.e. w_{MMSE} with $N_f = 24$.
(iii) Now design a FIR MMSE-DFE equalizer with $N_f = 14$ and $N_b = 10$.

(d) Now, let's compare the performance of these FIR equalizers experimentally for $E_x = 1$ and $0.1 \leq \sigma^2 \leq 1$. You will need to simulate 1,000,000 bits and the corresponding noise, and find the probability of bit error experimentally and plot it as a function of σ^2 (a log scale might be best for both axes).
(i) No equalizer. In other words, treat all ISI as noise. Compare this to the mean-square distortion result.
(ii) ZF-LE FIR (iii) MMSE-LE FIR (iv) MMSE-DFE FIR

On the same plot, compare your simulated receiver error performance with the theoretical BER results for those receivers.

Note: You must turn in all of your code in addition to clearly marked answers to the above. You are welcome to have your MATLAB code merely print out the above answers, and then clearly highlight and label them. Your code must be your own - the goal of this assignment is for you to learn how to design these types of equalizers. You can of course confer with your classmates and refer to previously-developed modules for equalization - as long as you clearly develop your own solution.

Sample program 1: Baseband Digital Communication System



Block diagram of baseband digital communication system

```
%This program simulates a narrowband digital communication system
%including source, M-PSK modulator, and RRC filter in the
%presence of an AWGN channel using MATLAB R2010.a. Outputs are BER and PER.

clear all
tic

M=input('Modulation order M:');
if isempty(M);
   M=4;
   disp([num2str(M)]);
end
k = log2(M);% Number of bits per data  symbol
EbNo_dB=input('Enter Eb/No in dB [min:step:max]:');
if isempty(EbNo_dB);
   EbNo_dB=[0:2:10];
   disp([num2str(EbNo_dB)]);
end
seed = [sum(100*clock) 123456];
nPacket=input('Enter number of packets  ');
if isempty(nPacket);
   nPacket=100;
   disp([num2str(nPacket)]);
end
```

```
nBpPacket=input('Enter number of bits per packet ');
if isempty(nBpPacket);
   nBpPacket=1000;
   disp([num2str(nBpPacket)]);
end
nBit=nBpPacket*nPacket;
nSym=nBit/k;

PER=zeros(size(EbNo_dB));% Preallocate space for PER
BER= zeros(size(EbNo_dB));% Preallocate space for BER

% ----------Create a modulator object and display its properties.
hMod = modem.pskmod('M', M, 'PhaseOffset', 0, 'SymbolOrder','gray','InputType','Bit');
hDemod = modem.pskdemod(hMod);

%---------- RRC filter
OSR =4;
Rolloff = 0.22; % Roll-off factor
FiltOrder=64; % Filter order
D=FiltOrder/(OSR*2); % Group delay (# of input samples)

H=waitbar(0,'Cal');

%------------------- Start the main program ------------------

for n_ebno = 1:length(EbNo_dB) %********%Start of the  loop over Eb/No
  error=0; per=0;
for n_pa = 1:nPacket %********%Start of the  loop over nPacket

waitbar(n_ebno/length(EbNo_dB),H,'Oops!')

%----------Data generation
mtb=randint(nBpPacket,1);

%----------Modulation
mts_mod= modulate(hMod,mtb);

%----------Tx RRC Filter
mts_flt = rcosflt(mts_mod,1,OSR,'sqrt/fir',Rolloff,D);% Upsample and apply square root

%----------RF Channel
SNRdB = EbNo_dB(n_ebno) + 10*log10(k/OSR);
ms_noisy=awgn(mts_flt,SNRdB, 'measured', [], 'dB');

%----------Rx RRC Filter
mrs_flt = rcosflt(ms_noisy,1,OSR,'Fs/sqrt/fir',Rolloff,D);
mrs_flt = downsample(mrs_flt,OSR); % Downsample
mrs=mrs_flt(2*D+1:end-2*D,:);

%----------Demodulation
mrb=demodulate(hDemod,mrs);
```

```
%-----------BER/PER Calculations
error=sum(xor(mtb,mrb))+error;
  if sum(xor(mtb,mrb))~=0
    per=per+1;
  end
end % End of loop over number of frames
BER(n_ebno)=error/nBit;
PER(n_ebno)=per/nPacket;

end % End of loop over Eb/No

close(H)
save BER,PER

%-----------Plots
semilogy(EbNo_dB,BER,'r*-');%Plot simulated BER
xlabel('Eb/No[dB]');ylabel('Error Rate'); grid on;
axis([0 EbNo_dB(end) 1e-6 1])
EbNo_theo=[0:.1:EbNo_dB(end)];
linEbNo_theo=10 .^(0.1 .*EbNo_theo);%
ber= berawgn(EbNo_theo,'psk',M,'nondiff');
hold on
semilogy( EbNo_theo,ber,'b-');%Plot theoritical
figure
semilogy(EbNo_dB,PER(1:length(EbNo_dB)),'b*-');%Plot simulated PER
xlabel('Eb/No[dB]');ylabel('Packet Error Rate'); grid on;
axis([0 EbNo_dB(end) 1e-3 1])
toc
```

Sample program : Equalization

This section gives an overview of the process you typically use in the MATLAB environment to take advantage of the adaptive equalizer capabilities.

Basic Procedure for Equalizing a Signal

Equalizing a signal using Communications Toolbox software involves these steps:

1. Create an equalizer object that describes the equalizer class and the adaptive algorithm that you want to use. An equalizer object is a type of MATLAB variable that contains information about the equalizer, such as the name of the equalizer class, the name of the adaptive algorithm, and the values of the weights.
2. Adjust properties of the equalizer object, if necessary, to tailor it to your needs. For example, you can change the number of weights or the values of the weights.
3. Apply the equalizer object to the signal you want to equalize, using the `equalize` method of the equalizer object.

Example Illustrating the Basic Procedure

This code briefly illustrates the steps in the basic procedure above.

```
% Build a set of test data.
x = pskmod(randint(1000,1),2); % BPSK symbols
rxsig = conv(x,[1 0.8 0.3]);  % Received signal
% Create an equalizer object.
eqlms = lineareq(8,lms(0.03));
% Change the reference tap index in the equalizer.
eqlms.RefTap = 4;
% Apply the equalizer object to a signal.
y = equalize(eqlms,rxsig,x(1:200));
```

In this example, `eqlms` is an equalizer object that describes a linear LMS equalizer having eight weights and a step size of 0.03. At first, the reference tap index in the equalizer has a default value, but assigning a new value to the property `eqlms.RefTap` changes this index. Finally, the `equalize` command uses the `eqlms` object to equalize the signal `rxsig` using the training sequence `x(1:200)`.

Choosing an Adaptive Algorithm

Configuring an equalizer involves choosing an adaptive algorithm and indicating your choice when creating an equalizer object in the MATLAB environment.

Although the best choice of adaptive algorithm might depend on your individual situation, here are some generalizations that might influence your choice:

- The LMS algorithm executes quickly but converges slowly, and its complexity grows linearly with the number of weights.
- The RLS algorithm converges quickly, but its complexity grows with the square of the number of weights, roughly speaking. This algorithm can also be unstable when the number of weights is large.
- The various types of signed LMS algorithms simplify hardware implementation.
- The normalized LMS and variable-step-size LMS algorithms are more robust to variability of the input signal's statistics (such as power).
- The constant modulus algorithm is useful when no training signal is available, and works best for constant modulus modulations such as PSK.

  However, if CMA has no additional side information, it can introduce phase ambiguity. For example, CMA might find weights that produce a perfect QPSK constellation but might introduce a phase rotation of 90, 180, or 270 degrees. Alternatively, differential modulation can be used to avoid phase ambiguity.

Two typical ways to use a function from the table are as follows:

- Use the function in an inline expression when creating the equalizer object.

  For example, the code below uses the `lms` function inline when creating an equalizer object.

  ```
  eqlms = lineareq(10,lms(0.003));
  ```

- Use the function to create a variable in the MATLAB workspace and then use that variable when creating the equalizer object. The variable is called an *adaptive algorithm object*.

  For example, the code below creates an adaptive algorithm object named `alg` that represents the adaptive algorithm, and then uses `alg` when creating an equalizer object.

  ```
  alg = lms(0.003);
  eqlms = lineareq(10,alg);
  ```

To create an equalizer object, use one of the `lineareq` and `dfe` functions. For example, the code below creates three equalizer objects: one representing a symbol-spaced linear RLS equalizer having 10 weights, one representing a fractionally spaced linear RLS equalizer having 10 weights and two samples per symbol, and one representing a decision-feedback RLS equalizer having three weights in the feedforward filter and two weights in the feedback filter.

```
% Create equalizer objects of different types.
eqlin = lineareq(10,rls(0.3)); % Symbol-spaced linear
eqfrac = lineareq(10,rls(0.3),[-1 1],2); % Fractionally spaced linear
eqdfe = dfe(3,2,rls(0.3)); % DFE
```

Although the `lineareq` and `dfe` functions have different syntaxes, they both require an input argument that represents an adaptive algorithm.

An equalizer object has numerous properties such as the structure of the equalizer, the adaptive algorithm and the information about the equalizer's current state which can be viewed or changed properly. The example below illustrates that when you change the value of `nWeights`, MATLAB automatically changes the values of `Weights` and `WeightInputs` to make their vector lengths consistent with the new value of `nWeights`. Because the example uses the variable-step-size LMS algorithm, `StepSize` is a vector (not a scalar) and MATLAB changes its vector length to maintain consistency with the new value of `nWeights`.

```
eqlvar = lineareq(10,varlms(0.01,0.01,0,1)) % Create equalizer object.
eqlvar.nWeights = 8 % Change the number of weights from 10 to 8.
% MATLAB automatically changes the sizes of eqlvar.Weights and
% eqlvar.WeightInputs.
```

The output below displays all the properties of the equalizer object before and after the change in the value of the `nWeights` property. In the second listing of properties, the `nWeights`, `Weights`, `WeightInputs`, and `StepSize` properties all have different values compared to the first listing of properties.

```
eqlvar =

                EqType: 'Linear Equalizer'
               AlgType: 'Variable Step Size LMS'
              nWeights: 10
            nSampPerSym: 1
                RefTap: 1
              SigConst: [-1 1]
              InitStep: 0.0100
               IncStep: 0.0100
               MinStep: 0
               MaxStep: 1
         LeakageFactor: 1
              StepSize: [1x10 double]
               Weights: [0 0 0 0 0 0 0 0 0 0]
          WeightInputs: [0 0 0 0 0 0 0 0 0 0]
 ResetBeforeFiltering: 1
  NumSamplesProcessed: 0


eqlvar =

                EqType: 'Linear Equalizer'
               AlgType: 'Variable Step Size LMS'
              nWeights: 8
            nSampPerSym: 1
                RefTap: 1
              SigConst: [-1 1]
              InitStep: 0.0100
               IncStep: 0.0100
               MinStep: 0
```

```
               MaxStep: 1
         LeakageFactor: 1
              StepSize: [1x8 double]
               Weights: [0 0 0 0 0 0 0 0]
          WeightInputs: [0 0 0 0 0 0 0 0]
    ResetBeforeFiltering: 1
     NumSamplesProcessed: 0
```

In typical applications, an equalizer begins by using a known sequence of transmitted symbols when adapting the equalizer weights. The known sequence, called a *training sequence*, enables the equalizer to gather information about the channel characteristics. After the equalizer finishes processing the training sequence, it adapts the equalizer weights in decision-directed mode using a detected version of the output signal. To use a training sequence when invoking the `equalize` function, include the symbols of the training sequence as an input vector.

The following code illustrates how to use `equalize` with a training sequence. The training sequence in this case is just the beginning of the transmitted message.

```
% Set up parameters and signals.
M = 4; % Alphabet size for modulation
msg = randint(1500,1,M); % Random message
modmsg = pskmod(msg,M); % Modulate using QPSK.
trainlen = 500; % Length of training sequence
chan = [.986; .845; .237; .123+.31i]; % Channel coefficients
filtmsg = filter(chan,1,modmsg); % Introduce channel distortion.

% Equalize the received signal.
eq1 = lineareq(8, lms(0.01)); % Create an equalizer object.
eq1.SigConst = pskmod([0:M-1],M); % Set signal constellation.
[symbolest,yd] = equalize(eq1,filtmsg,modmsg(1:trainlen)); % Equalize.

% Plot signals.
h = scatterplot(filtmsg,1,trainlen,'bx'); hold on;
scatterplot(symbolest,1,trainlen,'g.',h);
scatterplot(eq1.SigConst,1,0,'k*',h);
legend('Filtered signal','Equalized signal',...
   'Ideal signal constellation');
hold off;

% Compute error rates with and without equalization.
demodmsg_noeq = pskdemod(filtmsg,M); % Demodulate unequalized signal.
demodmsg = pskdemod(yd,M); % Demodulate detected signal from equalizer.
[nnoeq,rnoeq] = symerr(demodmsg_noeq(trainlen+1:end),...
   msg(trainlen+1:end));
[neq,req] = symerr(demodmsg(trainlen+1:end),...
   msg(trainlen+1:end));
disp('Symbol error rates with and without equalizer:')
disp([req rnoeq])
```

The example goes on to determine how many errors occur in trying to recover the modulated message with and without the equalizer. The symbol error rates, below, show that the equalizer improves the performance significantly.

```
Symbol error rates with and without equalizer:
         0    0.3410
```

The example also creates a scatter plot that shows the signal before and after equalization, as well as the signal constellation for QPSK modulation. Notice on the plot that the points of the equalized signal are clustered more closely around the points of the signal constellation.

Equalizing in Decision-Directed Mode

Decision-directed mode means the equalizer uses a detected version of its output signal when adapting the weights. Adaptive equalizers typically start with a training sequence and switch to decision-directed mode after exhausting all symbols in the training sequence. CMA equalizers are an exception, using neither training mode nor decision-directed mode. For non-CMA equalizers, the `equalize` function operates in decision-directed mode when one of these conditions is true:

- The syntax does not include a training sequence.
- The equalizer has exhausted all symbols in the training sequence and still has more input symbols to process.

The example in Equalizing Using a Training Sequence uses training mode when processing the first `trainlen` symbols of the input signal, and decision-directed mode thereafter. The example below discusses another scenario.

Example: Equalizing Multiple Times, Varying the Mode

If you invoke `equalize` multiple times with the same equalizer object to equalize a series of signal vectors, you might use a training sequence the first time you call the function and omit the training sequence in subsequent calls. Each iteration of the `equalize` function after the first one operates completely in decision-directed mode. However, because the `ResetBeforeFiltering` property of the equalizer object is set to `0`, the `equalize` function uses the existing state information in the equalizer object when starting each iteration's equalization operation. As a result, the training affects all equalization operations, not just the first.

The code below illustrates this approach. Notice that the first call to `equalize` uses a training sequence as an input argument, and the second call to `equalize` omits a training sequence.

```
M = 4; % Alphabet size for modulation
msg = randint(1500,1,M); % Random message
modmsg = pskmod(msg,M); % Modulate using QPSK.
trainlen = 500; % Length of training sequence
chan = [.986; .845; .237; .123+.31i]; % Channel coefficients
filtmsg = filter(chan,1,modmsg); % Introduce channel distortion.

% Set up equalizer.
eqlms = lineareq(8, lms(0.01)); % Create an equalizer object.
eqlms.SigConst = pskmod([0:M-1],M); % Set signal constellation.
% Maintain continuity between calls to equalize.
```

```
eqlms.ResetBeforeFiltering = 0;

% Equalize the received signal, in pieces.
% 1. Process the training sequence.
s1 = equalize(eqlms,filtmsg(1:trainlen),modmsg(1:trainlen));
% 2. Process some of the data in decision-directed mode.
s2 = equalize(eqlms,filtmsg(trainlen+1:800));
% 3. Process the rest of the data in decision-directed mode.
s3 = equalize(eqlms,filtmsg(801:end));
s = [s1; s2; s3]; % Full output of equalizer
```

Back to Top

Delays from Equalization

For proper equalization using adaptive algorithms other than CMA, you should set the reference tap so that it exceeds the delay, in symbols, between the transmitter's modulator output and the equalizer input. When this condition is satisfied, the total delay between the modulator output and the equalizer output is equal to

```
(RefTap-1)/nSampPerSym
```

symbols. Because the channel delay is typically unknown, a common practice is to set the reference tap to the center tap in a linear equalizer, or the center tap of the forward filter in a decision-feedback equalizer.

For CMA equalizers, the expression above does not apply because a CMA equalizer has no reference tap. If you need to know the delay, you can find it empirically after the equalizer weights have converged. Use the `xcorr` function to examine cross-correlations of the modulator output and the equalizer output.

Techniques for Working with Delays

Here are some typical ways to take a delay of D into account by padding or truncating data:

- Pad your original data with D extra symbols at the end. Before comparing the original data with the received data, omit the first D symbols of the received data. In this approach, all the original data (not including the padding) is accounted for in the received data.
- Before comparing the original data with the received data, omit the last D symbols of the original data and the first D symbols of the received data. In this approach, some of the original symbols are not accounted for in the received data.

The example below illustrates the latter approach. For an example that illustrates both approaches in the context of interleavers, see Delays of Convolutional Interleavers.

```
M = 2; % Use BPSK modulation for this example.
```

```
msg = randint(1000,1,M); % Random data
modmsg = pskmod(msg,M); % Modulate.
trainlen = 100; % Length of training sequence
trainsig = modmsg(1:trainlen); % Training sequence

% Define an equalizer and equalize the received signal.
eqlin = lineareq(3,normlms(.0005,.0001),pskmod(0:M-1,M));
eqlin.RefTap = 2; % Set reference tap of equalizer.
[eqsig,detsym] = equalize(eqlin,modmsg,trainsig); % Equalize.

detmsg = pskdemod(detsym,M); % Demodulate the detected signal.

% Compensate for delay introduced by RefTap.
D = (eqlin.RefTap -1)/eqlin.nSampPerSym;
trunc_detmsg = detmsg(D+1:end); % Omit first D symbols of equalized data.
trunc_msg = msg(1:end-D); % Omit last D symbols.

% Compute bit error rate, ignoring training sequence.
[numerrs,ber] = biterr(trunc_msg(trainlen+1:end),...
   trunc_detmsg(trainlen+1:end))
```

The output is below.

```
numerrs =

     0


ber =

     0
```

Equalizing Using a Loop

If your data is partitioned into a series of vectors (that you process within a loop, for example), you can invoke the `equalize` function multiple times, saving the equalizer's internal state information for use in a subsequent invocation. In particular, the final values of the `WeightInputs` and `Weights` properties in one equalization operation should be the initial values in the next equalization operation. This section gives an example, followed by more general procedures for equalizing within a loop.

Example: Adaptive Equalization Within a Loop

The example below illustrates how to use `equalize` within a loop, varying the equalizer between iterations. Because the example is long, this discussion presents it in these steps:

- Initializing Variables
- Simulating the System Using a Loop

If you want to equalize iteratively while potentially changing equalizers between iterations, see Changing the Equalizer Between Iterations for help generalizing from this example to other cases.

Initializing Variables. The beginning of the example defines parameters and creates three equalizer objects:

- An RLS equalizer object.
- An LMS equalizer object.
- A variable, `eq_current`, that points to the equalizer object to use in the current iteration of the loop. Initially, this points to the RLS equalizer object. After the second iteration of the loop, `eq_current` is redefined to point to the LMS equalizer object.

```
% Set up parameters.
M = 16; % Alphabet size for modulation
sigconst = qammod(0:M-1,M); % Signal constellation for 16-QAM
chan = [1 0.45 0.3+0.2i]; % Channel coefficients

% Set up equalizers.
eqrls = lineareq(6, rls(0.99,0.1)); % Create an RLS equalizer object.
eqrls.SigConst = sigconst; % Set signal constellation.
eqrls.ResetBeforeFiltering = 0; % Maintain continuity between iterations.
eqlms = lineareq(6, lms(0.003)); % Create an LMS equalizer object.
eqlms.SigConst = sigconst; % Set signal constellation.
eqlms.ResetBeforeFiltering = 0; % Maintain continuity between iterations.
eq_current = eqrls; % Point to RLS for first iteration.
```

Simulating the System Using a Loop. The next portion of the example is a loop that

- Generates a signal to transmit and selects a portion to use as a training sequence in the first iteration of the loop
- Introduces channel distortion
- Equalizes the distorted signal using the chosen equalizer for this iteration, retaining the final state and weights for later use
- Plots the distorted and equalized signals, for comparison
- Switches to an LMS equalizer between the second and third iterations

```
% Main loop
for jj = 1:4
   msg = randi([0 M-1],500,1); % Random message
   modmsg = qammod(msg,M); % Modulate using 16-QAM.

   % Set up training sequence for first iteration.
   if jj == 1
      ltr = 200; trainsig = modmsg(1:ltr);
   else
      % Use decision-directed mode after first iteration.
      ltr = 0; trainsig = [];
   end

   % Introduce channel distortion.
```

```
   filtmsg = filter(chan,1,modmsg);

   % Equalize the received signal.
   s = equalize(eq_current,filtmsg,trainsig);

   % Plot signals.
   h = scatterplot(filtmsg(ltr+1:end),1,0,'bx'); hold on;
   scatterplot(s(ltr+1:end),1,0,'g.',h);
   scatterplot(sigconst,1,0,'k*',h);
   legend('Received signal','Equalized signal','Signal constellation');
   title(['Iteration #' num2str(jj) ' (' eq_current.AlgType ')']);
   hold off;

   % Switch from RLS to LMS after second iteration.
   if jj == 2
      eqlms.WeightInputs = eq_current.WeightInputs; % Copy final inputs.
      eqlms.Weights = eq_current.Weights; % Copy final weights.
      eq_current = eqlms; % Make eq_current point to eqlms.
   end
end
```

The example produces one scatter plot for each iteration, indicating the iteration number and the adaptive algorithm in the title. A sample plot is below. Your plot might look different because this example uses random numbers.

Using MLSE Equalizers

Section Overview

The `mlseeq` function uses the Viterbi algorithm to equalize a signal through a dispersive channel. The function receives a baseband linearly modulated input signal and outputs the maximum likelihood sequence estimate of the signal, using an estimate of the channel modeled as a finite input response (FIR) filter.

The function decodes the received signal using these steps:

1. Applies the FIR filter, corresponding to the channel estimate, to the symbols in the input signal.
2. Uses the Viterbi algorithm to compute the traceback paths and the state metric, which are the numbers assigned to the symbols at each step of the Viterbi algorithm. The metrics are based on Euclidean distance.
3. Outputs the maximum likelihood sequence estimate of the signal, as a sequence of complex numbers corresponding to the constellation points of the modulated signal.

An MLSE equalizer yields the best possible performance, in theory, but is computationally intensive.

For background material about MLSE equalizers, see the works listed in Selected Bibliography for Equalizers.

Equalizing a Vector Signal

In its simplest form, the `mlseeq` function equalizes a vector of modulated data when you specify the estimated coefficients of the channel (modeled as an FIR filter), the signal constellation for the modulation type, and the traceback depth that you want the Viterbi algorithm to use. Larger values for the traceback depth can improve the results from the equalizer but increase the computation time.

An example of the basic syntax for `mlseeq` is below.

```
M = 4; const = pskmod([0:M-1],M); % 4-PSK constellation
msg = pskmod([1 2 2 0 3 1 3 3 2 1 0 2 3 0 1]',M); % Modulated message
chcoeffs = [.986; .845; .237; .12345+.31i]; % Channel coefficients
filtmsg = filter(chcoeffs,1,msg); % Introduce channel distortion.
tblen =  10; % Traceback depth for equalizer
chanest = chcoeffs; % Assume the channel is known exactly.
msgEq = mlseeq(filtmsg,chanest,const,tblen,'rst'); % Equalize.
```

The `mlseeq` function has two operation modes:

- Continuous operation mode enables you to process a series of vectors using repeated calls to `mlseeq`, where the function saves its internal state information from one call to the next. To learn more, see Equalizing in Continuous Operation Mode.
- Reset operation mode enables you to specify a preamble and postamble that accompany your data. To learn more, see Using a Preamble or Postamble.

If you are not processing a series of vectors and do not need to specify a preamble or postamble, the operation modes are nearly identical. However, they differ in that continuous operation mode incurs a delay, while reset operation mode does not. The example above could have used either mode, except that substituting continuous operation mode would have produced a delay in the equalized output. To learn more about the delay in continuous operation mode, see Delays in Continuous Operation Mode.

Equalizing in Continuous Operation Mode

If your data is partitioned into a series of vectors (that you process within a loop, for example), continuous operation mode is an appropriate way to use the `mlseeq` function. In continuous operation mode, `mlseeq` can save its internal state information for use in a subsequent invocation and can initialize using previously stored state information. To choose continuous operation mode, use `'cont'` as an input argument when invoking `mlseeq`. Procedure for Continuous Operation Mode

The typical procedure for using continuous mode within a loop is as follows:

1. Before the loop starts, create three empty matrix variables (for example, `sm`, `ts`, `ti`) that eventually store the state metrics, traceback states, and traceback inputs for the equalizer.
2. Inside the loop, invoke `mlseeq` using a syntax like

3. `[y,sm,ts,ti] = mlseeq(x,chcoeffs,const,tblen,'cont',nsamp,sm,ts,ti);`

   Using `sm`, `ts`, and `ti` as input arguments causes `mlseeq` to continue from where it finished in the previous iteration. Using `sm`, `ts`, and `ti` as output arguments causes `mlseeq` to update the state information at the end of the current iteration. In the first iteration, `sm`, `ts`, and `ti` start as empty matrices, so the first invocation of the `mlseeq` function initializes the metrics of all states to 0.

Delays in Continuous Operation Mode

Continuous operation mode with a traceback depth of `tblen` incurs an output delay of `tblen` symbols. This means that the first `tblen` output symbols are unrelated to the input signal, while the last `tblen` input symbols are unrelated to the output signal. For example, the command below uses a traceback depth of `3`, and the first 3 output symbols are unrelated to the input signal of `ones(1,10)`.

```
y = mlseeq(ones(1,10),1,[-7:2:7],3,'cont')
y =

    -7    -7    -7     1     1     1     1     1     1     1
```

Keeping track of delays from different portions of a communication system is important, especially if you compare signals to compute error rates. The example in Example: Continuous Operation Mode illustrates how to take the delay into account when computing an error rate.

Example: Continuous Operation Mode

The example below illustrates the procedure for using continuous operation mode within a loop. Because the example is long, this discussion presents it in multiple steps:

- Initializing Variables
- Simulating the System Using a Loop
- Computing an Error Rate and Plotting Results

Initializing Variables. The beginning of the example defines parameters, initializes the state variables `sm`, `ts`, and `ti`, and initializes variables that accumulate results from each iteration of the loop.

```
n = 200; % Number of symbols in each iteration
numiter = 25; % Number of iterations
M = 4; % Use 4-PSK modulation.
const = pskmod(0:M-1,M); % PSK constellation
chcoeffs = [1 ; 0.25]; % Channel coefficients
chanest = chcoeffs; % Channel estimate
tblen = 10; % Traceback depth for equalizer
nsamp = 1; % Number of input samples per symbol
sm = []; ts = []; ti = []; % Initialize equalizer data.
% Initialize cumulative results.
```

```
fullmodmsg = []; fullfiltmsg = []; fullrx = [];
```

Simulating the System Using a Loop. The middle portion of the example is a loop that generates random data, modulates it using baseband PSK modulation, and filters it. Finally, `mlseeq` equalizes the filtered data. The loop also updates the variables that accumulate results from each iteration of the loop.

```
for jj = 1:numiter
   msg = randint(n,1,M); % Random signal vector
   modmsg = pskmod(msg,M); % PSK-modulated signal
   filtmsg = filter(chcoeffs,1,modmsg); % Filtered signal

   % Equalize, initializing from where the last iteration
   % finished, and remembering final data for the next iteration.
   [rx sm ts ti] = mlseeq(filtmsg,chanest,const,tblen,...
      'cont',nsamp,sm,ts,ti);

   % Update vectors with cumulative results.
   fullmodmsg = [fullmodmsg; modmsg];
   fullfiltmsg = [fullfiltmsg; filtmsg];
   fullrx = [fullrx; rx];
end
```

Computing an Error Rate and Plotting Results.

The last portion of the example computes the symbol error rate from all iterations of the loop. The `symerr` function compares selected portions of the received and transmitted signals, not the entire signals. Because continuous operation mode incurs a delay whose length in samples is the traceback depth (`tblen`) of the equalizer, it is necessary to exclude the first `tblen` samples from the received signal and the last `tblen` samples from the transmitted signal. Excluding samples that represent the delay of the equalizer ensures that the symbol error rate calculation compares samples from the received and transmitted signals that are meaningful and that truly correspond to each other.

The example also plots the signal before and after equalization in a scatter plot. The points in the equalized signal coincide with the points of the ideal signal constellation for 4-PSK.

```
% Compute total number of symbol errors. Take the delay into account.
numsymerrs = symerr(fullrx(tblen+1:end),fullmodmsg(1:end-tblen))

% Plot signal before and after equalization.
h = scatterplot(fullfiltmsg); hold on;
scatterplot(fullrx,1,0,'r*',h);
legend('Filtered signal before equalization','Equalized signal',...
   'Location','NorthOutside');
hold off;
```

The output and plot follow.

```
numsymerrs =
```

```
     0
```

Using a Preamble or Postamble

Some systems include a sequence of known symbols at the beginning or end of a set of data. The known sequence at the beginning or end is called a *preamble* or *postamble*, respectively. The `mlseeq` function can accommodate a preamble and postamble that are already incorporated into its input signal. When you invoke the function, you specify the preamble and postamble as integer vectors that represent the sequence of known symbols by indexing into the signal constellation vector. For example, a preamble vector of `[1 4 4]` and a 4-PSK signal constellation of `[1 j -1 -j]` indicate that the modulated signal begins with `[1 -j -j]`.

If your system uses a preamble without a postamble, use a postamble vector of `[]` when invoking `mlseeq`. Similarly, if your system uses a postamble without a preamble, use a preamble vector of `[]`.

Example: Using a Preamble

The example below illustrates how to accommodate a preamble when using `mlseeq`. The same preamble symbols appear at the beginning of the message vector and in the syntax for `mlseeq`. If you want to use a postamble, you can append it to the message vector and also include it as the last input argument for `mlseeq`. In this example, however, the postamble input in the `mlseeq` syntax is an empty vector because the system uses no postamble.

```
M = 4; % Use 4-PSK modulation.
const = pskmod(0:3,4); % PSK constellation
tblen = 16; % Traceback depth for equalizer

preamble = [3; 1]; % Expected preamble, as integers
msgIdx = randint(98,1,M); % Random symbols
msgIdx = [preamble; msgIdx]; % Include preamble at the beginning.
msg = pskmod(msgIdx,M); % Modulated message
chcoeffs = [.623; .489+.234i; .398i; .21]; % Channel coefficients
chanest = chcoeffs; % Channel estimate
filtmsg = filter(chcoeffs,1,msg); % Introduce channel distortion.
d = mlseeq(filtmsg,chanest,const,tblen,'rst',1,preamble,[]);

[nsymerrs ser] = symerr(msg,d) % Symbol error rate
```

The output is below.

```
nsymerrs =

     0


ser =

     0
```

Procedures for Equalizing Within a Loop

This section describes two procedures for equalizing within a loop. The first procedure uses the same equalizer in each iteration, and the second is useful if you want to change the equalizer between iterations.

Using the Same Equalizer in Each Iteration. The typical procedure for using `equalize` within a loop is as follows:

1. Before the loop starts, create the equalizer object that you want to use in the first iteration of the loop.
2. Set the equalizer object's `ResetBeforeFiltering` property to `0` to maintain continuity between successive invocations of `equalize`.
3. Inside the loop, invoke `equalize` using a syntax like one of these:

```
y = equalize(eqz,x,trainsig);
y = equalize(eqz,x);
```

   The `equalize` function updates the state and weights of the equalizer at the end of the current iteration. In the next iteration, the function continues from where it finished in the previous iteration because `ResetBeforeFiltering` is set to `0`.

This procedure is similar to the one used in Example: Equalizing Multiple Times, Varying the Mode. That example uses `equalize` multiple times but not within a loop.

Changing the Equalizer Between Iterations. In some applications, you might want to modify the adaptive algorithm between iterations. For example, you might use a CMA equalizer for the first iteration and an LMS or RLS equalizer in subsequent iterations. The procedure below gives one way to accomplish this, roughly following the example in Example: Adaptive Equalization Within a Loop:

1. Before the loop starts, create the different kinds of equalizer objects that you want to use during various iterations of the loop.

   For example, create one CMA equalizer object, `eqcma`, and one LMS equalizer object, `eqlms`.

2. For each equalizer object, set the `ResetBeforeFiltering` property to `0` to maintain continuity between successive invocations of `equalize`.
3. Create a variable `eq_current` that points to the equalizer object you want to use for the first iteration. Use `=` to establish the connection so that the two objects get updated together:

```
eq_current = eqcma; % Point to eqcma.
```

   The purpose of `eq_current` is to represent the equalizer used in each iteration, where you can switch equalizers from one iteration to the next by using a command like `eq_current = eqlms`. The example illustrates this approach near the end of its loop.

5. Inside the loop, perform these steps:
    a. Invoke `equalize` using a syntax like one of these:
    b. `y = equalize(eq_current,x,trainsig);`
    c. `y = equalize(eq_current,x);`
    d. Copy the values of the `WeightInputs` and `Weights` properties from `eq_current` to the equalizer object that you want to use for the next iteration. Use dot notation. For example,
    e. `eqlms.WeightInputs = eq_current.WeightInputs;`
    f. `eqlms.Weights = eq_current.Weights;`
    g. Redefine `eq_current` to point to the equalizer object that you want to use for the next iteration, using `=`. Now `eq_current` is set up for the next iteration, because it represents the new kind of equalizer but retains the old values for the state and weights.

The reason for creating multiple equalizer objects and then copying the state and weights, instead of simply changing the equalizer class or adaptive algorithm in a single equalizer object, is that the class and adaptive algorithm properties of an equalizer object are fixed.

Least Mean Square (LMS)

This computer exercise deals with the LMS algorithm, which is derived from the method of steepest descent. The recursive equations for the error and the filter coefficients of the least mean square algorithm are given by

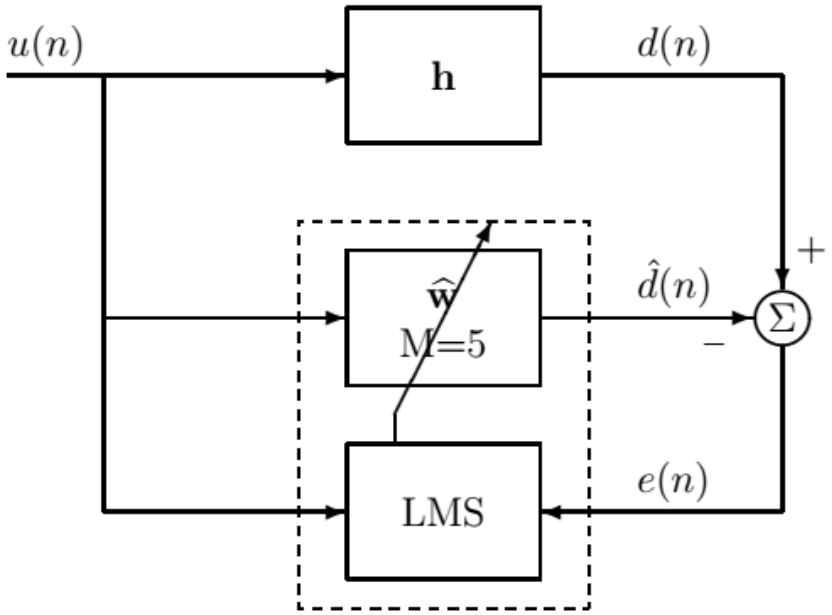
$$e(n) = d(n) - \widehat{\mathbf{w}}^H(n)\mathbf{u}(n)$$
$$\widehat{\mathbf{w}}(n+1) = \widehat{\mathbf{w}}(n) + \mu\mathbf{u}(n)e^*(n) \ .$$

Write a function in Matlab, which takes an input vector u and a reference signal d, both of length N , and calculates the error e for all time instants.

```
function [e,w]=lms(mu,M,u,d);
% Call:
% [e,w]=lms(mu,M,u,d);
%
% Input arguments:
% mu = step size, dim 1x1
% M = filter length, dim 1x1
% u = input signal, dim Nx1
% d = desired signal, dim Nx1
%
% Output arguments:
% e = estimation error, dim Nx1
% w = final filter coefficients, dim Mx1
```

Let the initial values of the filter coefficients be b w(0) = [0, 0, ..., 0]T. Mind the order of the elements in u(n) = [u(n), u(n − 1), ..., u(n − M + 1)]T! In Matlab you have to use uvec=u(n:-1:(n-M+1)), so that uvec corresponds to u(n). Furthermore, the input signal vector u is required to be a column vector.

Computer exercise 2.2 Now you shall verify that your LMS algorithm works properly. As a simple test, the adaptive filter should identify a short FIR-filter, shown in the figure below.



The filter that should be identified is h(n) = {1,1/2, 1/4, 1/8, 1/16}. Use white Gaussian noise as input signal that you filter with h(n) in order to obtain the desired signal d(n). Write in Matlab

```
% filter coefficients
h=0.5.^[0:4];
% input signal
u=randn(1000,1);
% filtered input signal == desired signal
d=conv(h,u);
% LMS
[e,w]=lms(0.1,5,u,d);
```

Compare the final filter coefficients (w) obtained by the LMS algorithm with the filter that it should identify (h). If the coefficients are equal, your LMS algorithm is correct. Note that in the current example there is no noise source influencing the driving noise u(n). Furthermore, the length of the adaptive filter M corresponds to the length of the FIR-filter to be identified. Therefore the error e(n) tends towards zero.

Computer exercise 2.3 Now you shall follow the example in Haykin, edition 4, chapter 5.7, pp. 285-291, (edition 3: chapter 9.7, pp. 412-421), Computer Experiment on Adaptive Equalization, and reproduce the result. Below follow some hints that will simplify the implementation.
A Bernoulli sequence is a random sequence of "+1" and "-1", where both occur with probability 1/2. In Matlab, such a sequence is generated by

```
% Bernoulli sequence of length N
x=2*round(rand(N,1))-1;
```

A learning curve is generated by taking the mean of the squared error e2(n) over several realizations of an ensemble, i.e.,

$$J(n) = \frac{1}{K}\sum_{k=1}^{K} e_k^2(n) \ , \quad n = 0, ..., N-1$$

where ek(n) is the estimation error at time instant n for the k-th realization, and K is the number of realizations to be considered. In order to plot J (n) with a logarithmic scale on the vertical axis, use the command semilogy.

Computer exercise 2.4 Calculation of the autocorrelation matrix R = E{u(n)uH(n)} and the cross-correlation vector p = E{u(n)d∗(n)} for the system in Haykin yields

$$\mathbf{R} = \begin{bmatrix} r(0) & r(1) & r(2) & \dots & r(10) \\ r(1) & r(0) & r(1) & \dots & r(9) \\ & \ddots & \ddots & \ddots & \\ r(10) & r(9) & r(8) & \dots & r(0) \end{bmatrix} \ ,$$

with

$$\begin{aligned} r(0) &= (h_1^2 + h_2^2 + h_3^3)\sigma_x^2 + \sigma_v^2 \\ r(1) &= (h_1 h_2 + h_2 h_3)\sigma_x^2 \\ r(2) &= h_1 h_3 \sigma_x^2 \\ r(k) &= 0, \quad k > 2 \quad , \end{aligned}$$

and

$$\mathbf{p} = \sigma_x^2 [0, \ 0, \ 0, \ 0, \ h_3, \ h_2, \ h_1, \ 0, \ 0, \ 0, \ 0 \ ]^T \ ,$$

respectively. The autocorrelation matrix can be generated in Matlab by

```
R=sigmax2*toeplitz([h1^2+h2^2+h3^2,h1*h2+h2*h3,h1*h3,zeros(1,8)])
+sigmav2*eye(11);
```

Calculate the Wiener filter for W = 3.1, and determine J_{min}. Give an estimate for Jex(∞) in Haykin, edition 4, figure 5.23, (edition 3: figure 9.23) for μ = 0.075 and μ = 0.025.

Which value of μ results in quicker convergence?
Which value of μ results in a smaller value for Jex(∞)?

If you have time, simulate the case μ = 0.0075 for N = 2500 samples, and give an estimate for Jex(∞)! Compare your results with the results obtained by applying the rules of thumb for M.

Program code

LMS

```
function [e,w]=lms(mu,M,u,d);
% Call:
% [e,w]=lms(mu,M,u,d);
%
% Input arguments:
% mu = step size, dim 1x1
% M = filter length, dim 1x1
% u = input signal, dim Nx1
% d = desired signal, dim Nx1
%
% Output arguments:
% e = estimation error, dim Nx1
% w = final filter coefficients, dim Mx1
%initial values: 0
w=zeros(M,1);
%number of samples of the input signal
N=length(u);
%Make sure that u and d are column vectors
u=u(:);
d=d(:);
%LMS
for n=M:N
uvec=u(n:-1:n-M+1);
e(n)=d(n)-w'*uvec;
w=w+mu*uvec*conj(e(n));
end
e=e(:);
```

Optimal filter

```
function [Jmin,R,p,wo]=wiener(W,sigmav2);
%WIENER Returns R and p, together with the Wiener filter
% solution and Jmin for computer exercise 2.4.
%
% Call:
% [Jmin,R,p,wo]=wiener(W,sigmav2);
%
% Input arguments:
% W =eigenvalue spread, dim 1x1
% sigmav2 =variance of the additive noise
% source, dim 1x1
%
% Output arguments:
% Jmin =minimum MSE obtained by the Wiener
% filter, dim 1x1
% R =autocorrelation matrix, dim 11x11
% p =cross-correlation vector, dim 11x1
% wo =optimal filter, dim 11x1
%
% Choose the remaining parameters according to
% Haykin chapter 9.7.
%filter coefficients h1,h2,h3
h1=1/2*(1+cos(2*pi/W*(1-2)));
h2=1/2*(1+cos(2*pi/W*(2-2)));
h3=1/2*(1+cos(2*pi/W*(3-2)));
%variance of driving noise
sigmax2=1;
%theoretical autocorrelation matrix R 11x11
R=sigmax2*toeplitz([h1^2+h2^2+h3^2,...
h1*h2+h2*h3,h1*h3,zeros(1,8)])+sigmav2*eye(11);
%theoretical cross-correlation vector p 11x1
p=sigmax2*[zeros(4,1);h3;h2;h1;zeros(4,1)];
%Wiener filter
wo=R\p;
%Jmin
Jmin=sigmax2-p'*wo;
```