



SİVAS UNIVERSITY OF SCIENCE AND TECHNOLOGY

FACULTY OF ENGINEERING AND NATURAL SCIENCES

**DIGITAL COMMUNICATIONS SYSTEMS
LABORATORY**

Experiments Manual Report

Supervisor: Assoc. Prof. Kholoud ELMABRUK

Prepared by:

Res. Asst. Kağan Murat PÜRLÜ

SİVAS

TABLE OF CONTENTS

Experiment 1: Sampling Theorem, Aliasing and Signal Reconstruction.....	3
Experiment 2: PCM Generation, Uniform Quantization and Binary Coding	12
Experiment 3: Non-Uniform Quantization, Companding and Differential PCM	24
Experiment 4: Line Coding and Time-Division Multiplexing.....	39
Experiment 5: Delta Modulation and Adaptive Delta Modulation	56
Experiment 6: ASK and FSK Modulation and Detection	77
Experiment 7: BPSK and Differential PSK	90
Experiment 8: QPSK Modulation, Constellation and Coherent Detection	103
Experiment 9: Optimum Baseband Receiver, Matched Filter, Correlator and BER	116
Experiment 10: Information Theory, Channel Capacity and Source Coding	131
Experiment 11: Linear Block, Hamming and Cyclic Codes	146
Experiment 12: Convolutional Encoding, Trellis Representation and Viterbi Decoding.....	159

Experiment 1: Sampling Theorem, Aliasing and Signal Reconstruction

1. Objective of the Experiment

- The objective of this experiment is to investigate the sampling process used in digital communication systems and to verify the sampling theorem through MATLAB. Students will generate a band-limited continuous-time reference signal, obtain discrete samples at different sampling frequencies, and examine the conditions required for accurate signal reconstruction.
- The experiment also demonstrates aliasing when the sampling frequency is lower than the Nyquist rate. Ideal, natural, and flat-top sampling methods are compared to show how practical pulse sampling differs from ideal impulse sampling and how the sample-and-hold operation may affect the sampled waveform.
- At the end of the experiment, students are expected to:

Identify the highest frequency component of a band-limited signal.

Calculate the Nyquist rate and select a suitable sampling frequency.

Generate and visualize sampled signals using MATLAB.

Explain spectral replication caused by periodic sampling.

Identify aliasing in both the time and frequency domains.

Reconstruct a signal from its samples using sinc interpolation.

Compare ideal, natural, and flat-top sampling methods.

Evaluate reconstruction quality using the mean squared error.

2. Theoretical Background

2.1. Sampling in Digital Communication Systems

- Sampling is the process of converting a continuous-time signal into a discrete-time sequence by measuring its amplitude at regular time intervals. If the sampling period is T_s seconds, the sampling frequency is

$$f_s = 1/T_s$$

- The sampled sequence is written as $x[n] = x(nT_s)$. Sampling is the first operation used in pulse-code modulation and other digital pulse-modulation systems. Before sampling, the input is normally passed through a low-pass filter to limit its bandwidth.

2.2. Sampling Theorem and Nyquist Rate

- According to the sampling theorem, a band-limited continuous-time signal can be represented completely by its samples and reconstructed without loss when the sampling frequency is at least twice the highest frequency component of the signal.

$$f_s \geq 2f_{\max}$$

- The minimum sampling frequency $2f_{\max}$ is called the Nyquist rate. The corresponding maximum sampling interval is

$$T_s \leq 1/(2f_{\max})$$

- In practice, a sampling frequency greater than the Nyquist rate is generally selected to provide a transition band for the reconstruction filter and to reduce sensitivity to nonideal conditions.

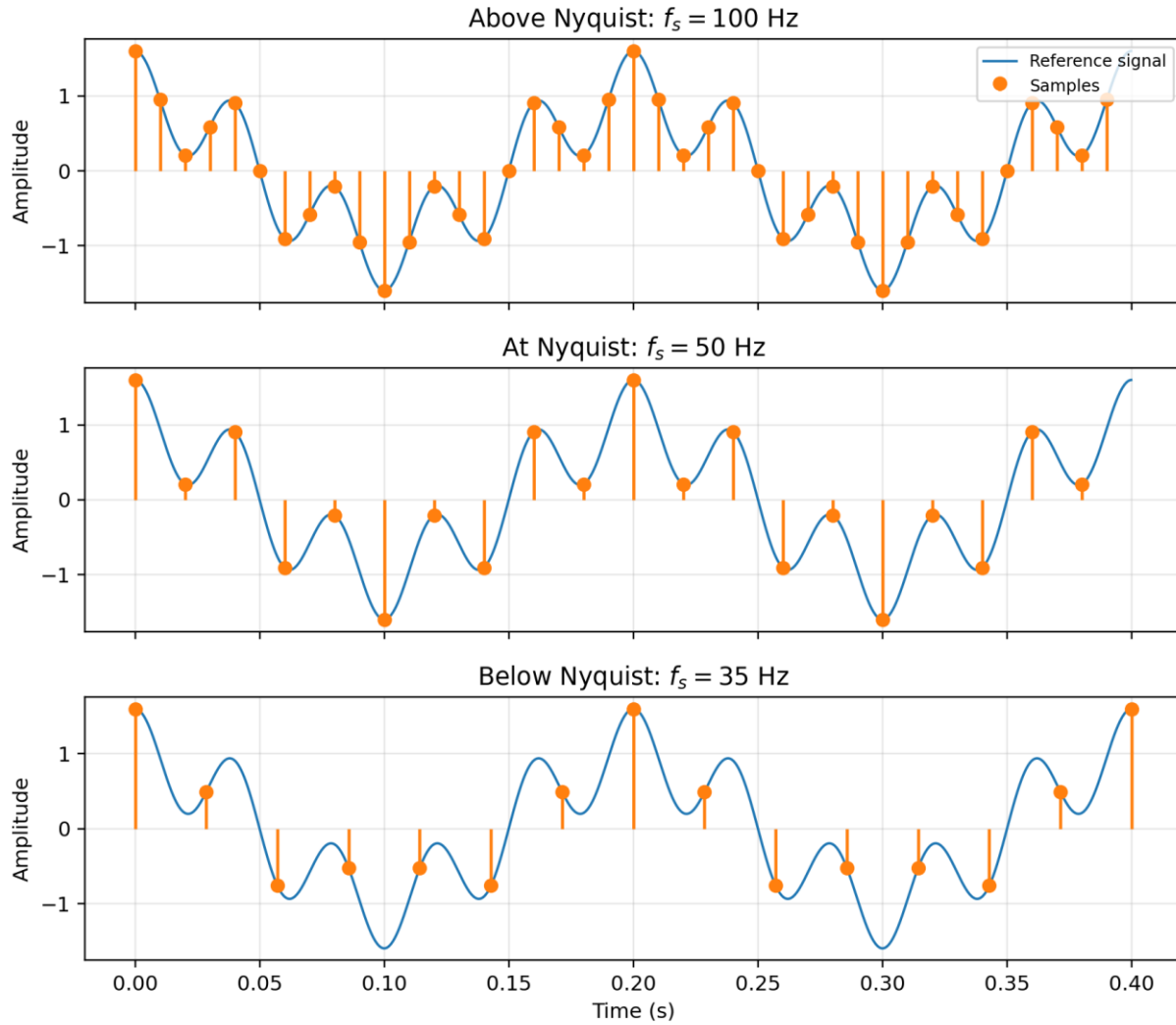


Fig. 1.1. Sampling of a two-tone signal above, at, and below the Nyquist rate.

2.3. Frequency-Domain Interpretation and Aliasing

- Ideal sampling may be represented as the multiplication of $x(t)$ by an impulse train. Multiplication in the time domain produces periodic replicas of the original spectrum in the frequency domain. The sampled spectrum can be expressed conceptually as

$$X_s(f) = f_s \sum X(f - kf_s), \quad k = \dots, -2, -1, 0, 1, 2, \dots$$

- When $f_s \geq 2f_{\max}$, adjacent spectral replicas do not overlap and the original spectrum can be selected by a low-pass reconstruction filter. When $f_s < 2f_{\max}$, the replicas overlap. This irreversible spectral overlap is called aliasing. High-frequency components then appear as lower-frequency components in the sampled data.

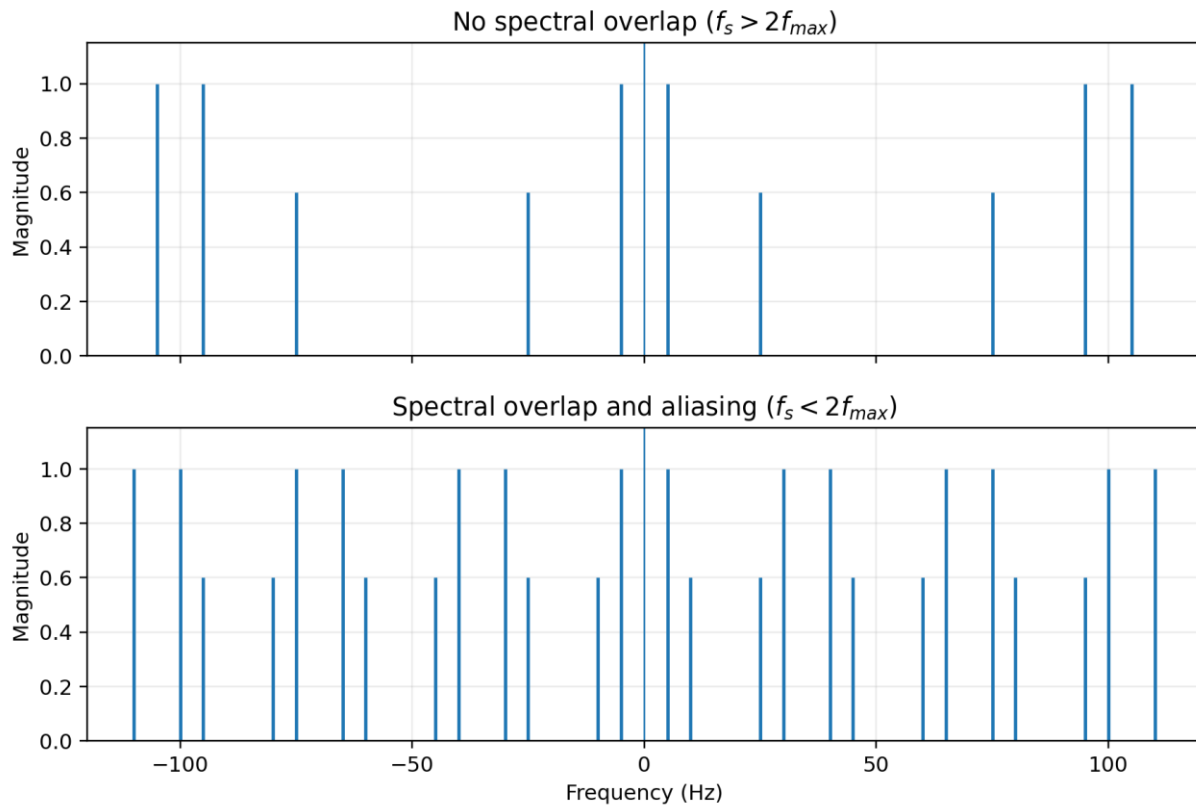


Fig. 1.2. Conceptual spectral replication without overlap and with aliasing.

2.4. Ideal, Natural, and Flat-Top Sampling

- The lecture notes distinguish three sampling methods:

Ideal sampling: the signal is multiplied by an impulse train. This representation is mathematically convenient but cannot be implemented physically.

Natural sampling: a train of short pulses gates the input signal. The top of each sample follows the variation of the analog signal during the pulse width.

Flat-top sampling: a sample-and-hold circuit keeps each sampled amplitude constant during the pulse duration.

- Natural and flat-top sampling weight the spectrum by the Fourier transform of the sampling pulse. In flat-top sampling, high-frequency components are attenuated because of the sinc-shaped pulse response. This attenuation is referred to as the aperture effect, and it becomes more significant as the pulse width increases.

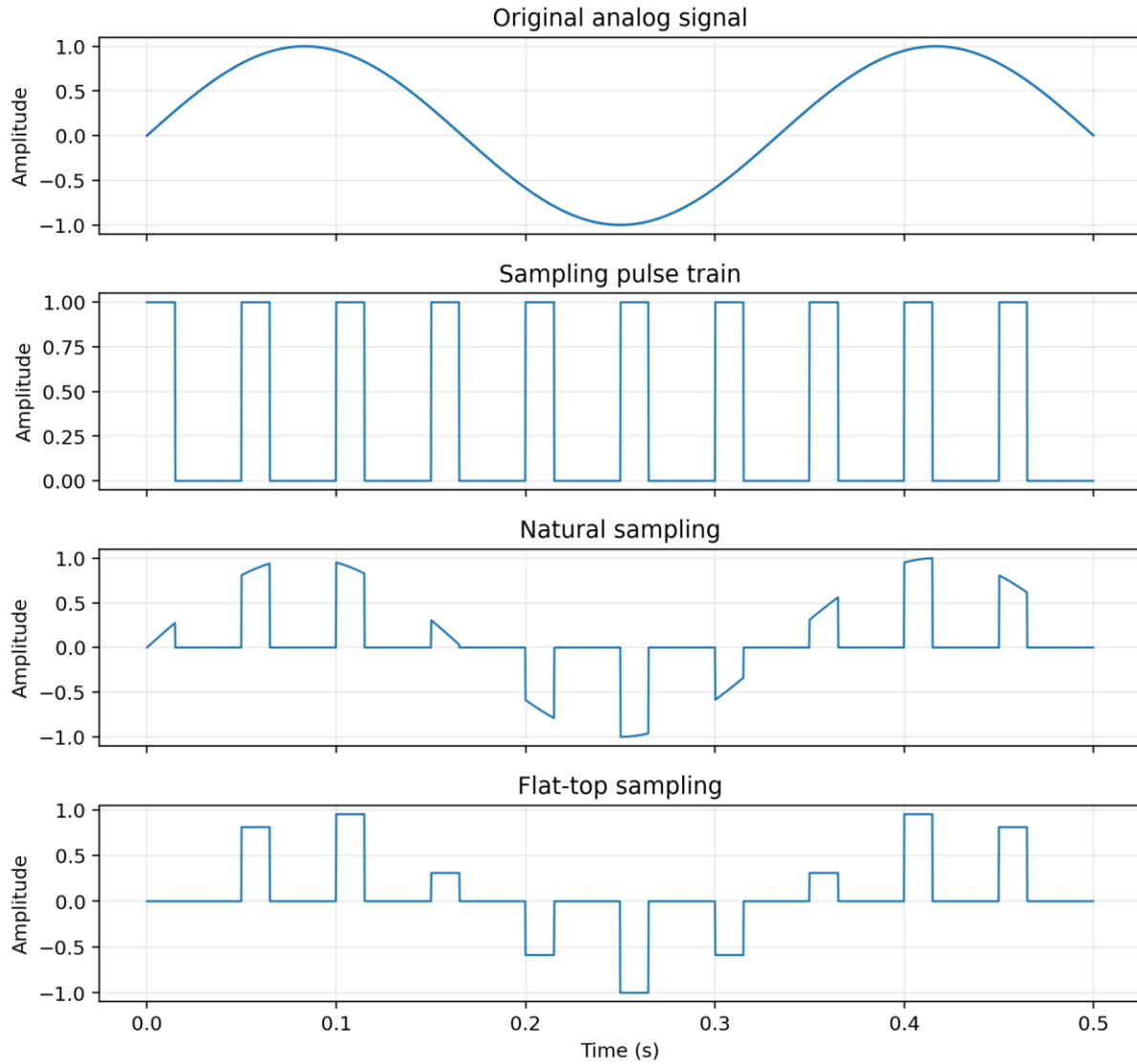


Fig. 1.3. Conceptual comparison of the original signal, sampling pulse train, natural sampling, and flat-top sampling.

2.5. Signal Reconstruction

- When the sampling theorem is satisfied, the original signal can be reconstructed by passing the sampled signal through an ideal low-pass filter whose cutoff frequency includes the original signal bandwidth. The equivalent time-domain expression is sinc interpolation:

$$\hat{x}(t) = \sum x(nT_s) \text{sinc}[(t - nT_s)/T_s]$$

- where $\text{sinc}(u) = \sin(\pi u)/(\pi u)$. Each sample scales a shifted sinc function, and the sum of all shifted sinc functions reconstructs the continuous-time waveform. If aliasing has already occurred, ideal interpolation cannot recover the original signal because different frequency components have become indistinguishable.

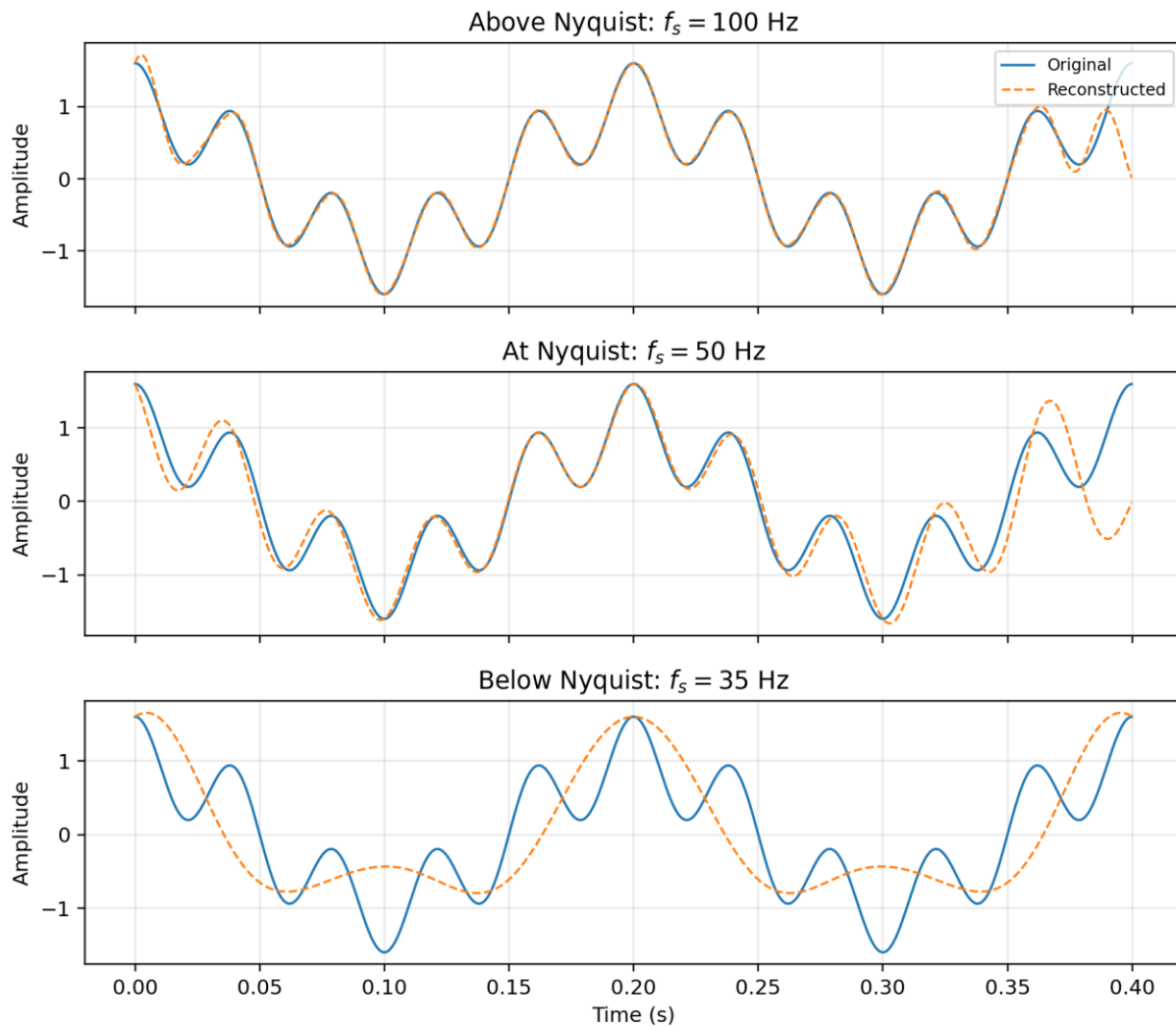


Fig. 1.4. Sinc reconstruction for sampling frequencies above, at, and below the Nyquist rate.

2.6. Reconstruction Error

- The difference between the reference signal $x(t)$ and the reconstructed signal $\hat{x}(t)$ may be evaluated using the mean squared error (MSE):

$$\text{MSE} = (1/N) \sum [x[k] - \hat{x}[k]]^2$$

- A small MSE indicates that the reconstructed waveform closely follows the reference waveform. The measured MSE may still be nonzero because the observation interval is finite and only a finite number of sinc terms are included.

3. Materials Used

Computer

MATLAB software

MATLAB Script Editor

Digital Communications lecture notes

Laboratory worksheet

Calculator

- The main MATLAB commands used in this experiment are:

cos, sin, mod, abs and double

plot, stem, subplot, grid, xlabel, ylabel and title

fft, fftshift and abs

sinc, mean and length

4. Preliminary (PreLab)

- Complete the following tasks before attending the laboratory. Use the reference signal

$$x(t) = \cos(2\pi \cdot 5t) + 0.6\cos(2\pi \cdot 25t)$$

Identify all frequency components and determine $f_{\max}$.

Calculate the Nyquist rate and the maximum allowable sampling period.

For $f_s = 100$ Hz, 50 Hz, and 35 Hz, calculate T_s and classify each case as above, at, or below the Nyquist rate.

For $f_s = 35$ Hz, predict the apparent frequency of the 25 Hz component after sampling.

Sketch the expected sampled waveform for each sampling frequency.

Explain why reconstruction is expected to fail in the below-Nyquist case.

State the main difference between natural and flat-top sampling.

Table 1.1. Preliminary calculations

Parameter	Expression or Given Value	Calculated Result
Signal frequency components	5 Hz and 25 Hz	
Highest signal frequency, $f_{\max}$		
Nyquist rate, $2f_{\max}$		
Maximum sampling interval		
Sampling period for $f_s = 100$ Hz		
Sampling period for $f_s = 50$ Hz		
Sampling period for $f_s = 35$ Hz		
Predicted alias of 25 Hz at $f_s = 35$ Hz		

5. Procedure

Part 1: Generation of the Continuous-Time Reference Signal

Open MATLAB and create a new script file.

Define a high reference sampling frequency of 5000 Hz to represent the continuous-time waveform numerically.

Define an observation interval of 0.4 s.

Generate the two-tone reference signal $x(t) = \cos(2\pi \cdot 5t) + 0.6\cos(2\pi \cdot 25t)$.

Plot the signal and identify the slow and fast oscillatory components.

```
f_ref = 5000;
Tobs = 0.4;
t_ref = 0:1/f_ref:Tobs-1/f_ref;
x_ref = cos(2*pi*5*t_ref) + 0.6*cos(2*pi*25*t_ref);
```

Part 2: Sampling Above, At, and Below the Nyquist Rate

Create the sampling-frequency vector $f_s = [100 \ 50 \ 35]$ Hz.

For each sampling frequency, generate the sampling-time vector $t_s = 0:T_s:T_{\text{obs}}-T_s$.

Evaluate the reference signal at the sampling instants.

Use stem to display the discrete samples and plot to display the continuous-time reference signal.

Place the three cases in the same figure using subplot(3,1,...).

Compare the sample density and waveform representation for the three cases.

```
fs_values = [100 50 35];
for k = 1:length(fs_values)
    fs = fs_values(k);
    ts = 0:1/fs:Tobs-1/fs;
    xs = cos(2*pi*5*ts) + 0.6*cos(2*pi*25*ts);
    % Plot x_ref and the samples using plot and stem
end
```

Part 3: Frequency-Domain Analysis and Aliasing

Calculate the FFT of the sampled sequence for each sampling frequency.

Normalize the FFT magnitude by the number of samples.

Use fftshift to center the spectrum around zero frequency.

Construct the frequency axis using $f = (-N/2:N/2-1)(f_s/N)$ for even N .

Identify the peaks corresponding to the 5 Hz and 25 Hz components.

For $f_s = 35$ Hz, determine the frequency at which the aliased 25 Hz component appears.

Record the measured peak frequencies in Table 1.2.

```
N = length(xs);
X = fftshift(fft(xs))/N;
f = (-N/2:N/2-1)*(fs/N);
plot(f,abs(X));
grid on;
```

Part 4: Sinc Reconstruction

Reconstruct the signal for each sampling frequency using shifted sinc functions.

Plot the original and reconstructed signals on the same axes.

Calculate the MSE between the reference and reconstructed signals.

Comment on the reconstruction quality above, at, and below the Nyquist rate.

Explain why the below-Nyquist reconstruction contains an incorrect low-frequency component.

```
x_rec = zeros(size(t_ref));
for n = 1:length(ts)
    x_rec = x_rec + xs(n)*sinc((t_ref-ts(n))*fs);
end
MSE = mean((x_ref-x_rec).^2);
```

Part 5: Natural and Flat-Top Sampling

Generate the test signal $m(t) = \sin(2\pi \cdot 3t)$ for $0 \leq t < 0.5$ s.

Use $f_s = 20$ Hz and a sampling-pulse width of 15 ms.

Generate a rectangular sampling-pulse train.

Obtain the naturally sampled waveform by multiplying the analog signal by the pulse train.

Generate the flat-top waveform by holding each sample value constant during the pulse width.

Display the original signal, pulse train, naturally sampled signal, and flat-top sampled signal in one figure.

Increase the pulse width and observe the change in the flat-top waveform. Relate the observation to the aperture effect described in the lecture notes.

6. Evaluation of Results

- Tabulate the data obtained during the experiment and attach all MATLAB figures to the laboratory report.

Table 1.2. Sampling and spectral observations

Sampling Frequency	Sampling Period	Classification	Expected Components	Measured Components
100 Hz		Above Nyquist	± 5 Hz, ± 25 Hz	
50 Hz		At Nyquist	± 5 Hz, ± 25 Hz	
35 Hz		Below Nyquist	± 5 Hz and aliased component	

Table 1.3. Signal-reconstruction results

Sampling Frequency	Nyquist Condition	MSE	Visual Reconstruction Quality	Aliasing Observed?
100 Hz				
50 Hz				

Sampling Frequency	Nyquist Condition	MSE	Visual Reconstruction Quality	Aliasing Observed?
35 Hz				

Table 1.4. Comparison of sampling methods

Sampling Method	Pulse/Sample Characteristic	Main Observation	Expected Spectral Effect
Ideal sampling			
Natural sampling			
Flat-top sampling			

- Percentage error for a measured frequency may be calculated using

$$\% \text{ Error} = |(\text{Measured Value} - \text{Theoretical Value}) / \text{Theoretical Value}| \times 100$$

- Answer the following questions:

What is the highest frequency component of the reference signal, and what is its Nyquist rate?

Why is $f_s = 100$ Hz expected to provide accurate reconstruction?

What practical limitation may still affect reconstruction when f_s is exactly equal to the Nyquist rate?

At which frequency does the 25 Hz component appear when $f_s = 35$ Hz? Explain the result.

Why can an ideal low-pass reconstruction filter not remove aliasing after it has occurred?

How does increasing the sampling frequency affect the separation of spectral replicas?

What is the difference between ideal, natural, and flat-top sampling?

What is the aperture effect, and how is it influenced by sampling-pulse width?

Why is an anti-aliasing low-pass filter used before a practical sampler?

Compare the MSE values obtained for the three sampling frequencies.

Which time-domain and frequency-domain observations provide evidence of aliasing?

Explain whether a signal sampled below the Nyquist rate can ever be reconstructed uniquely from the samples alone.

- Submit the following:

MATLAB script file with the .m extension

All time-domain and frequency-domain plots

Completed Tables 1.1–1.4

Calculated MSE values

A brief interpretation of the results and answers to the evaluation questions

Experiment 2: PCM Generation, Uniform Quantization and Binary Coding

1. Objective of the Experiment

- The objective of this experiment is to implement the main stages of a pulse-code modulation (PCM) system using MATLAB. Students will sample a band-limited analog signal, quantize the sample amplitudes with uniform mid-rise and mid-tread quantizers, encode the quantizer indices into binary codewords, and generate a serial PCM bit stream.
- The experiment also investigates quantization error, quantization-noise power, signal-to-quantization-noise ratio (SQNR), PCM bit rate, and the transmission bandwidth required for different numbers of bits per sample. The decoded and reconstructed waveform is compared with the original signal to show the tradeoff between quantizer resolution, data rate, bandwidth, and reconstruction quality.
- At the end of the experiment, students are expected to:

Describe the sampling, quantization, encoding, and reconstruction stages of a PCM system.

Implement uniform mid-rise and mid-tread quantizers in MATLAB.

Calculate the number of levels, step size, and binary codeword length.

Generate a serialized PCM bit stream from quantized sample indices.

Measure quantization error and quantization-noise power.

Evaluate SQNR as the number of bits per sample changes.

Calculate PCM signaling rate and minimum transmission bandwidth.

Decode and reconstruct a PCM waveform and assess reconstruction quality.

2. Theoretical Background

2.1. Pulse-Code Modulation

- Pulse-code modulation converts an analog signal into a digital sequence through three principal operations: sampling, quantization, and binary encoding. In the transmitter described in the lecture notes, the input signal is first bandwidth-limited by a low-pass filter, sampled by a sample-and-hold circuit, quantized to one of a finite number of amplitude levels, encoded into a binary word, and converted from parallel to serial form for transmission.

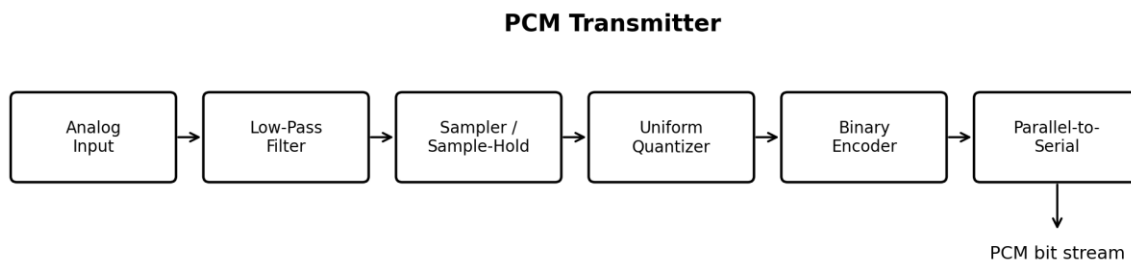


Fig. 2.1. Main processing stages of a PCM transmitter.

- If the highest input frequency is W hertz, the sampling theorem requires

$$f_s \geq 2W$$

- The sampled signal is discrete in time but remains continuous in amplitude. Quantization makes the sample amplitudes discrete, and the encoder represents each quantized level by a binary word.

2.2. Uniform Quantization

- Quantization represents each sampled amplitude by the nearest value from a finite set of representation or reconstruction levels. For a quantizer with n bits per sample, the number of levels is

$$L = 2^n$$

- For an input range from $x_{\min}$ to $x_{\max}$, the uniform quantization step size is

$$\Delta = (x_{\max} - x_{\min})/L$$

- The difference between the original sample $x[k]$ and the quantized sample $x_q[k]$ is the quantization error:

$$e[k] = x[k] - x_q[k]$$

- For a properly operating uniform quantizer without overload, the error magnitude is bounded approximately by

$$-\Delta/2 \leq e[k] \leq \Delta/2$$

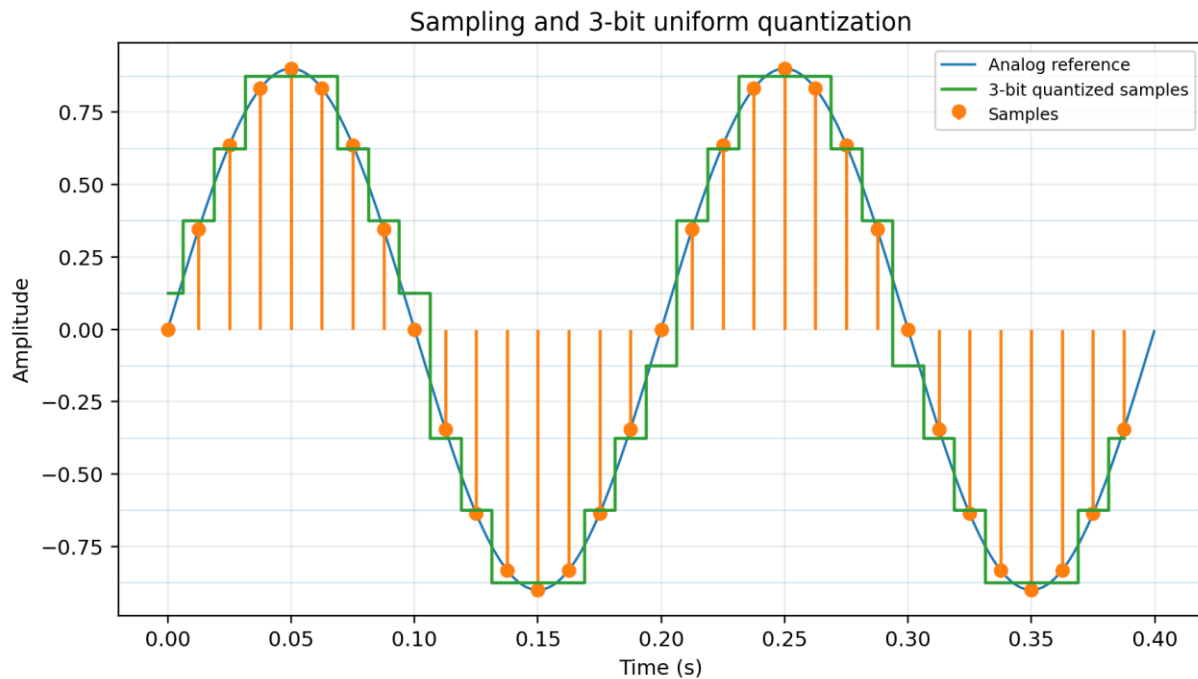


Fig. 2.2. Sampling and 3-bit uniform quantization of a sinusoidal signal.

2.3. Mid-Rise and Mid-Tread Quantizers

- The lecture notes identify two common uniform quantizers. In a mid-rise quantizer, the origin lies in the middle of a rising part of the staircase characteristic and the number of levels is even. In a mid-tread quantizer, the origin lies in the middle of a horizontal tread, so zero is one of the representation levels and the number of levels is odd.

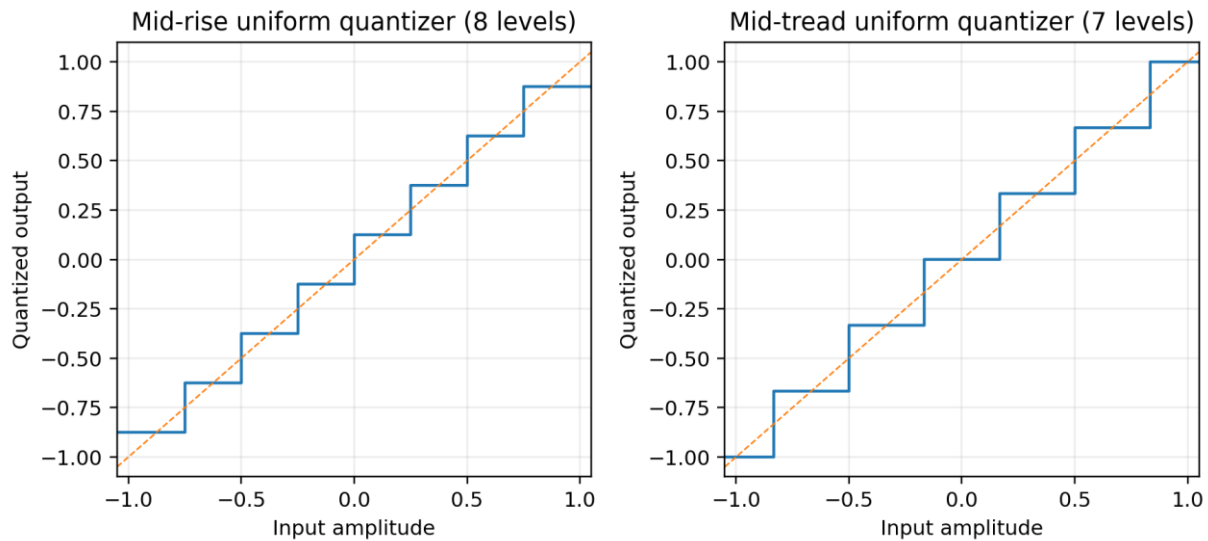


Fig. 2.3. Uniform mid-rise and mid-tread quantizer characteristics.

- The quantizer must also be checked for overload. If a sample exceeds the selected input range, it is clipped to the highest or lowest available level, causing overload distortion that is not described by the usual $\pm\Delta/2$ error bound.

2.4. Binary Encoding and PCM Bit Stream

- Each of the L quantization levels is assigned a unique n -bit binary codeword. The quantizer index may be converted directly to binary, where the lowest level is assigned index 0 and the highest level is assigned index $L-1$. The parallel codewords are then serialized to form the PCM baseband signal.

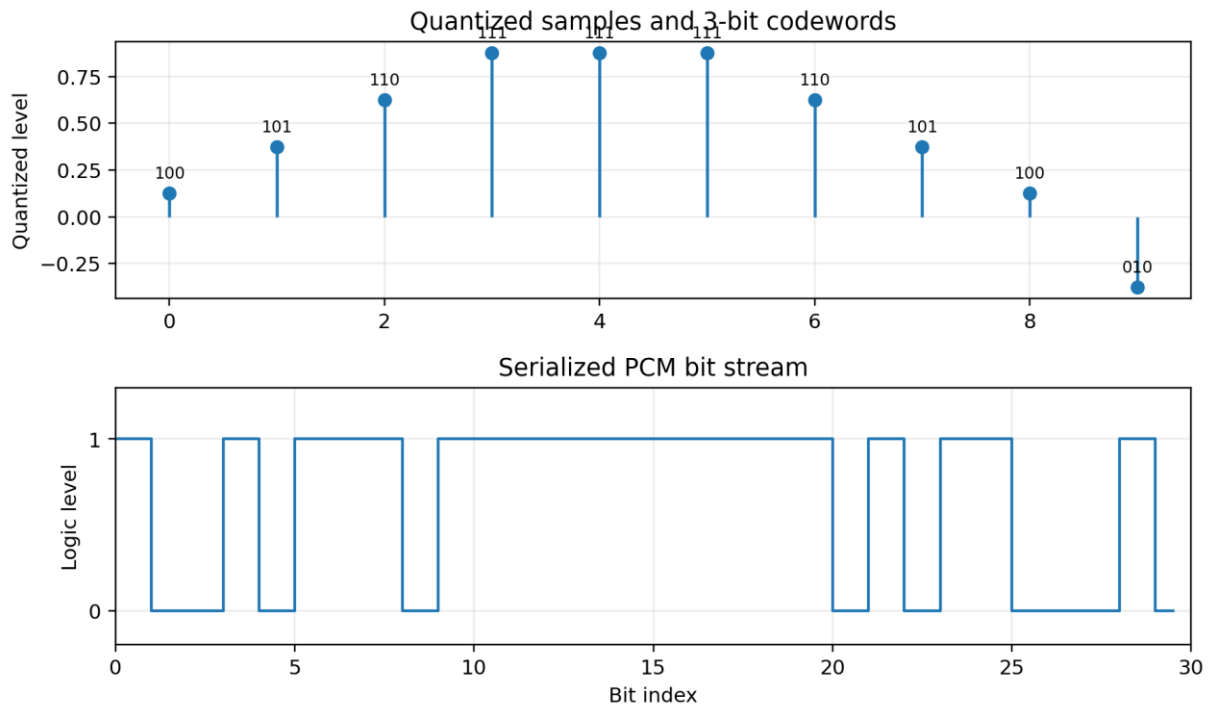


Fig. 2.4. Example quantized samples, 3-bit codewords, and serialized PCM bit stream.

2.5. Quantization Noise and SQNR

- For a uniform quantizer with sufficiently small step size, the quantization error is commonly modeled in the lecture notes as a uniformly distributed random variable over the interval from $-\Delta/2$ to $+\Delta/2$. Its mean-square value, and therefore the normalized quantization-noise power for a 1-ohm load, is

$$P_q = \Delta^2/12$$

- The signal-to-quantization-noise ratio is

$$\text{SQNR} = P_s/P_q$$

- For the normalized maximum-signal condition developed in the lecture notes, the maximum ratio increases with the number of bits approximately as

$$\text{SQNR}_{\text{max,dB}} \approx 4.77 + 6.02n$$

- Thus, increasing the quantizer resolution substantially reduces quantization noise, but it also increases the number of transmitted bits per sample.

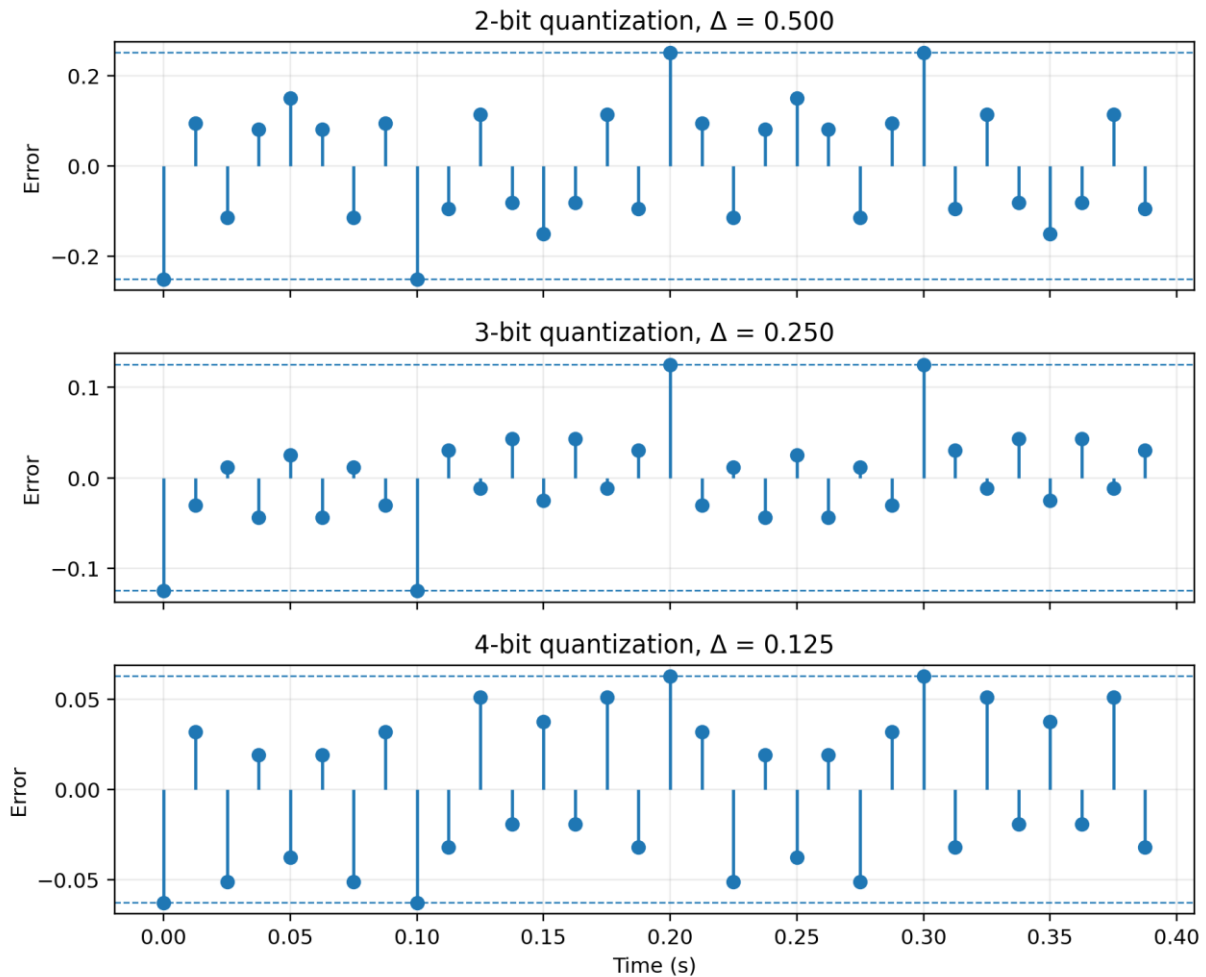


Fig. 2.5. Quantization error for different numbers of bits per sample.

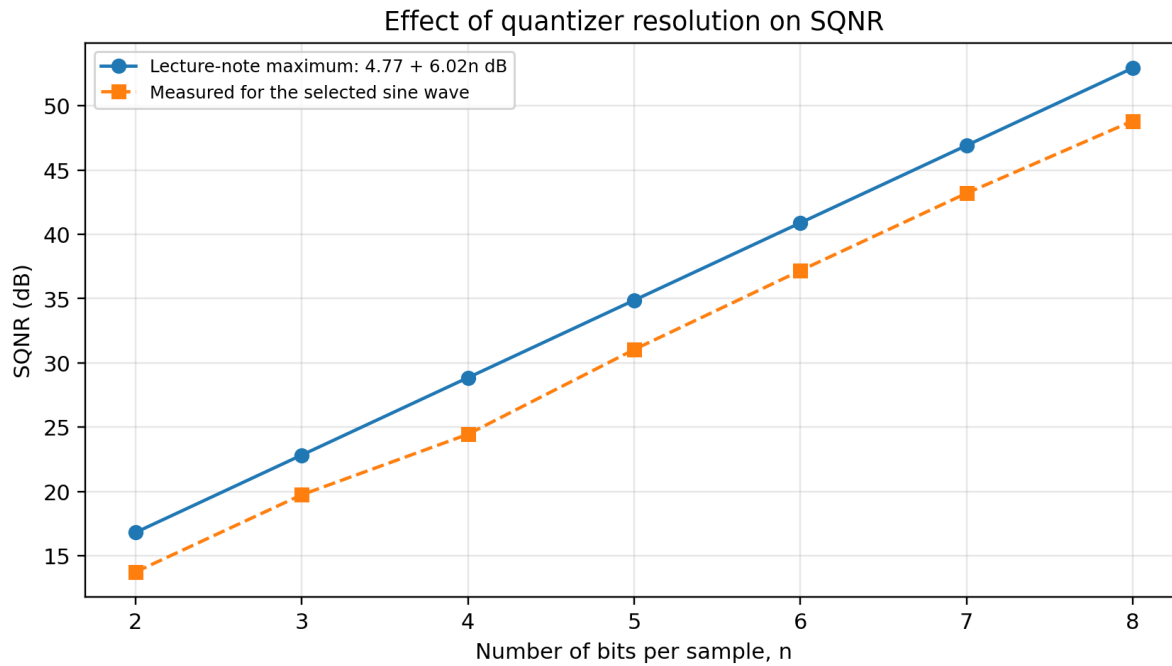


Fig. 2.6. Theoretical and measured SQNR as quantizer resolution increases.

2.6. PCM Signaling Rate and Transmission Bandwidth

- If each sample is represented by n bits and the sampling frequency is f_s samples per second, the PCM signaling or bit rate is

$$R_\beta = nf_s \text{ bit/s}$$

- The lecture notes state that the minimum baseband transmission bandwidth is approximately one-half of the signaling rate:

$$B_t \geq R_\beta/2 = nf_s/2$$

- When the sampling frequency is selected at the Nyquist rate $f_s = 2W$, this becomes

$$B_t \geq nW$$

- Increasing the number of bits therefore improves amplitude resolution and SQNR but requires a proportionally higher bit rate and transmission bandwidth.

2.7. PCM Receiver and Reconstruction

- At the receiver, the serial PCM stream is converted to parallel codewords, decoded into quantized amplitudes, converted to an analog sample-and-hold waveform, and passed through a reconstruction low-pass filter. Even with error-free transmission, the recovered signal generally contains permanent quantization error because the decoder can reproduce only the selected reconstruction levels.

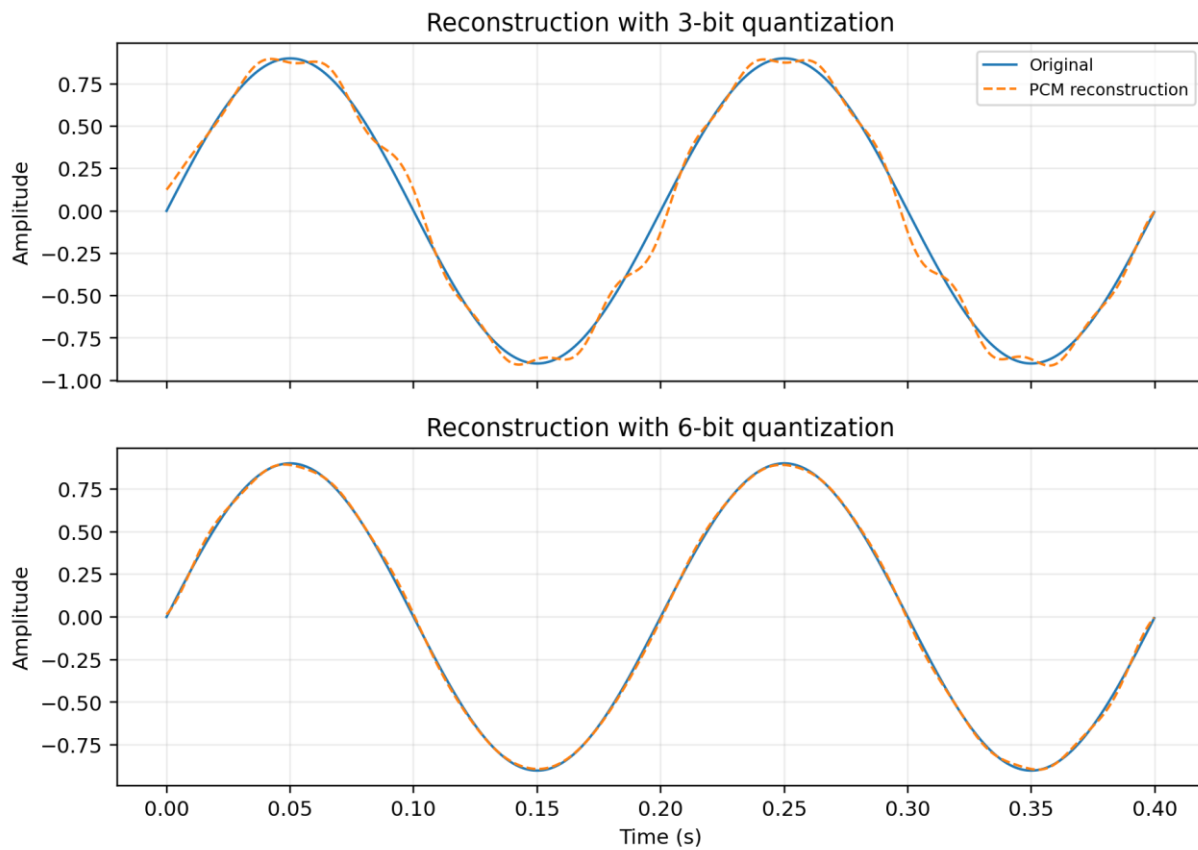


Fig. 2.7. PCM reconstruction with 3-bit and 6-bit quantization.

3. Materials Used

Computer

MATLAB software

MATLAB Script Editor

Digital Communications lecture notes

Laboratory worksheet

Calculator

- The main MATLAB commands used in this experiment are:

sin, floor, round, clip operations, min and max

dec2bin or format-based binary conversion

reshape, transpose and logical indexing

plot, stem, stairs, subplot, grid and legend

mean, sum, log10 and sinc

4. Preliminary (PreLab)

- Use the test signal

$$x(t) = 0.9\sin(2\pi \cdot 5t)$$

- with the quantizer input range $-1 \leq x \leq 1$ and sampling frequency $f_s = 80$ Hz.
1. Determine the highest signal frequency and verify that f_s satisfies the sampling theorem.
 2. For $n = 2, 3, 4, 6$, and 8 bits, calculate the number of quantization levels L and the step size Δ .
 3. For each resolution, calculate the theoretical quantization-noise power $\Delta^2/12$.
 4. Calculate the lecture-note maximum SQNR estimate $4.77 + 6.02n$ dB.
 5. For each resolution, calculate the PCM bit rate $R_\beta = nf_s$.
 6. Calculate the minimum PCM transmission bandwidth $B_t = R_\beta/2$.
 7. For a 3-bit mid-rise quantizer, determine the reconstruction level and binary index for the sample values $-0.82, -0.18, 0.13, 0.52$, and 0.94 .
 8. Explain the expected difference between mid-rise and mid-tread quantization around zero.

Table 2.1. Preliminary PCM calculations

Bits n	Levels L	Step Size Δ	Noise Power $\Delta^2/12$	SQNR Estimate (dB)	Bit Rate (bit/s)	Minimum BW (Hz)
2						
3						
4						
6						
8						

Table 2.2. Manual 3-bit quantization and encoding

Input Sample	Quantizer Index	Quantized Level	Quantization Error	3-bit Codeword
-0.82				
-0.18				
0.13				
0.52				
0.94				

5. Procedure

Part 1: Signal Generation and Sampling

1. Open MATLAB and create a new script file.
2. Define the reference display frequency as 4000 Hz, the PCM sampling frequency as 80 Hz, and the observation interval as 0.4 s.
3. Generate the reference signal $x(t) = 0.9\sin(2\pi \cdot 5t)$.

4. Generate the sample sequence $x[k] = x(kT_s)$.
5. Plot the reference signal and the discrete samples in the same figure.
6. Confirm that the selected sampling frequency satisfies the sampling theorem.

```
f_ref = 4000;
fs = 80;
Tobs = 0.4;
t = 0:1/f_ref:Tobs-1/f_ref;
ts = 0:1/fs:Tobs-1/fs;
x = 0.9*sin(2*pi*5*t);
xs = 0.9*sin(2*pi*5*ts);
```

Part 2: Uniform Mid-Rise Quantization

1. Set the quantizer input range to $x_{\min} = -1$ and $x_{\max} = 1$.
2. Implement a 3-bit mid-rise quantizer using $L = 2^3$ levels.
3. Calculate the quantizer index of each sample.
4. Calculate the corresponding reconstruction level.
5. Plot the original samples and quantized samples.
6. Verify that the quantization error remains within approximately $\pm\Delta/2$ when overload does not occur.
7. Repeat the quantization for $n = 2, 4, 6$, and 8 bits.

```
n = 3;
L = 2^n;
xmin = -1; xmax = 1;
Delta = (xmax-xmin)/L;
xs_clip = min(max(xs,xmin),xmax-eps);
index = floor((xs_clip-xmin)/Delta);
xq = xmin + (index+0.5)*Delta;
error = xs-xq;
```

Part 3: Mid-Tread Quantization

1. Implement a symmetric mid-tread quantizer with a zero reconstruction level.
2. Quantize the same sample sequence.
3. Plot the mid-rise and mid-tread outputs for samples close to zero.
4. Compare the number and placement of levels.
5. Comment on the treatment of small-amplitude samples around the origin.

```
Delta_t = 2/6; % Seven levels from -1 to +1
xq_t = round(xs/Delta_t)*Delta_t;
xq_t = min(max(xq_t,-1),1);
```

Part 4: Binary Encoding and Serialization

1. Convert each 3-bit mid-rise quantizer index into a three-character binary codeword.

2. Display the first ten sample values, indices, quantized amplitudes, and codewords.
3. Concatenate the codewords to generate one serial PCM bit stream.
4. Plot the serialized bit stream using a staircase waveform.
5. Verify that the bit-stream length equals the number of samples multiplied by the number of bits per sample.

```
codewords = dec2bin(index,n);  
serial_bits = reshape(codewords,'-0',1,[]);  
stairs(0:length(serial_bits)-1,serial_bits,'LineWidth',1.2);  
ylim([-0.2 1.2]);
```

Part 5: Quantization Noise and SQNR

1. For $n = 2, 3, 4, 6$, and 8 bits, calculate the measured quantization-noise power $\text{mean}(\text{error}^2)$.
2. Calculate the measured signal power $\text{mean}(x_s^2)$.
3. Calculate $\text{SQNR} = 10\log_{10}(P_s/P_q)$.
4. Calculate the theoretical noise power $\Delta^2/12$.
5. Calculate the lecture-note maximum SQNR estimate $4.77 + 6.02n$ dB.
6. Plot the measured and theoretical SQNR values against n .
7. Explain any difference between the measured curve and the maximum normalized estimate.

```
Psignal = mean(xs.^2);  
Pnoise = mean(error.^2);  
SQNR_dB = 10*log10(Psignal/Pnoise);  
Pnoise_theory = Delta^2/12;  
SQNR_note_dB = 4.77 + 6.02*n;
```

Part 6: PCM Bit Rate and Transmission Bandwidth

1. For every quantizer resolution, calculate $R_\beta = n f_s$.
2. Calculate the corresponding minimum transmission bandwidth $B_t = R_\beta/2$.
3. Plot bit rate and minimum bandwidth against n .
4. Explain the tradeoff between quantization quality and required transmission resources.

Part 7: PCM Decoding and Reconstruction

1. Reshape the serial bit stream into n -bit codewords.
2. Convert each codeword back to its decimal quantizer index.
3. Convert the indices back to quantized reconstruction levels.
4. Reconstruct a continuous-time waveform using sinc interpolation of the decoded quantized samples.
5. Perform the reconstruction for 3-bit and 6-bit PCM.

6. Plot the original and reconstructed signals on the same axes.
7. Calculate the reconstruction MSE for each case and record the results.

```
xrec = zeros(size(t));
for k = 1:length(ts)
    xrec = xrec + xq(k)*sinc((t-ts(k))*fs);
end
MSE = mean((x-xrec).^2);
```

Part 8: Overload Test

1. Change the input amplitude from 0.9 to 1.2 while keeping the quantizer range at -1 to 1 .
 2. Quantize the new samples with the 3-bit quantizer.
 3. Identify the clipped samples and plot the overload distortion.
 4. Compare this error with ordinary granular quantization error.
 5. Explain why increasing the number of bits does not remove clipping unless the input range is also changed.
6. Evaluation of Results
- Complete the tables using MATLAB results and attach the requested figures to the laboratory report.

Table 2.3. Quantization performance

Bits	Levels	Step Size	Measured Noise Power	Theoretical $\Delta^2/12$	Measured SQNR (dB)	Lecture-Note Estimate (dB)
2						
3						
4						
6						
8						

Table 2.4. PCM data rate and bandwidth

Bits per Sample	Sampling Frequency	PCM Bit Rate	Minimum Transmission Bandwidth	Bits in 0.4 s
2	80 Hz			
3	80 Hz			
4	80 Hz			
6	80 Hz			
8	80 Hz			

Table 2.5. Reconstruction results

Resolution	Reconstruction MSE	Maximum Absolute Error	Visual Quality	Overload Present?
3 bits				
6 bits				
3 bits, input amplitude 1.2				

Table 2.6. Mid-rise and mid-tread comparison

Property	Mid-Rise Quantizer	Mid-Tread Quantizer
Number of levels used		
Zero reconstruction level		
Behavior near zero		
Typical staircase origin		

- Answer the following questions:
 1. What are the three principal operations required to convert an analog signal to a PCM bit stream?
 2. How are the number of quantization levels and the number of bits per sample related?
 3. Why does quantization cause a permanent error even when the communication channel is noiseless?
 4. How does the quantization step size change when one additional bit is used?
 5. Do all measured errors remain between $-\Delta/2$ and $+\Delta/2$? Explain any exceptions.
 6. Compare mid-rise and mid-tread quantizers with particular attention to the origin.
 7. Why does SQNR improve as the number of bits per sample increases?
 8. Why may the measured SQNR differ from the lecture-note maximum normalized estimate?
 9. How does increasing n affect the PCM signaling rate and minimum transmission bandwidth?
 10. What information is contained in the binary codeword of each PCM sample?
 11. Why must the transmitter and receiver agree on the quantizer range, resolution, and codeword mapping?
 12. Compare the reconstructed waveforms and MSE values for 3-bit and 6-bit PCM.
 13. What is overload distortion, and why is it different from ordinary quantization noise?
 14. How would a channel bit error affect a decoded PCM sample, particularly if the most significant bit were corrupted?

- Submit the following:

MATLAB script file with the .m extension

Sampling, quantization, error, SQNR, PCM bit-stream, and reconstruction figures

Completed Tables 2.1–2.6

A list of the first ten samples and their PCM codewords

Answers to all evaluation questions and a brief discussion of the resolution-bandwidth tradeoff

Experiment 3: Non-Uniform Quantization, Companding and Differential PCM**1. Objective of the Experiment**

- The objective of this experiment is to investigate two methods used to reduce the limitations and redundancy of conventional pulse-code modulation. The first part focuses on non-uniform quantization implemented through companding. Students will apply μ -law and A-law compression before uniform quantization, perform the corresponding expansion at the receiver, and compare the results with ordinary uniform PCM.
- The second part focuses on differential pulse-code modulation (DPCM). Students will exploit the correlation between adjacent samples by predicting the current sample from previously reconstructed samples and quantizing only the prediction error. MATLAB simulations will be used to compare dynamic range, quantization step size, error power, SQNR, and reconstruction quality.
- At the end of the experiment, students are expected to:

Describe why a non-uniform quantizer uses small steps for low amplitudes and larger steps for high amplitudes.

Implement μ -law and A-law compressor and expander characteristics in MATLAB.

Construct a companded PCM system using a uniform quantizer between the compressor and expander.

Compare low-level and high-level quantization performance for uniform, μ -law, and A-law PCM.

Explain the sample-to-sample redundancy used by a DPCM system.

Implement a one-tap DPCM predictor, prediction-error quantizer, and local reconstruction loop.

Compare the prediction-error range with the original-signal range.

Evaluate the effect of predictor coefficient and quantizer range on DPCM reconstruction quality.

2. Theoretical Background**2.1. Non-Uniform Quantization**

- In uniform quantization, the distance between adjacent reconstruction levels is constant. This provides the same absolute resolution over the entire input range. However, the lecture notes emphasize that low-amplitude signals require finer resolution because a fixed quantization error represents a larger fraction of a small signal than of a large signal.
- A non-uniform quantizer therefore uses smaller step sizes near the origin and larger step sizes at higher amplitudes. Low-level samples are represented more accurately, while large-amplitude samples are represented with coarser absolute spacing. This behavior improves the signal-to-quantization-noise ratio over a wide dynamic range, especially for speech-like signals containing many low-level samples.

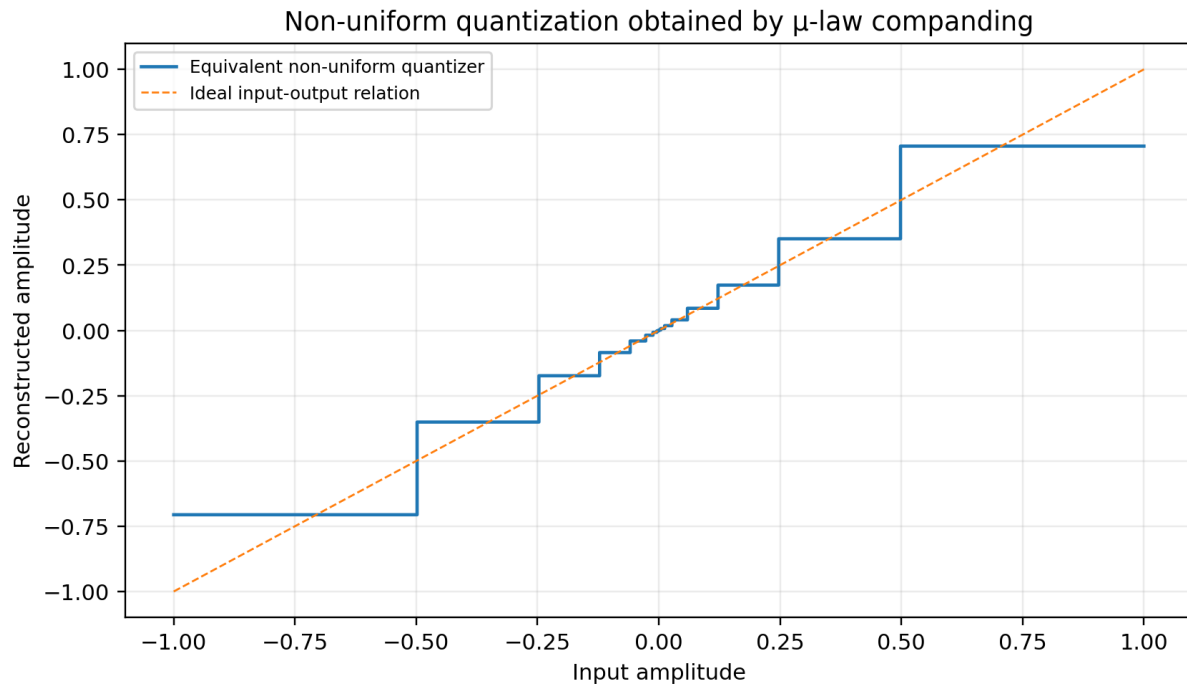


Fig. 3.1. Equivalent non-uniform quantizer characteristic obtained by μ -law companding.

2.2. Companding PCM System

- Direct implementation of a non-uniform quantizer may be difficult and expensive. The lecture notes present companding as an alternative. The signal is compressed by a nonlinear function before entering a uniform quantizer. At the receiver, an inverse nonlinear function expands the decoded signal. The combined compressor, uniform quantizer, and expander produce an overall characteristic similar to a non-uniform quantizer.

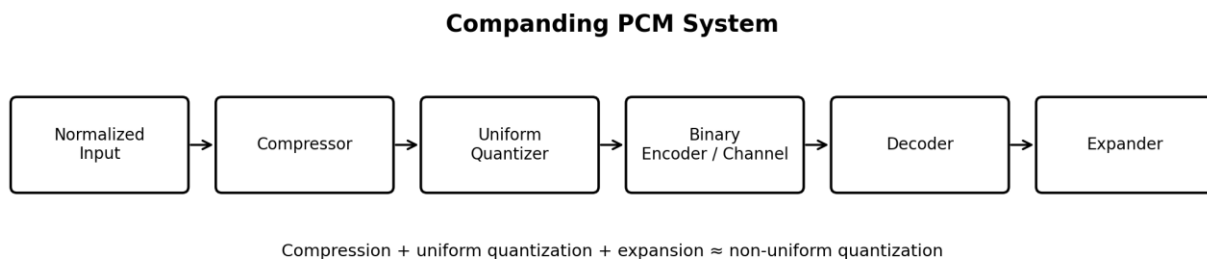


Fig. 3.2. Basic processing stages of a companding PCM system.

- For all companding equations used in this experiment, the input is normalized so that

$$|x| \leq 1$$

2.3. μ -Law Companding

- The μ -law compressor is commonly used for speech signals. The normalized compressor characteristic given in the lecture notes is

$$C_{\mu}(x) = \text{sgn}(x) \cdot \ln(1 + \mu|x|) / \ln(1 + \mu), \quad |x| \leq 1$$

- The practical value used in this experiment is $\mu = 255$. The corresponding expander is the inverse function

$$C\mu^{-1}(y) = \text{sgn}(y) \cdot [(1 + \mu)^{|y|} - 1] / \mu$$

- The logarithmic compression increases the relative spacing of small input amplitudes in the compressed domain. After uniform quantization and expansion, the effective quantization steps are small near zero and increase with input magnitude.

2.4. A-Law Companding

- The lecture notes also describe A-law companding, which uses a linear segment for very small amplitudes and a logarithmic segment for larger amplitudes. The normalized compressor is

$$CA(x) = \text{sgn}(x) \cdot A|x|/(1 + \ln A), \quad 0 \leq |x| < 1/A$$

$$CA(x) = \text{sgn}(x) \cdot [1 + \ln(A|x|)]/(1 + \ln A), \quad 1/A \leq |x| \leq 1$$

- The practical value is approximately $A = 87.6$. The inverse characteristic used by the receiver is

$$CA^{-1}(y) = \text{sgn}(y) \cdot |y|(1 + \ln A)/A, \quad |y| < 1/(1 + \ln A)$$

$$CA^{-1}(y) = \text{sgn}(y) \cdot \exp[|y|(1 + \ln A) - 1]/A, \quad \text{otherwise}$$

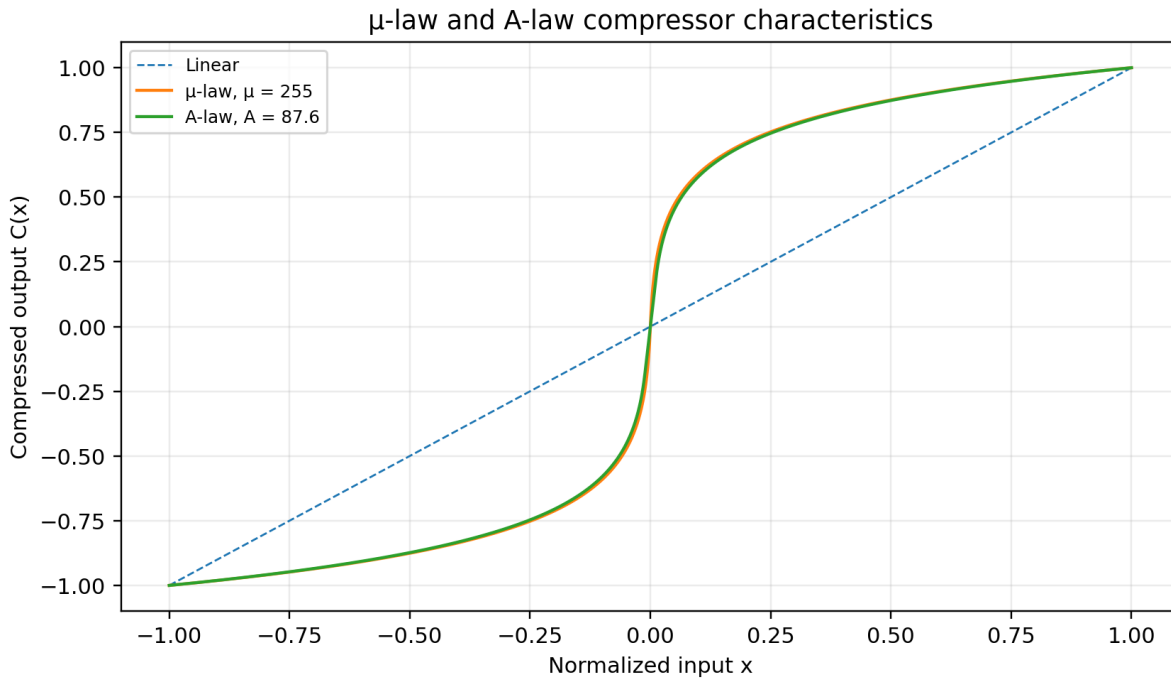


Fig. 3.3. Normalized linear, μ -law, and A-law compressor characteristics.

2.5. Effect of Companding on Quantization Error

- The compressed signal is quantized uniformly. Because compression expands low-amplitude values before quantization, more code intervals are effectively assigned to the low-amplitude region. After expansion, the quantization error is therefore reduced for small signals. The tradeoff is that high-amplitude regions may have larger effective step sizes than an ordinary uniform quantizer with the same number of bits.
- Companding should consequently be evaluated separately over low- and high-amplitude intervals rather than only through one overall mean-square error value.

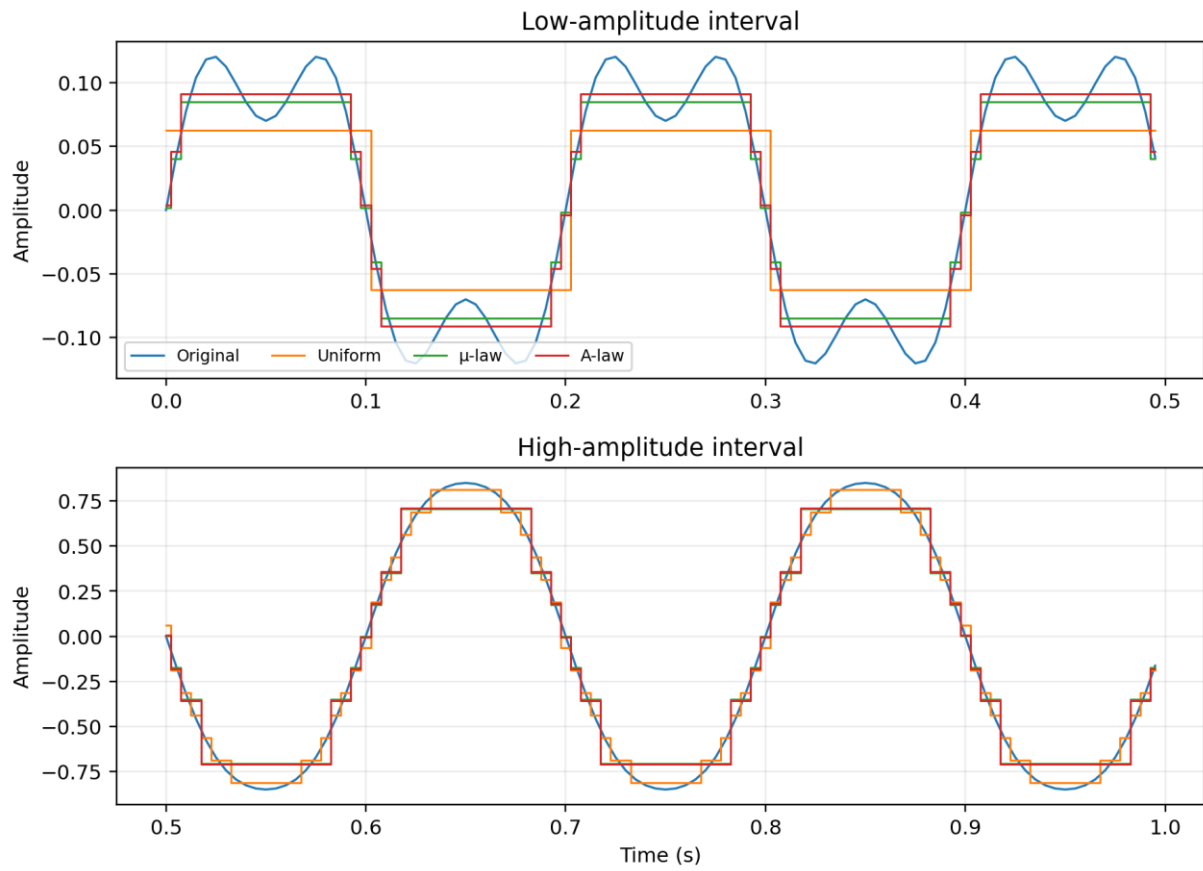


Fig. 3.4. Four-bit uniform and companded quantization for low- and high-amplitude intervals.

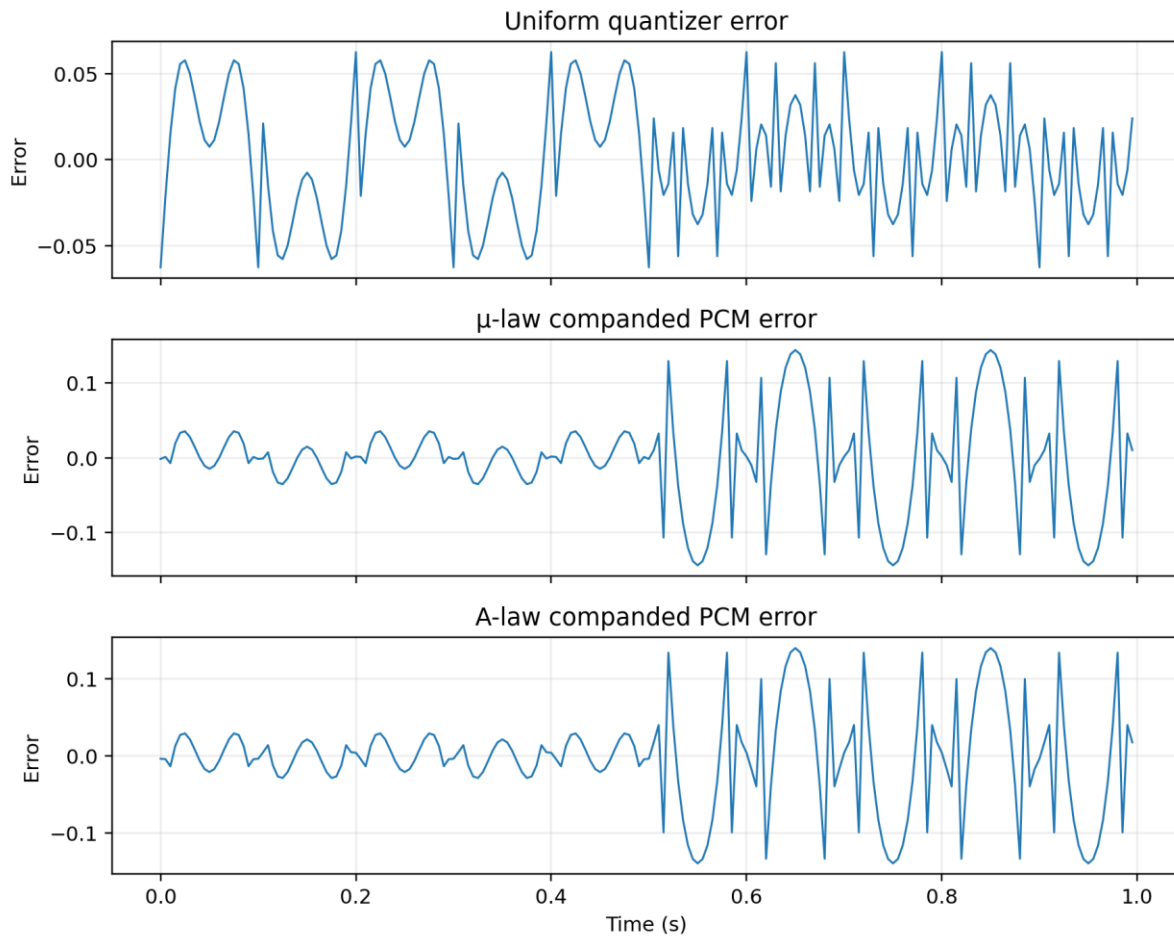


Fig. 3.5. Quantization-error comparison for uniform, μ -law, and A-law PCM.

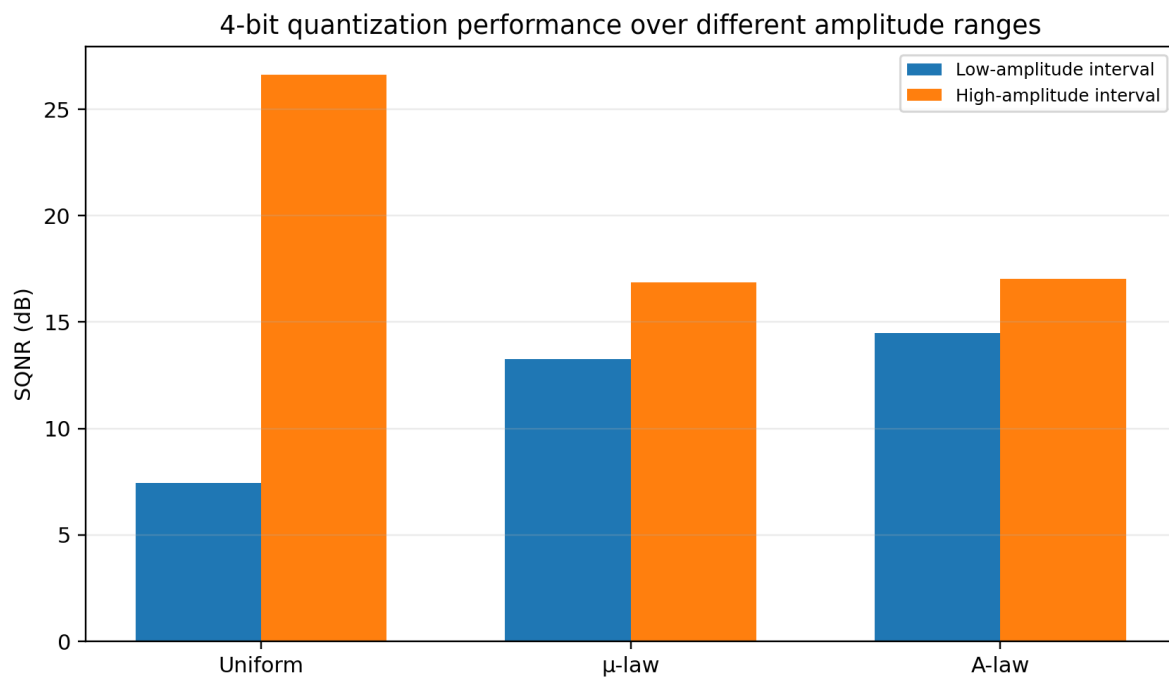


Fig. 3.6. Low- and high-amplitude SQNR comparison for 4-bit quantization.

2.6. Redundancy in Conventional PCM

- The lecture notes state that neighboring samples of a sufficiently oversampled or slowly varying signal are highly correlated. In conventional PCM, each sample is encoded independently even though several successive samples may have similar codewords. This produces redundant information.
- DPCM reduces this redundancy by predicting the current sample and encoding the difference between the actual and predicted values. If the predictor is effective, the prediction error has a smaller dynamic range than the original signal and can be quantized with a smaller step size for the same number of bits.

2.7. DPCM Transmitter and Receiver

- Let $x[k]$ be the current input sample and $\hat{x}[k]$ be its predicted value. The prediction error is

$$e[k] = x[k] - \hat{x}[k]$$

- The error is quantized and encoded. The transmitter also performs local reconstruction so that its predictor uses the same reconstructed samples as the receiver:

$$x_q[k] = \hat{x}[k] + eq[k]$$

- A first-order predictor used in this laboratory is

$$\hat{x}[k] = a \cdot x_q[k - 1]$$

- where a is the predictor coefficient. At the receiver, the decoded quantized error is added to the receiver prediction. Because both transmitter and receiver predictors are driven by previously reconstructed samples, they remain synchronized in an error-free channel.

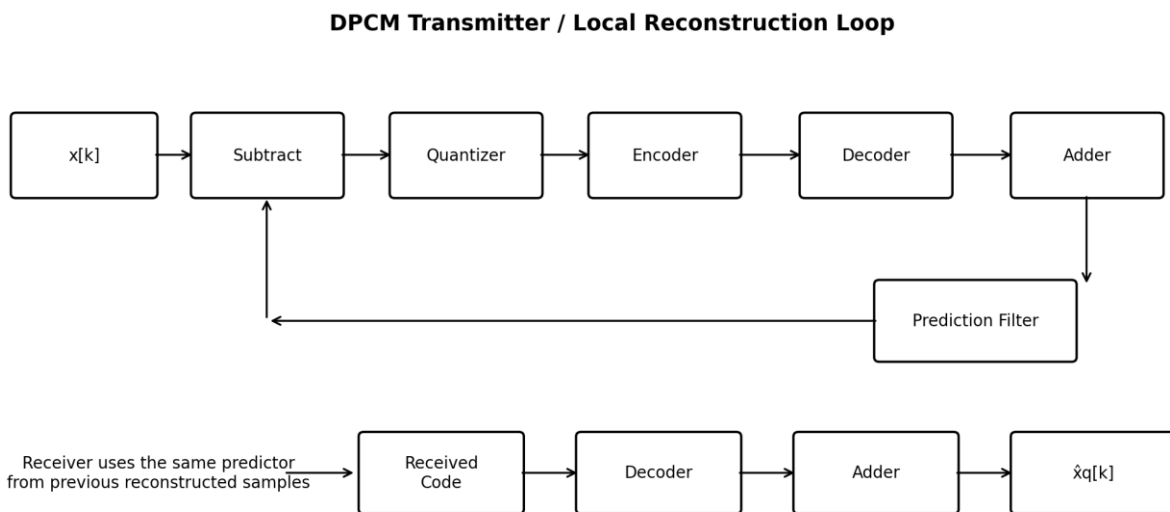


Fig. 3.7. DPCM transmitter with local reconstruction and the corresponding receiver principle.

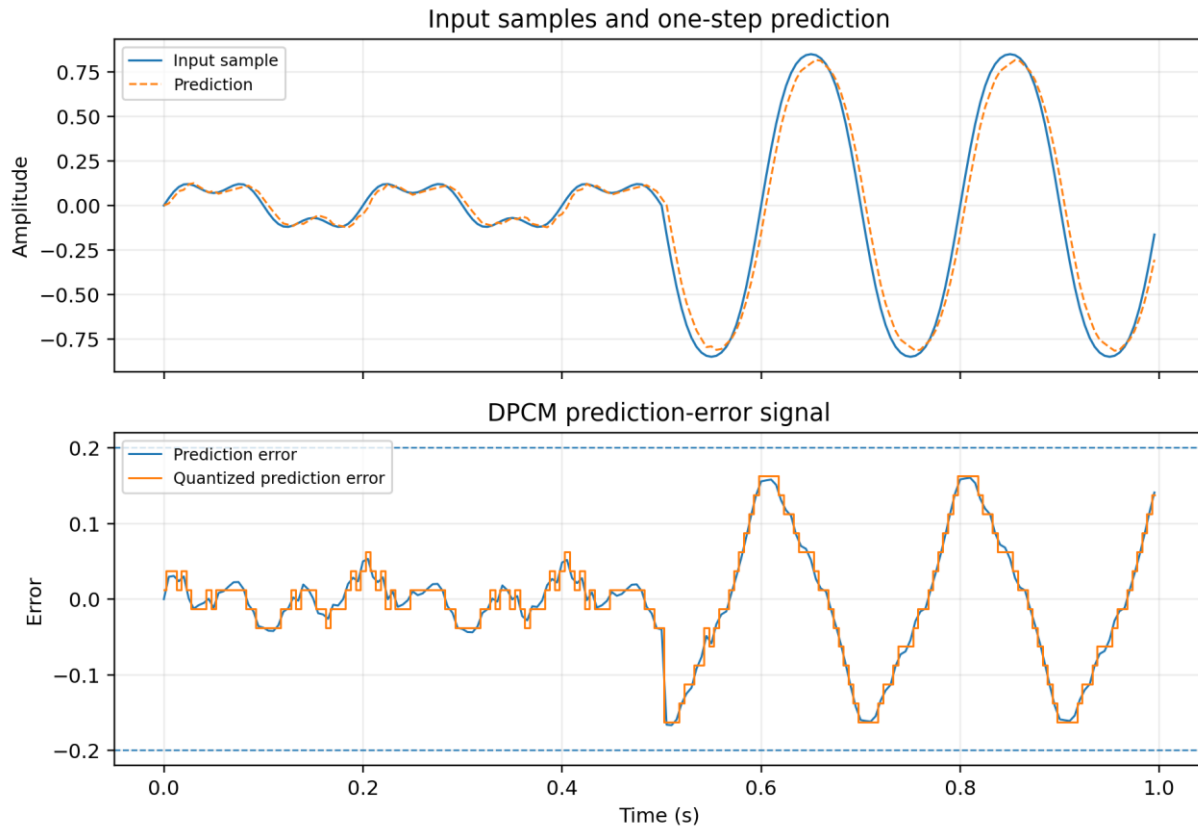


Fig. 3.8. Input prediction and the resulting DPCM prediction-error signal.

2.8. DPCM Performance Measures

- The principal benefit is obtained when the variance and peak magnitude of $e[k]$ are lower than those of $x[k]$. In this laboratory, prediction gain is evaluated as

$$G_p = \text{var}\{x[k]\} / \text{var}\{e[k]\}$$

- The reconstruction mean-square error is

$$\text{MSE} = \text{mean}\{[x[k] - x_q[k]]^2\}$$

- An appropriate predictor coefficient reduces the prediction-error range and allows a finer DPCM quantizer step. A poor predictor or an excessively narrow prediction-error range may cause overload and increase distortion.

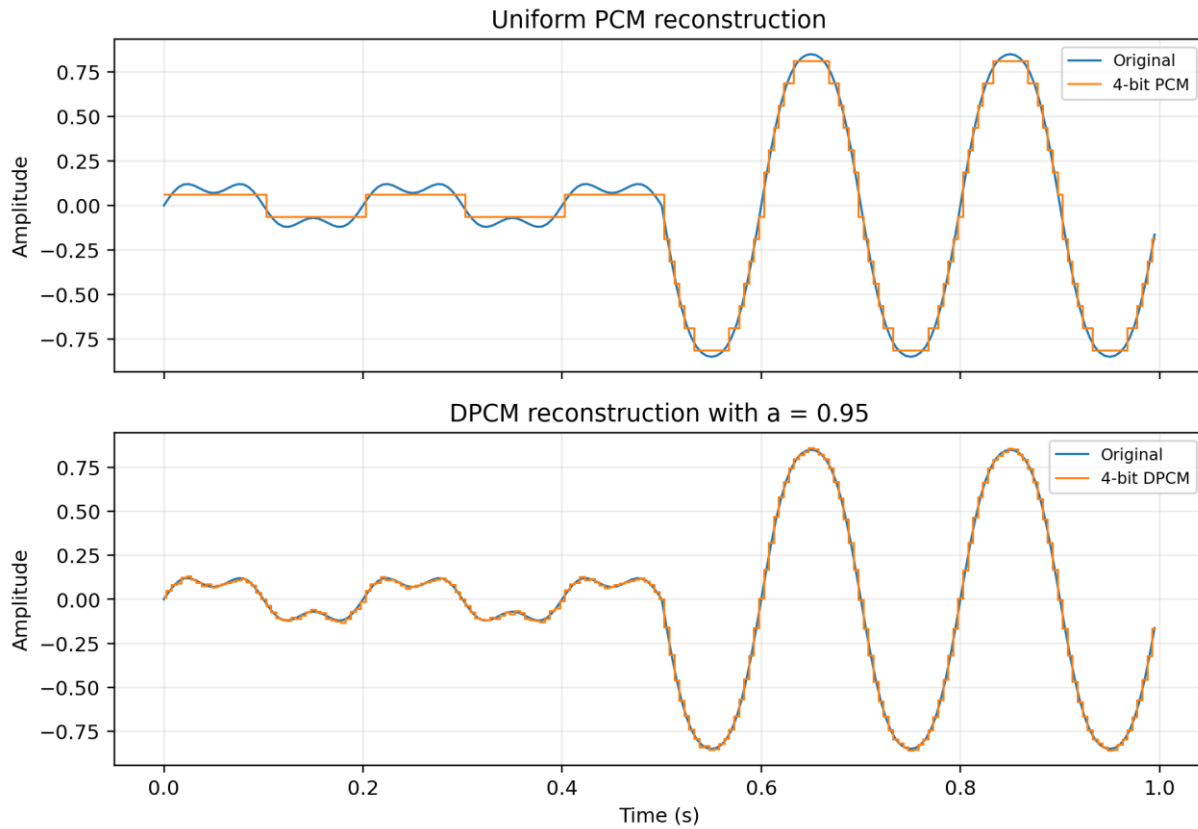


Fig. 3.9. Comparison of 4-bit uniform PCM and 4-bit DPCM reconstruction.

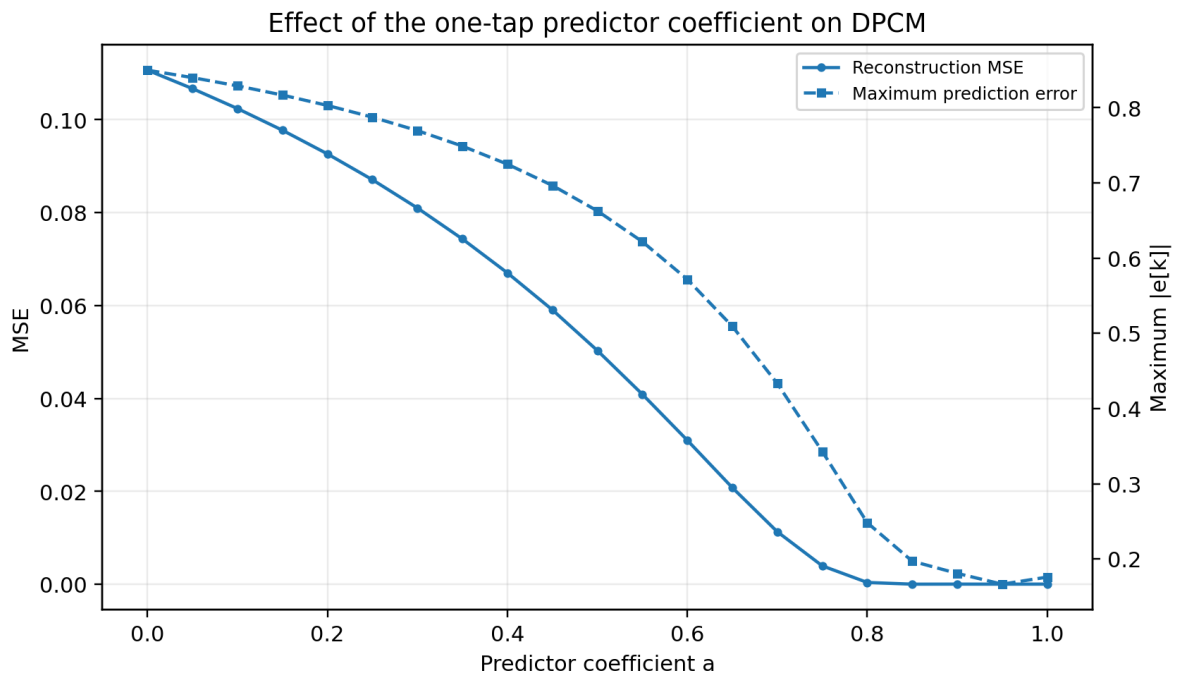


Fig. 3.10. Effect of predictor coefficient on DPCM error range and reconstruction MSE.

3. Materials Used

Computer

MATLAB software

MATLAB Script Editor

Digital Communications lecture notes

Laboratory worksheet

Calculator

- The main MATLAB commands and operations used in this experiment are:

sign, abs, log, log1p, exp and power operations

floor, min, max and logical indexing

zeros, diff, var, mean and log10

plot, stem, stairs, subplot, grid and legend

for loops and piecewise function implementation

4. Preliminary (PreLab)

- Use the sampled test signal

$$x(t) = A(t)\sin(2\pi \cdot 5t) + 0.05\sin(2\pi \cdot 15t)$$

$$A(t) = 0.12, \quad 0 \leq t < 0.5 \text{ s}; \quad A(t) = 0.90, \quad 0.5 \leq t < 1 \text{ s}$$

- with sampling frequency $f_s = 200$ Hz and normalized quantizer range $-1 \leq x \leq 1$.
- Verify that the selected sampling frequency satisfies the sampling theorem for the test signal.
 - For a 4-bit uniform quantizer over -1 to 1 , calculate the number of levels and step size.
 - Write the μ -law compressor and expander equations for $\mu = 255$.
 - Write the A-law compressor and expander equations for $A = 87.6$.
 - Explain why the low-amplitude interval is expected to benefit more from companding than the high-amplitude interval.
 - For a first-order DPCM predictor $\hat{x}[k] = ax_q[k-1]$, explain the expected prediction for $a = 0$ and $a = 1$.
 - Using an estimated prediction-error range of -0.2 to 0.2 and 4 bits, calculate the DPCM quantizer step size.
 - Compare the 4-bit PCM step size with the 4-bit DPCM error-quantizer step size.
 - Explain why the DPCM transmitter must contain a local reconstruction loop.

Table 3.1. Preliminary quantizer calculations

System	Input Range	Bits	Levels	Step Size
Uniform PCM	-1 to 1	4		
DPCM error quantizer	-0.2 to 0.2	4		

Table 3.2. Expected qualitative comparison

Criterion	Uniform PCM	μ -Law PCM	A-Law PCM	DPCM
Low-amplitude resolution				
High-amplitude resolution				
Quantity being quantized	Signal sample	Compressed sample	Compressed sample	Prediction error
Main design parameter	Input range	μ	A	Predictor coefficient and error range

5. Procedure

Part 1: Generate the Dynamic-Range Test Signal

1. Open MATLAB and create a new script file.
2. Define $f_s = 200$ Hz and a one-second observation interval.
3. Generate the piecewise amplitude function $A(t)$.
4. Generate the test signal and plot the low- and high-amplitude intervals.
5. Calculate the peak magnitude and verify that the signal lies within the normalized range.

```
fs = 200;
t = 0:1/fs:1-1/fs;
Aenv = 0.12*ones(size(t));
Aenv(t>=0.5) = 0.90;
x = Aenv.*sin(2*pi*5*t) + 0.05*sin(2*pi*15*t);
peak_x = max(abs(x));
```

Part 2: Four-Bit Uniform PCM Reference

1. Implement a 4-bit mid-rise uniform quantizer over -1 to 1 .
2. Quantize the test signal and calculate the quantization error.
3. Calculate MSE and SQNR for the complete signal.
4. Calculate SQNR separately over 0 to 0.5 s and 0.5 to 1 s.
5. Plot the original and uniformly quantized signals.

```
n = 4; xmin = -1; xmax = 1;
L = 2^n; Delta = (xmax-xmin)/L;
index = floor((min(max(x,xmin),xmax-eps)-xmin)/Delta);
xq_uniform = xmin + (index+0.5)*Delta;
e_uniform = x-xq_uniform;
```

Part 3: μ -Law Companding PCM

1. Compress the normalized signal using $\mu = 255$.
2. Quantize the compressed signal with the same 4-bit uniform quantizer over -1 to 1 .
3. Expand the quantized compressed signal using the inverse μ -law function.

4. Calculate the reconstruction error, MSE, and SQNR.
5. Calculate low-amplitude and high-amplitude SQNR separately.
6. Plot the compressor output, quantized compressed signal, and expanded reconstruction.

```
mu = 255;
y_mu = sign(x).*log(1+mu*abs(x))/log(1+mu);
idx_mu = floor((min(max(y_mu,-1),1-eps)+1)/Delta);
yq_mu = -1 + (idx_mu+0.5)*Delta;
xq_mu = sign(yq_mu).*((1+mu).^abs(yq_mu)-1)/mu;
```

Part 4: A-Law Companding PCM

1. Implement the piecewise A-law compressor with $A = 87.6$.
2. Quantize the compressed signal with the 4-bit uniform quantizer.
3. Implement the inverse A-law expander.
4. Calculate the reconstruction error, MSE, and SQNR.
5. Calculate low-amplitude and high-amplitude SQNR separately.
6. Compare the result with uniform PCM and μ -law PCM.

```
Aval = 87.6; ax = abs(x);
y_A = zeros(size(x));
k1 = ax < 1/Aval;
y_A(k1) = Aval*ax(k1)/(1+log(Aval));
y_A(~k1) = (1+log(Aval*ax(~k1)))/(1+log(Aval));
y_A = sign(x).*y_A;
```

```
idx_A = floor((min(max(y_A,-1),1-eps)+1)/Delta);
yq_A = -1 + (idx_A+0.5)*Delta;
ay = abs(yq_A); xq_A = zeros(size(yq_A));
k2 = ay < 1/(1+log(Aval));
xq_A(k2) = ay(k2)*(1+log(Aval))/Aval;
xq_A(~k2) = exp(ay(~k2)*(1+log(Aval))-1)/Aval;
xq_A = sign(yq_A).*xq_A;
```

Part 5: Quantitative Companding Comparison

1. Create one figure containing the original, uniform PCM, μ -law PCM, and A-law PCM signals over the low-amplitude interval.
2. Create the same comparison over the high-amplitude interval.
3. Plot the three error signals over the full observation interval.
4. Tabulate complete-signal MSE and SQNR.
5. Tabulate low- and high-amplitude SQNR.
6. Explain whether companding improves every region equally.

```
low = t < 0.5; high = t >= 0.5;
SQNR_low_mu = 10*log10(mean(x(low).^2)/mean((x(low)-xq_mu(low)).^2));
SQNR_high_mu = 10*log10(mean(x(high).^2)/mean((x(high)-xq_mu(high)).^2));
```

Part 6: DPCM Prediction and Error Signal

1. Use the first-order predictor coefficient $a = 0.95$.
2. Initialize the predictor and reconstructed sample arrays to zero.
3. For each sample, calculate the prediction from the previous reconstructed sample.
4. Calculate the prediction error $e[k]$.
5. Plot the input, prediction, and prediction error.
6. Calculate the variance and maximum magnitude of the original signal and prediction error.
7. Calculate the experimental prediction gain $G_p = \text{var}(x)/\text{var}(e)$.

```
a = 0.95;
xpred = zeros(size(x));
e = zeros(size(x));
xrec = zeros(size(x));
for k = 1:length(x)
    if k > 1, xpred(k) = a*xrec(k-1); end
    e(k) = x(k)-xpred(k);
    xrec(k) = x(k); % temporary ideal reconstruction
end
Gp = var(x)/var(e);
```

Part 7: Four-Bit DPCM Encoding and Reconstruction

1. Set the prediction-error quantizer range to -0.2 to 0.2 and use 4 bits.
2. Quantize each prediction-error sample inside the feedback loop.
3. Add the quantized prediction error to the predicted sample to obtain the local reconstruction.
4. Store the quantizer index as the transmitted DPCM codeword.
5. Reconstruct the signal at the receiver using the same predictor.
6. Calculate DPCM MSE and SQNR.
7. Compare 4-bit PCM and 4-bit DPCM reconstruction quality.
8. Check whether prediction-error overload occurs.

```
emin = -0.2; emax = 0.2;
Delta_e = (emax-emin)/2^n;
xpred = zeros(size(x)); e = zeros(size(x));
eq = zeros(size(x)); xrec = zeros(size(x));
for k = 1:length(x)
    if k > 1, xpred(k) = a*xrec(k-1); end
    e(k) = x(k)-xpred(k);
    idx = floor((min(max(e(k),emin),emax-eps)-emin)/Delta_e);
    eq(k) = emin + (idx+0.5)*Delta_e;
    xrec(k) = xpred(k)+eq(k);
end
```

Part 8: Predictor-Coefficient and Error-Range Tests

1. Repeat DPCM for $a = 0, 0.5, 0.8, 0.9, 0.95$, and 1.0 .
 2. Record the maximum prediction error, prediction gain, and reconstruction MSE.
 3. Plot MSE against predictor coefficient.
 4. Repeat the $a = 0.95$ experiment using error ranges $\pm 0.1, \pm 0.2$, and ± 0.4 .
 5. Identify overload for the narrow range and coarse quantization for the wide range.
 6. Determine the best tested predictor coefficient and error range for the selected signal.
6. Evaluation of Results
- Complete the following tables using MATLAB results. Attach all requested plots and explain the observed tradeoffs.

Table 3.3. Uniform and companded PCM performance

Method	MSE	Overall SQNR (dB)	Low-Level SQNR (dB)	High-Level SQNR (dB)	Maximum Error
Uniform PCM					
μ -law PCM					
A-law PCM					

Table 3.4. Companding observations

Observation	Uniform PCM	μ -Law PCM	A-Law PCM
Resolution near zero			
Resolution at high amplitude			
Best-performing interval			
Main design parameter	Input range	$\mu =$	$A =$

Table 3.5. PCM and DPCM comparison

Quantity	Uniform PCM	DPCM
Quantized variable	$x[k]$	$e[k]$
Quantizer input range	-1 to 1	-0.2 to 0.2
Bits per sample	4	4
Step size		
MSE		
SQNR (dB)		
Maximum quantizer-input magnitude		

Quantity	Uniform PCM	DPCM
Overload observed?		

Table 3.6. DPCM predictor-coefficient sweep

a	Maximum $ e[k] $	var(x)	var(e)	Prediction Gain	Reconstruction MSE
0					
0.5					
0.8					
0.9					
0.95					
1.0					

Table 3.7. DPCM error-range test for $a = 0.95$

Error Range	Step Size	Overloaded Samples	MSE	Interpretation
-0.1 to 0.1				
-0.2 to 0.2				
-0.4 to 0.4				

- Answer the following questions:
 1. Why is a fixed uniform step size unsuitable for signals with a very large dynamic range?
 2. How does companding produce an effective non-uniform quantizer while still using a uniform quantizer internally?
 3. Why are μ -law and A-law input signals normalized before compression?
 4. Compare the linear and logarithmic regions of the A-law characteristic.
 5. Which method gave the best low-amplitude SQNR? Explain the result using the compressor characteristic.
 6. Did companding reduce the error in the high-amplitude interval? Explain why the result may differ from the low-amplitude interval.
 7. Why is overall MSE alone insufficient for evaluating companding performance?
 8. What redundancy in a conventional PCM sample sequence is used by DPCM?
 9. Why must the DPCM transmitter use reconstructed rather than original previous samples in its predictor?

10. Compare the dynamic range and quantizer step size of the original-signal and prediction-error quantizers.
11. How did the predictor coefficient affect prediction-error magnitude and reconstruction MSE?
12. Why can a prediction-error quantizer with a very narrow range cause slope-like or overload distortion?
13. Why does an excessively wide DPCM error range reduce the advantage of prediction?
14. Under what signal conditions would DPCM provide little or no improvement over ordinary PCM?
15. Summarize the different purposes of companding and DPCM in one paragraph.

- Submit the following:

MATLAB script file with the .m extension

Compressor characteristics, quantized signals, error signals, SQNR comparison, DPCM prediction, and reconstruction figures

Completed Tables 3.1–3.7

First ten DPCM prediction errors, quantized errors, code indices, and reconstructed samples

Answers to all evaluation questions and a brief comparison of uniform PCM, companded PCM, and DPCM

Experiment 4: Line Coding and Time-Division Multiplexing

1. Objective of the Experiment

- The objective of this experiment is to investigate how binary information is represented by baseband line-coded waveforms and how several sampled message signals are transmitted over a common channel by time-division multiplexing (TDM). The first part introduces unipolar, polar, biphase, and bipolar line codes. Students will generate the waveforms in MATLAB, identify their transition rules, examine their DC content and frequency spectra, and evaluate their suitability for timing recovery.
- The second part implements a synchronized sample-interleaved TDM system. Three band-limited message signals will be sampled, assigned to successive time slots, combined into one multiplexed sequence, separated by a demultiplexer, and numerically reconstructed. The effect of a time-slot synchronization error will also be examined.
- At the end of the experiment, students are expected to:

Define line coding and explain its role in a digital baseband link.

Generate NRZ-L, NRZ-M, NRZ-S, RZ, biphase, differential Manchester, and bipolar waveforms.

Interpret how logical 1 and logical 0 are represented by levels or transitions.

Compare DC content, transition density, and approximate bandwidth using FFT results.

Explain why long runs of equal bits can create synchronization difficulties for some codes.

Calculate frame rate, frame duration, slot duration, and aggregate sample rate for a TDM system.

Construct a sample-interleaved TDM sequence and recover each channel by deinterleaving.

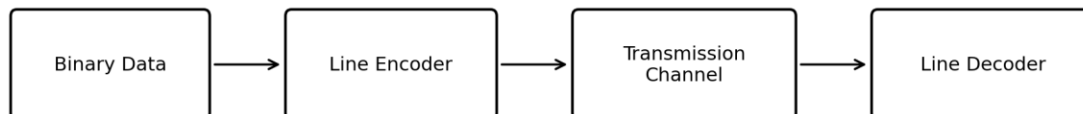
Compare original and reconstructed signals and identify the consequence of slot synchronization error.

2. Theoretical Background

2.1. Line Coding

- The lecture notes define line coding as the selection of a voltage or current waveform pattern used to represent the binary symbols 1 and 0 on a transmission link. Line coding is widely used for digital data transport, especially in short-distance computer communication networks. The line encoder maps the information sequence into a baseband waveform, and the receiver interprets the waveform according to the selected coding rule.

Line Coding in a Digital Baseband Link



Binary symbols are represented by a controlled voltage or current waveform.

Fig. 4.1. Functional role of a line encoder and line decoder in a digital baseband link.

- For a binary bit rate R_b , the bit interval is

$$T_b = 1 / R_b$$

- The waveform within every interval T_b depends on both the current data bit and, for transition-based codes, the previous line state. Important design considerations include the presence of a DC component, the number of transitions available for clock synchronization, occupied bandwidth, error visibility, and implementation complexity.

2.2. Non-Return-to-Zero Codes

- In a non-return-to-zero (NRZ) code, the signal does not deliberately return to zero during the bit interval. The lecture notes list the following forms:

NRZ-L: the logical value is represented by the line level. In the polar implementation used in this experiment, logic 1 is $+A$ and logic 0 is $-A$.

NRZ-M: logic 1 forces a transition at the beginning of the bit interval, while logic 0 produces no transition.

NRZ-S: logic 0 forces a transition at the beginning of the bit interval, while logic 1 produces no transition.

Unipolar NRZ: logic 1 is represented by $+A$ and logic 0 by zero; it is included to demonstrate the strong DC component of a single-polarity code.

- NRZ codes use one constant level per bit and are therefore simple and relatively bandwidth-efficient. However, long runs without transitions can make timing recovery difficult.

2.3. Return-to-Zero and Biphasic Codes

- In return-to-zero (RZ) coding, the active pulse returns to zero before the end of the bit interval. The shorter pulse duration increases the number of transitions and generally broadens the spectrum compared with NRZ coding.
- Biphasic codes contain a transition in the middle of every bit interval. The lecture notes describe Biphasic-L, Biphasic-M, Biphasic-S, and Differential Manchester forms. In this experiment:

Biphasic-L uses opposite half-bit polarities for logic 1 and logic 0. It is commonly recognized as a Manchester-type level code.

Biphasic-M always has a middle transition; logic 1 also forces a transition at the beginning of the bit interval.

Biphasic-S always has a middle transition; logic 0 also forces a transition at the beginning of the bit interval.

Differential Manchester is interpreted from transitions rather than absolute polarity, which makes the code insensitive to a complete polarity reversal of the line.

- The mandatory middle-bit transition provides a timing reference for every symbol. The cost of this self-clocking property is a larger occupied bandwidth.

2.4. Bipolar Line Coding

- The lecture notes describe bipolar coding as an alternating-pulse code. In bipolar AMI, logic 0 is represented by zero level, while successive logic-1 pulses alternate between $+A$ and $-A$. The alternating

polarity reduces the average DC value when the data contain logic 1s and allows a violation of the alternation rule to be used as an error indication in practical systems.

Line-coded waveforms for the sequence 1 0 1 1 0 0 1 0
(a) NRZ and RZ codes

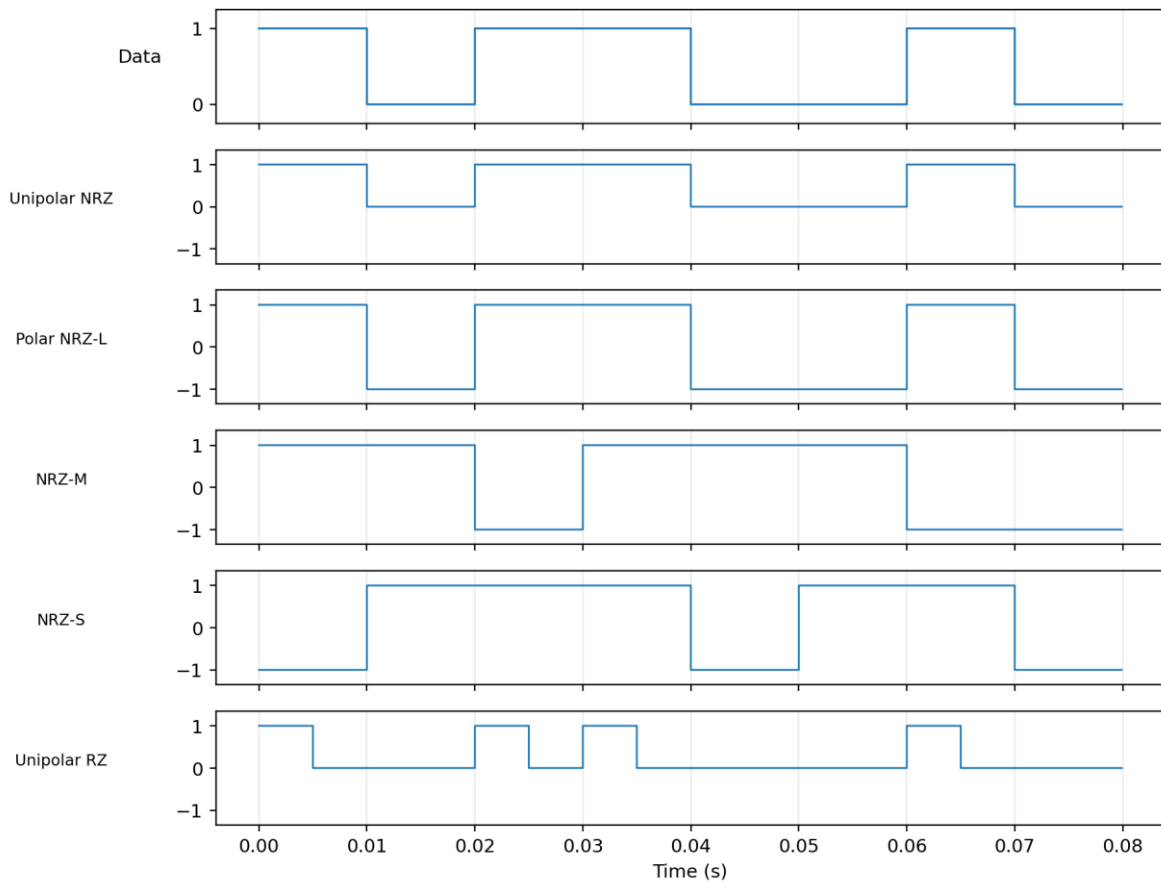


Fig. 4.2(a). NRZ and RZ line-coded waveforms for the same binary input sequence.

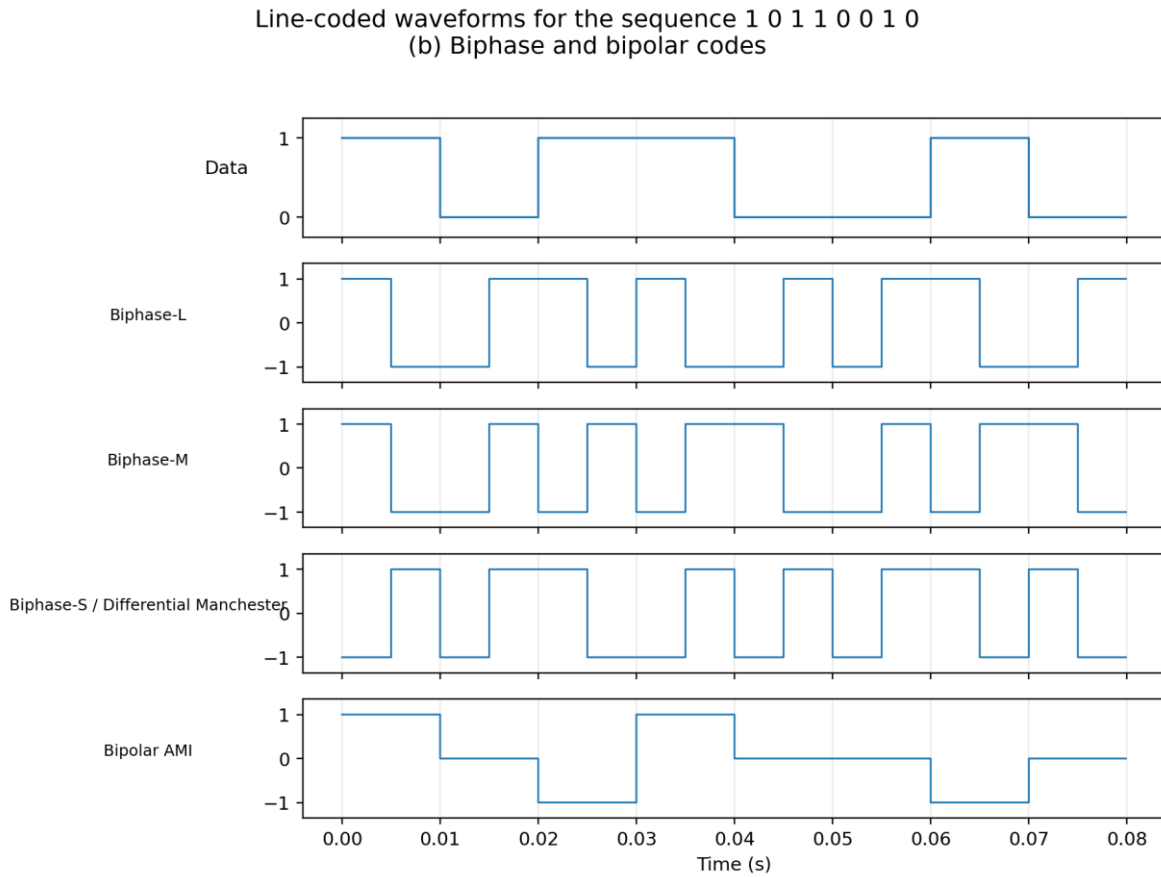


Fig. 4.2(b). Biphas and bipolar line-coded waveforms for the same binary input sequence.

2.5. Frequency-Domain Characteristics

- The spectrum of a line-coded waveform depends on the pulse shape, transition rule, and data pattern. A nonzero average value appears as a component near zero frequency. Codes containing both positive and negative levels can reduce this component. Codes with mandatory within-bit transitions contain more high-frequency content and generally require more bandwidth.
- For a sampled MATLAB waveform $x[n]$ containing N samples, the centered numerical magnitude spectrum is obtained from

$$|X(f)| = |\text{fftshift}\{\text{FFT}\{x[n]\}\}| / N$$

- The FFT of a finite periodic test sequence is used only as a numerical comparison. It is not a complete statistical power spectral density model for random data.

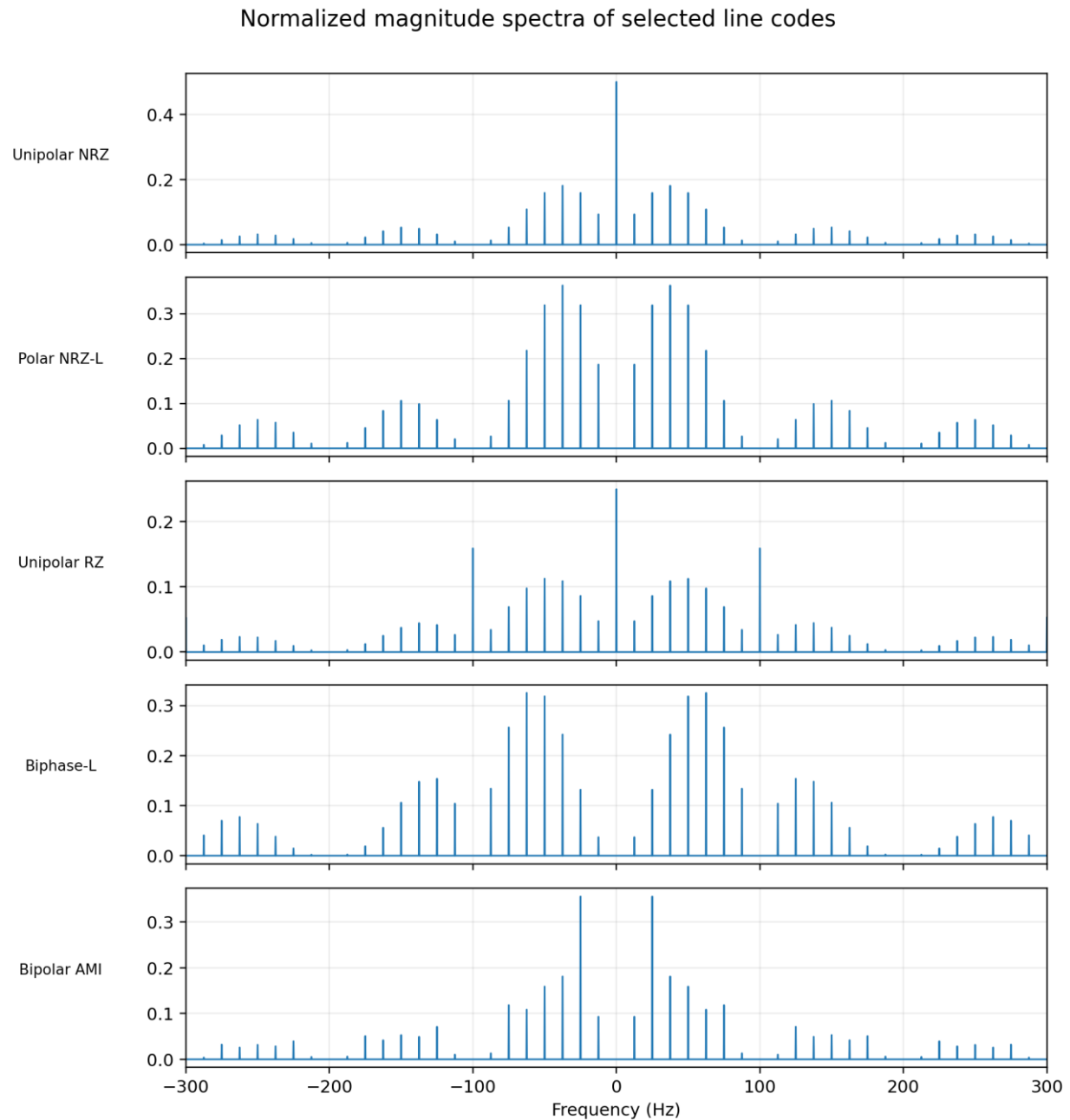


Fig. 4.3. Numerical magnitude spectra of representative line codes.

2.6. Timing Information and Long Runs

- A receiver normally derives the sampling instants from transitions in the received waveform. A line code that can remain constant for many consecutive bits provides little timing information during those intervals. Biphas codes contain a middle transition in every bit and therefore preserve a regular timing reference even when the data contain long runs of identical symbols.

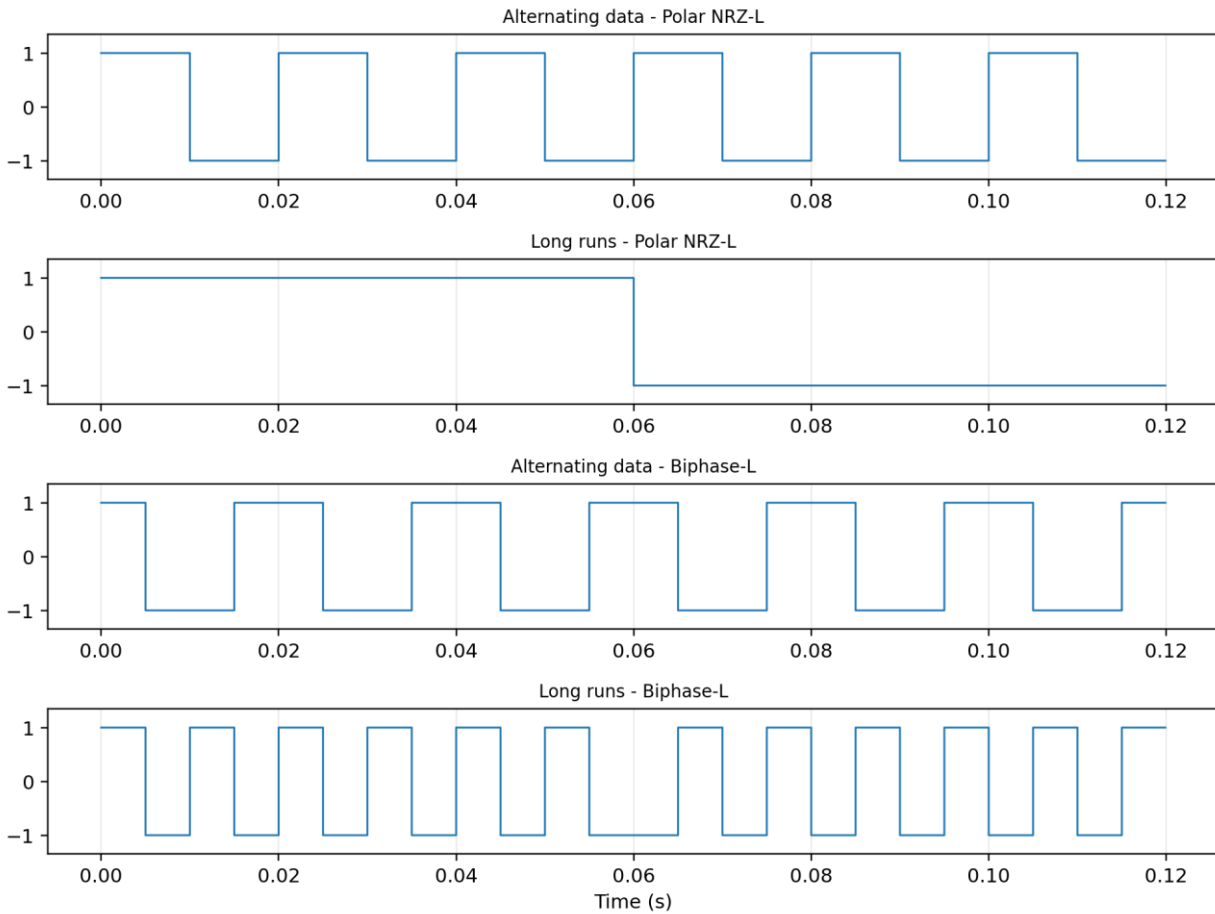
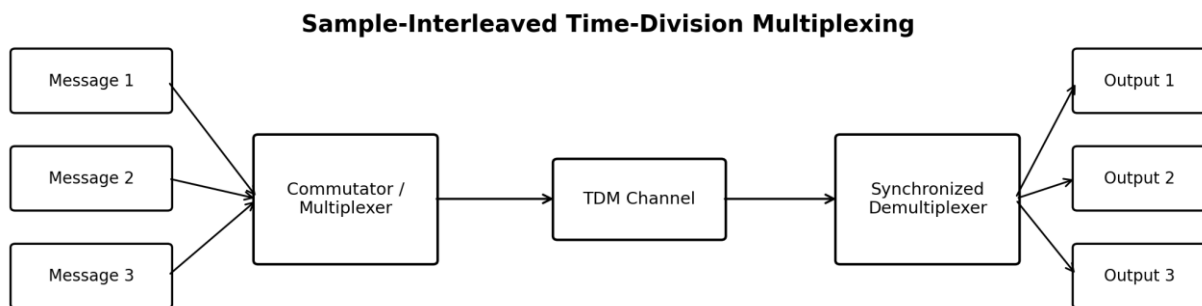


Fig. 4.4. Comparison of transition availability for alternating data and long runs.

2.7. Time-Division Multiplexing

- Time-division multiplexing transmits several independent message signals over one common channel by assigning each source a separate time slot. According to the lecture notes, each input message is first restricted in bandwidth by a low-pass anti-aliasing filter and then sampled periodically. A commutator sequentially selects one sample from each input channel to form an interleaved pulse-amplitude sequence. At the receiver, a synchronized demultiplexer distributes the samples to the corresponding output channels, and reconstruction filters recover the message signals.



Each frame contains one time slot for every input channel.

Fig. 4.5. Functional block diagram of a synchronized sample-interleaved TDM system.

- For M channels with one sample from every channel in each frame, the per-channel sampling frequency and frame frequency are related by

$$f_{\text{frame}} = f_{s,\text{ch}}$$

- The total slot or sample rate on the multiplexed link is

$$R_{\text{slot}} = M f_{s,\text{ch}}$$

- The frame and slot durations are

$$T_{\text{frame}} = 1 / f_{\text{frame}}, \quad T_{\text{slot}} = T_{\text{frame}} / M$$

- If every sample is represented by n binary bits in a PCM-TDM system, the uncoded aggregate bit rate is

$$R_{b,\text{TDM}} = M n f_{s,\text{ch}}$$

- Correct operation requires both sampling-theorem compliance for every input signal and synchronization between the transmitter commutator and receiver demultiplexer. A slot displacement can route samples to the wrong channels even if the transmitted sample values themselves are unchanged.

3. Materials Used

Computer

MATLAB software

MATLAB Script Editor

Digital Communications lecture notes

Laboratory worksheet

Calculator

- The main MATLAB functions used in this experiment are:

repelem, repmat, reshape, and kron

stairs, stem, plot, and subplot

fft, fftshift, fftfreq construction, and abs

mean, diff, sum, and find

interp1 for numerical reconstruction

mod, xor, and logical indexing

4. Preliminary (PreLab)

- Complete the following calculations and sketches before attending the laboratory.

4.1. Line-Coding Preparation

- Use the binary sequence

$$\mathbf{b} = [1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0]$$

- Assume $A = 1 \text{ V}$, $R_b = 100 \text{ bit/s}$, and the initial line state is $-A$ for transition-based codes.

- Calculate the bit duration T_b .

2. Draw the expected Unipolar NRZ, Polar NRZ-L, NRZ-M, NRZ-S, Unipolar RZ, Biphase-L, Biphase-M, Biphase-S, and Bipolar AMI waveforms.
3. Count the number of level transitions for each waveform. Do not count the assumed state before the first bit unless the first coding rule explicitly requires a beginning transition.
4. Predict which codes have a nonzero average level for this finite sequence.
5. Identify which codes contain a mandatory middle-bit transition.
6. Predict which codes will occupy the largest bandwidth.

Table 4.1. PreLab line-code predictions

Line Code	Levels / Transition Rule	Predicted Transitions	Predicted Mean	Timing Quality
Unipolar NRZ				
Polar NRZ-L				
NRZ-M				
NRZ-S				
Unipolar RZ				
Biphase-L				
Biphase-M				
Biphase-S				
Bipolar AMI				

4.2. TDM Preparation

- Three analog message signals will be multiplexed:

$$m_1(t) = \sin(2\pi \cdot 2t)$$

$$m_2(t) = 0.8 \cos(2\pi \cdot 3t)$$

$$m_3(t) = 0.6 \sin(2\pi \cdot 5t + \pi/6)$$

- The selected per-channel sampling frequency is $f_{s,ch} = 40$ Hz, and one sample from each channel is placed in every frame.
1. Determine the highest message frequency and verify the Nyquist condition for each channel.
 2. Calculate the frame frequency and frame duration.
 3. Calculate the total TDM slot rate and slot duration for $M = 3$ channels.
 4. Calculate the number of frames and the total number of transmitted sample slots during a 1 s observation interval.
 5. If each sample is encoded using 8 bits, calculate the aggregate PCM-TDM bit rate.
 6. Write the first twelve slot labels in transmission order.

Table 4.2. PreLab TDM calculations

Quantity	Symbol / Formula	Calculated Value	Unit
Maximum message frequency	$f_{\max}$		Hz
Per-channel sampling frequency	$f_{s, \text{ch}}$	40	Hz
Frame frequency	f_{frame}		frame/s
Frame duration	T_{frame}		s
Total slot rate	R_{slot}		sample/s
Slot duration	T_{slot}		s
Frames in 1 s			frame
Slots in 1 s			sample
8-bit PCM-TDM rate	$R_{\text{b, TDM}}$		bit/s

5. Procedure

Part 1: Generate the Binary Test Sequence

1. Open MATLAB and create a new script file.
2. Define $b = [1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0]$, $R_b = 100$ bit/s, and $\text{samplesPerBit} = 100$.
3. Construct a time vector with a sampling rate equal to $R_b \times \text{samplesPerBit}$.
4. Expand every bit into samplesPerBit numerical samples.
5. Plot the binary input waveform with bit boundaries visible.

```
b = [1 0 1 1 0 0 1 0];
Rb = 100; Tb = 1/Rb;
spb = 100; Fs = Rb*spb;
t = (0:length(b)*spb-1)/Fs;
data = repelem(b,spb);
```

Part 2: Generate NRZ, RZ, Biphase, and Bipolar Codes

1. Generate Unipolar NRZ using +A for logic 1 and zero for logic 0.
2. Generate Polar NRZ-L using +A for logic 1 and -A for logic 0.
3. Generate NRZ-M by forcing a beginning transition for every logic 1.
4. Generate NRZ-S by forcing a beginning transition for every logic 0.
5. Generate Unipolar RZ by returning a logic-1 pulse to zero after half of the bit interval.
6. Generate Biphase-L using opposite half-bit polarities for logic 1 and logic 0.
7. Generate Biphase-M and Biphase-S using a mandatory middle transition and the appropriate beginning-transition rule.
8. Generate Bipolar AMI by alternating the polarity of successive logic-1 pulses.

9. Display the input data and all line-coded waveforms in one multi-panel figure.
10. Check every waveform against the PreLab sketches.

```
A = 1;
unipolarNRZ = repelem(A*b,spb);
polarNRZL = repelem(2*b-1,spb);

% Unipolar RZ
unipolarRZ = zeros(1,length(b)*spb);
for k = 1:length(b)
    idx = (k-1)*spb + (1:spb);
    if b(k)==1, unipolarRZ(idx(1:spb/2)) = A; end
end
```

```
% Bipolar AMI
amiLevel = zeros(size(b));
polarity = -A;
for k = 1:length(b)
    if b(k)==1
        polarity = -polarity;
        amiLevel(k) = polarity;
    end
end
ami = repelem(amiLevel,spb);
```

Part 3: Transition Count, Mean Value, and Spectrum

1. For each waveform, calculate the sample mean as a numerical indication of DC content.
2. Count transitions by detecting changes between consecutive samples. For biphasic waveforms, include both beginning and middle transitions.
3. Repeat the eight-bit sequence 64 times to obtain a longer periodic record for FFT comparison.
4. Calculate and center the normalized FFT magnitude for every selected code.
5. Plot the spectra over $-3R_b$ to $+3R_b$.
6. Record the magnitude at or nearest zero frequency.
7. Estimate the frequency range containing the main spectral energy and compare NRZ, RZ, biphasic, and bipolar codes.
8. Explain why the finite repeated pattern produces discrete spectral lines.

```
transitionCount = sum(abs(diff(waveform)) > 1e-12);
meanValue = mean(waveform);
N = length(waveform);
X = abs(fftshift(fft(waveform)))/N;
f = (-N/2:N/2-1)*(Fs/N);
```

Part 4: Timing Information Test

1. Create an alternating sequence [1 0 1 0 ...] containing twelve bits.
2. Create a long-run sequence containing six logic 1s followed by six logic 0s.

3. Generate Polar NRZ-L and Biphas-L waveforms for both sequences.
4. Plot the four results in one figure.
5. Count the transitions in every result.
6. Explain which waveform preserves a regular clock indication during long runs.

Part 5: Generate and Multiplex Three Sampled Messages

1. Define the three message signals given in the PreLab section.
2. Use $f_{s,ch} = 40$ Hz and an observation interval of 1 s.
3. Calculate one sample sequence for every message channel.
4. Create the TDM sequence by placing the channel samples in the order ch1, ch2, ch3 for every frame.
5. Construct the slot-time axis using $R_{slot} = 3f_{s,ch}$.
6. Plot the three individual sample sequences and the multiplexed TDM sequence.
7. Write the first twelve multiplexed sample values and their channel labels.

```
fsCh = 40; M = 3;
ts = 0:1/fsCh:1-1/fsCh;
x1 = sin(2*pi*2*ts);
x2 = 0.8*cos(2*pi*3*ts);
x3 = 0.6*sin(2*pi*5*ts + pi/6);
frameMatrix = [x1; x2; x3];
tdm = frameMatrix(:).';
Rslot = M*fsCh;
tslot = (0:length(tdm)-1)/Rslot;
```

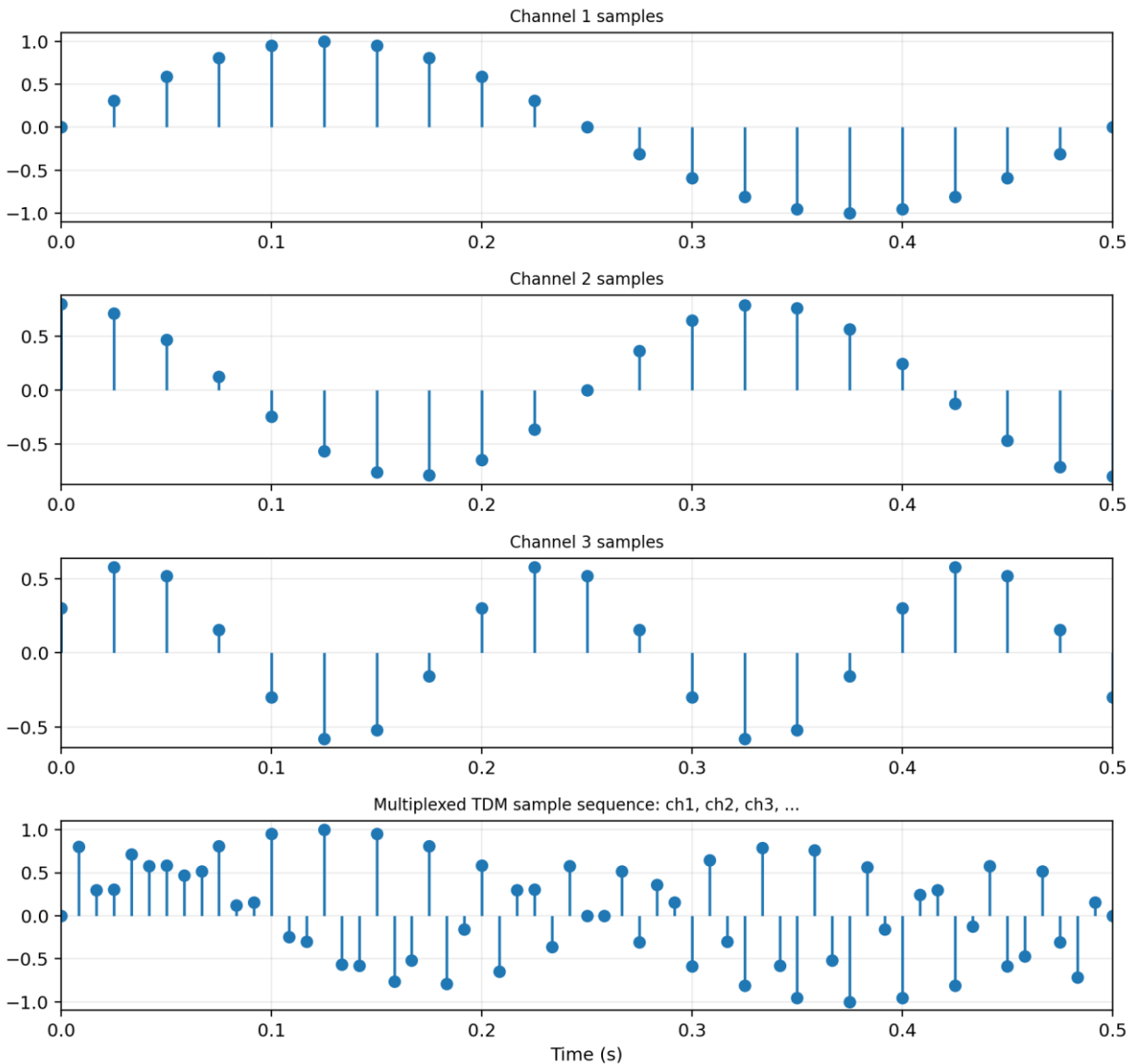


Fig. 4.6. Individual channel samples and the sample-interleaved TDM sequence.

Part 6: Demultiplexing and Reconstruction

1. Recover channel 1 from samples 1, 4, 7, ... of the TDM sequence.
2. Recover channel 2 from samples 2, 5, 8, ... and channel 3 from samples 3, 6, 9,
3. Verify that every recovered sample sequence is identical to the corresponding transmitter sample sequence.
4. Construct a dense display-time vector.
5. Use `interp1` with the `linear` or `pchip` option as a numerical approximation to reconstruction.
6. Plot original continuous-time reference signals, recovered samples, and reconstructed signals.
7. Calculate the MSE between each original reference waveform and its numerical reconstruction.
8. Explain that an actual analog receiver would use a low-pass reconstruction filter rather than `interp1`.

```
rx1 = tdm(1:M:end);
rx2 = tdm(2:M:end);
rx3 = tdm(3:M:end);
tFine = linspace(0,1,2000);
xlrec = interp1(ts,rx1,tFine,'pchip','extrap');
```

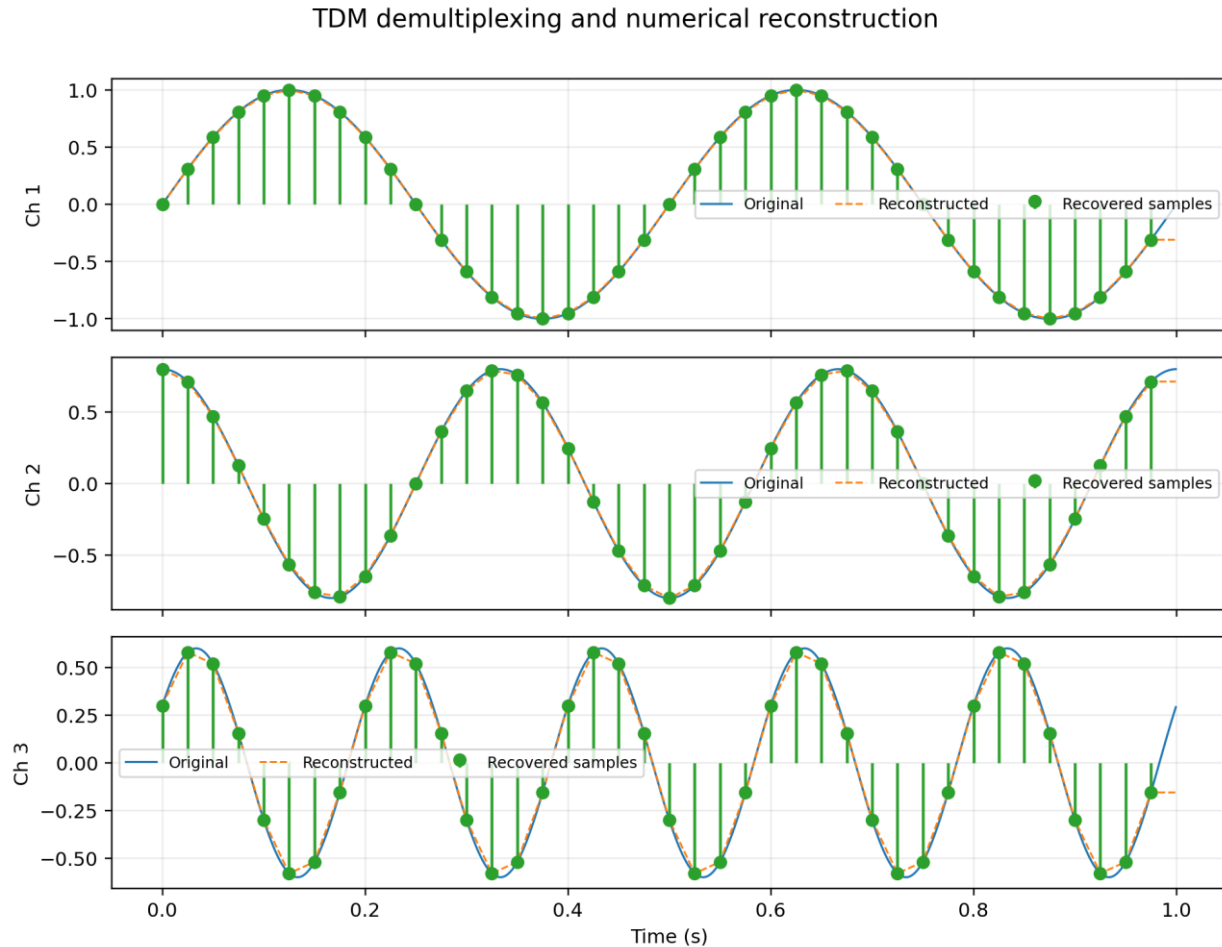


Fig. 4.7. Correct TDM demultiplexing and numerical reconstruction of the three message channels.

Part 7: Demultiplexer Synchronization Error

1. Circularly shift the received TDM sequence by one sample slot before demultiplexing.
2. Repeat the deinterleaving and reconstruction operations.
3. Compare every recovered output with the intended original channel.
4. Determine which transmitted channel now appears at each receiver output.
5. Explain why this is a synchronization or channel-assignment error rather than ordinary amplitude noise.
6. Restore the correct slot alignment and verify normal operation.

```
tdmShifted = circshift(tdm,1);
wrong1 = tdmShifted(1:M:end);
wrong2 = tdmShifted(2:M:end);
wrong3 = tdmShifted(3:M:end);
```

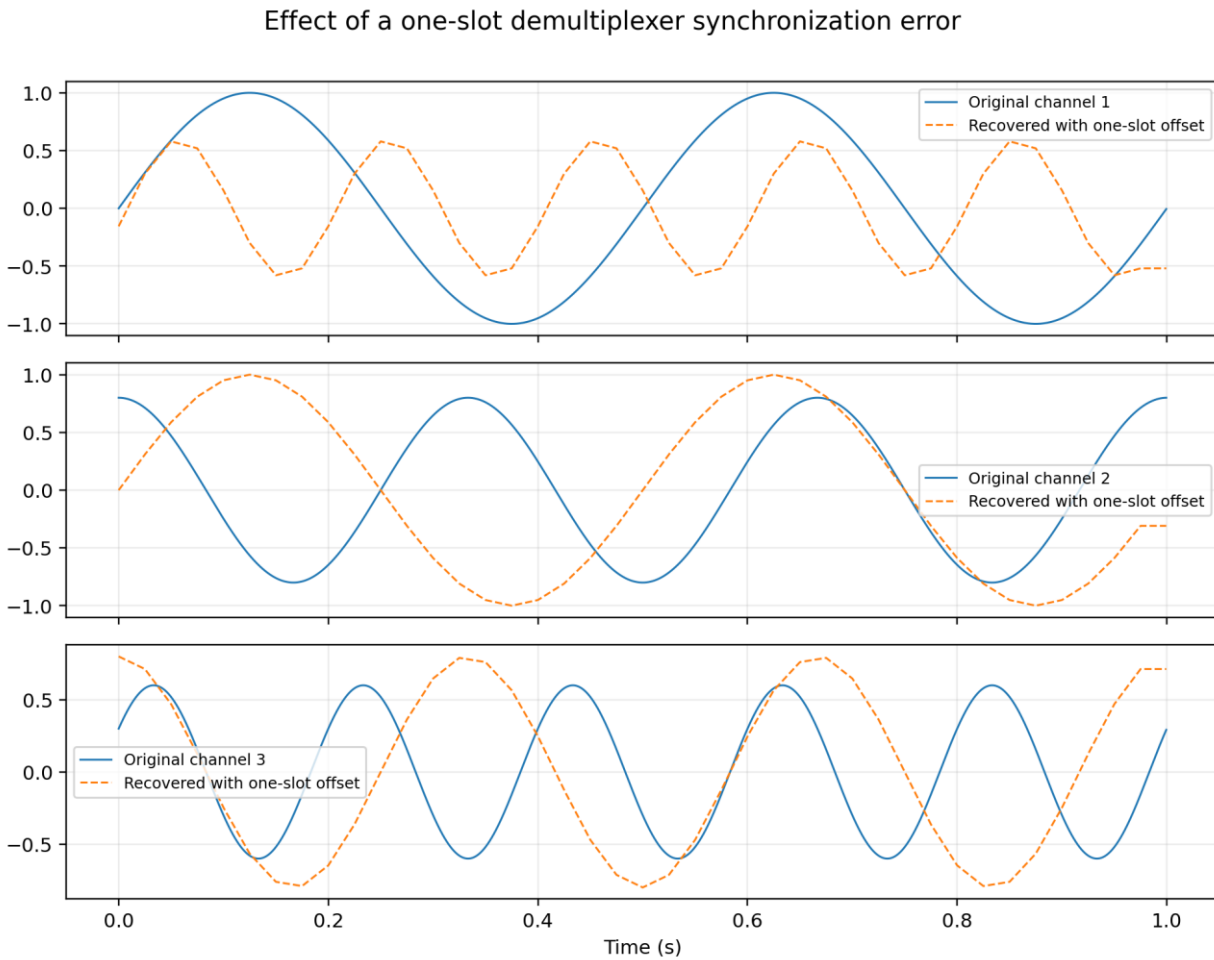


Fig. 4.8. Distortion and channel reassignment produced by a one-slot synchronization error.

6. Evaluation of Results

- Complete the following tables using the theoretical predictions and MATLAB results. Attach the requested plots and discuss all differences.

Table 4.3. Measured line-code properties

Line Code	Measured Mean	Transition Count	Zero-Frequency Magnitude	Relative Bandwidth	Clock Recovery
Unipolar NRZ					
Polar NRZ-L					
NRZ-M					
NRZ-S					
Unipolar RZ					
Biphase-L					
Biphase-M					

Line Code	Measured Mean	Transition Count	Zero-Frequency Magnitude	Relative Bandwidth	Clock Recovery
Biphase-S					
Bipolar AMI					

Table 4.4. Transition-density test

Data Pattern	Code	Number of Bits	Measured Transitions	Transitions per Bit	Observation
Alternating	Polar NRZ-L	12			
Long run	Polar NRZ-L	12			
Alternating	Biphase-L	12			
Long run	Biphase-L	12			

Table 4.5. TDM system parameters

Parameter	Theoretical Value	MATLAB Value	Unit
Number of channels			
Per-channel sample rate			Hz
Frame rate			frame/s
Frame duration			s
Slot rate			sample/s
Slot duration			s
Frames in 1 s			frame
Slots in 1 s			sample
8-bit PCM-TDM rate			bit/s

Table 4.6. Correct TDM reconstruction performance

Channel	Maximum Frequency	Sample Rate	Nyquist Ratio $f_s/(2f_{max})$	Sample Equality Verified?	Reconstruction MSE
1					
2					
3					

Table 4.7. One-slot synchronization-error mapping

Receiver Output	Intended Channel	Actual Dominant Source	MSE Relative to Intended	Interpretation
Output 1	Channel 1			
Output 2	Channel 2			
Output 3	Channel 3			

- Answer the following questions:

1. What is the difference between representing a bit by a level and representing it by a transition?
2. Why does Unipolar NRZ generally contain a stronger DC component than Polar NRZ-L?
3. Explain how NRZ-M and NRZ-S behave for long runs of the symbol that does not force a transition.
4. Why does RZ coding generally require more bandwidth than NRZ coding?
5. What timing advantage is provided by the mandatory middle transition of a biphasic code?
6. Why is Differential Manchester insensitive to a complete reversal of line polarity?
7. How does bipolar AMI reduce the average line level?
8. Which tested line code provided the best timing information, and which appeared most bandwidth-efficient?
9. Why does the FFT of the repeated finite bit pattern contain discrete lines rather than a smooth random-data spectrum?
10. Define a TDM frame and a TDM time slot for the three-channel experiment.
11. Why must every input signal be bandwidth-limited before sampling in a practical TDM system?
12. Why is the TDM slot rate three times the per-channel sampling rate in this experiment?
13. What information must the receiver know to demultiplex the TDM sequence correctly?
14. What happened when the demultiplexer was shifted by one time slot?
15. How would the aggregate bit rate change if the number of channels or the number of PCM bits per sample were doubled?
16. Summarize the separate purposes of line coding and time-division multiplexing in a digital communication system.

- Submit the following:

MATLAB script file with the .m extension

Line-code waveform, spectral-comparison, transition-density, TDM sample, reconstruction, and synchronization-error figures

Completed Tables 4.1-4.7

The first twelve TDM sample values with their frame and channel labels

Answers to all evaluation questions and a concise comparison of the tested line codes

Experiment 5: Delta Modulation and Adaptive Delta Modulation

1. Objective of the Experiment

The objective of this experiment is to investigate delta modulation (DM) and adaptive delta modulation (ADM) as one-bit differential waveform coding methods. Students will implement the transmitter and receiver algorithms in MATLAB, generate the binary decision sequence, reconstruct the input waveform from accumulated positive and negative steps, and evaluate the effects of step size and sampling frequency on reconstruction quality.

The experiment emphasizes the two principal distortions of conventional delta modulation: slope-overload distortion, which occurs when the staircase waveform cannot follow a rapidly changing input, and granular noise, which occurs when the step size is unnecessarily large for small input variations. An adaptive step-size algorithm is then applied to improve tracking over a wider dynamic range.

At the end of this experiment, students are expected to:

- Explain the operating principle of a one-bit delta modulator and its receiver.
- Generate the DM bit stream and staircase approximation of a sampled message signal.
- Calculate the maximum staircase slope and apply the slope-overload condition.
- Distinguish slope-overload distortion from granular noise in time-domain plots.
- Examine the effects of sampling frequency and fixed step size on reconstruction error.
- Implement a simple adaptive delta-modulation step-size controller in MATLAB.
- Compare fixed-step DM and ADM using mean-square error, reconstruction SNR, and bit-rate requirements.

2. Theoretical Background

2.1. Operating Principle of Delta Modulation

Delta modulation transmits only the direction of change between the current input sample and the previous staircase approximation. At each sampling instant, the input sample $x[n]$ is compared with the previous reconstructed value $\hat{x}[n-1]$. The comparison error is

$$e[n] = x[n] - \hat{x}[n-1]$$

A one-bit quantizer represents the sign of this error. A positive error produces an upward step and is encoded as binary 1; a negative error produces a downward step and is encoded as binary 0. For a fixed step size Δ , the quantizer output is

$$b[n] = +\Delta, e[n] \geq 0; \quad b[n] = -\Delta, e[n] < 0$$

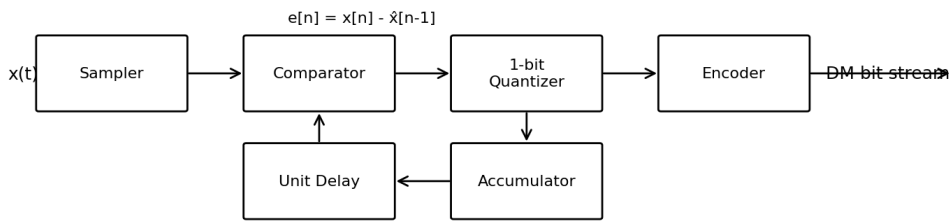
The staircase approximation is updated by accumulation:

$$\hat{x}[n] = \hat{x}[n-1] + b[n]$$

Because only one decision bit is transmitted for each sample, the signaling rate of a basic delta-modulation system is equal to the sampling frequency:

$$R_{b,DM} = f_s \text{ bit/s}$$

Delta Modulation Transmitter



Delta Modulation Receiver

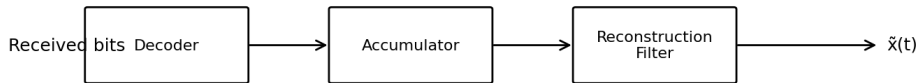


Fig. 5.1. Functional block diagrams of a delta-modulation transmitter and receiver.

2.2. Staircase Approximation and Reconstruction

The receiver converts each received bit into either $+\Delta$ or $-\Delta$ and accumulates these increments to reproduce the staircase waveform. A reconstruction low-pass filter may subsequently smooth the staircase and suppress high-frequency step components. The reconstructed waveform is an approximation because the transmitted data specify only the direction of change, not the exact sample amplitude.

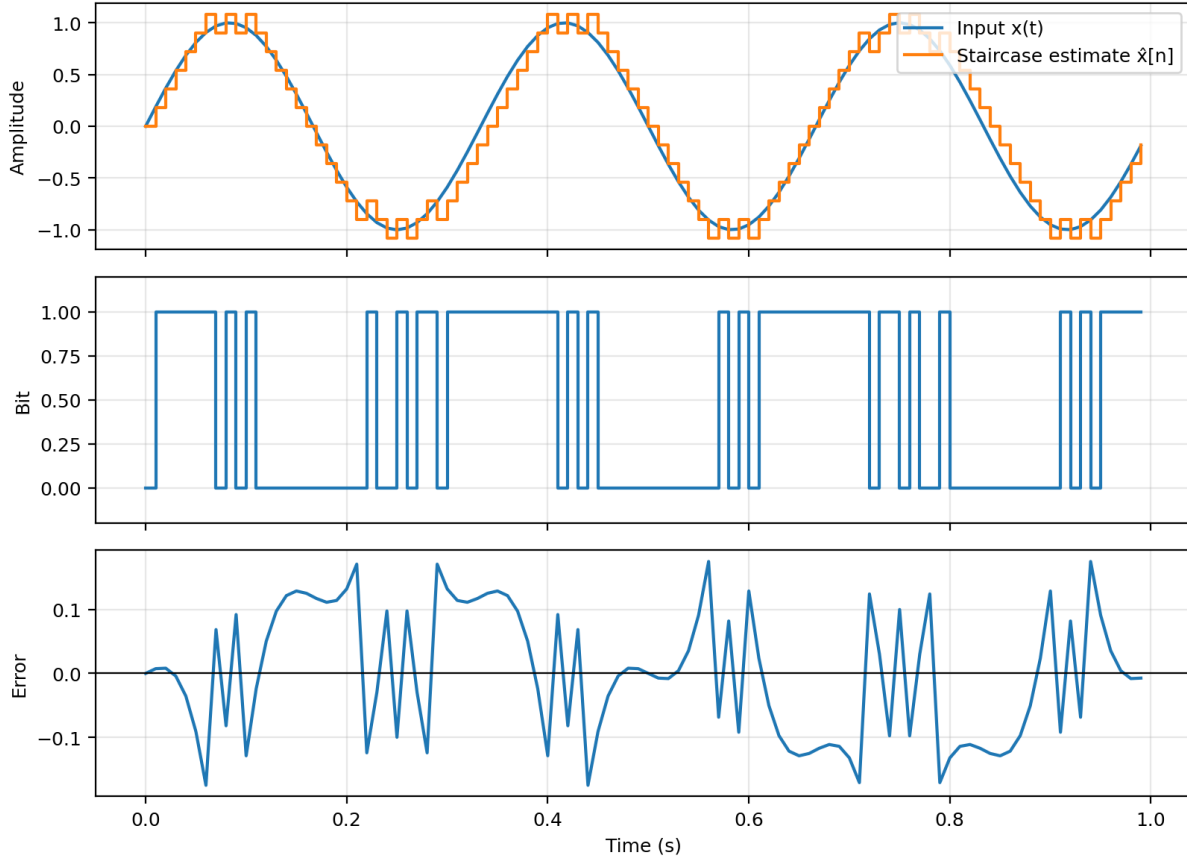


Fig. 5.2. Input signal, DM decision sequence, staircase estimate, and instantaneous reconstruction error.

2.3. Step Size and Maximum Tracking Slope

During one sampling interval T_s , the staircase output can change by at most Δ . Therefore, the maximum slope that the modulator can reproduce is

$$\text{Maximum staircase slope} = \Delta / T_s = \Delta f_s$$

For a sinusoidal input

$$x(t) = A_m \sin(2\pi f_m t)$$

the maximum magnitude of the input slope is

$$\max |dx(t)/dt| = 2\pi f_m A_m$$

To avoid slope-overload distortion, the staircase slope should satisfy

$$\Delta f_s \geq 2\pi f_m A_m$$

Equivalently, slope overload occurs when

$$A_m > \Delta / (2\pi f_m T_s)$$

2.4. Slope-Overload Distortion

Slope-overload distortion occurs when the input changes faster than the staircase waveform can follow. It is especially visible around the steepest portions of a high-amplitude or high-frequency message signal. A long run of identical decision bits is produced, but the accumulated steps still remain behind the input waveform.

2.5. Granular Noise (Hunting)

Granular noise occurs when the step size is large compared with small or slowly varying changes in the input. The staircase repeatedly moves above and below the message waveform, producing rapid alternation between binary 1 and binary 0. Increasing Δ reduces slope overload but generally increases granular noise; decreasing Δ reduces granular noise but increases the possibility of slope overload.

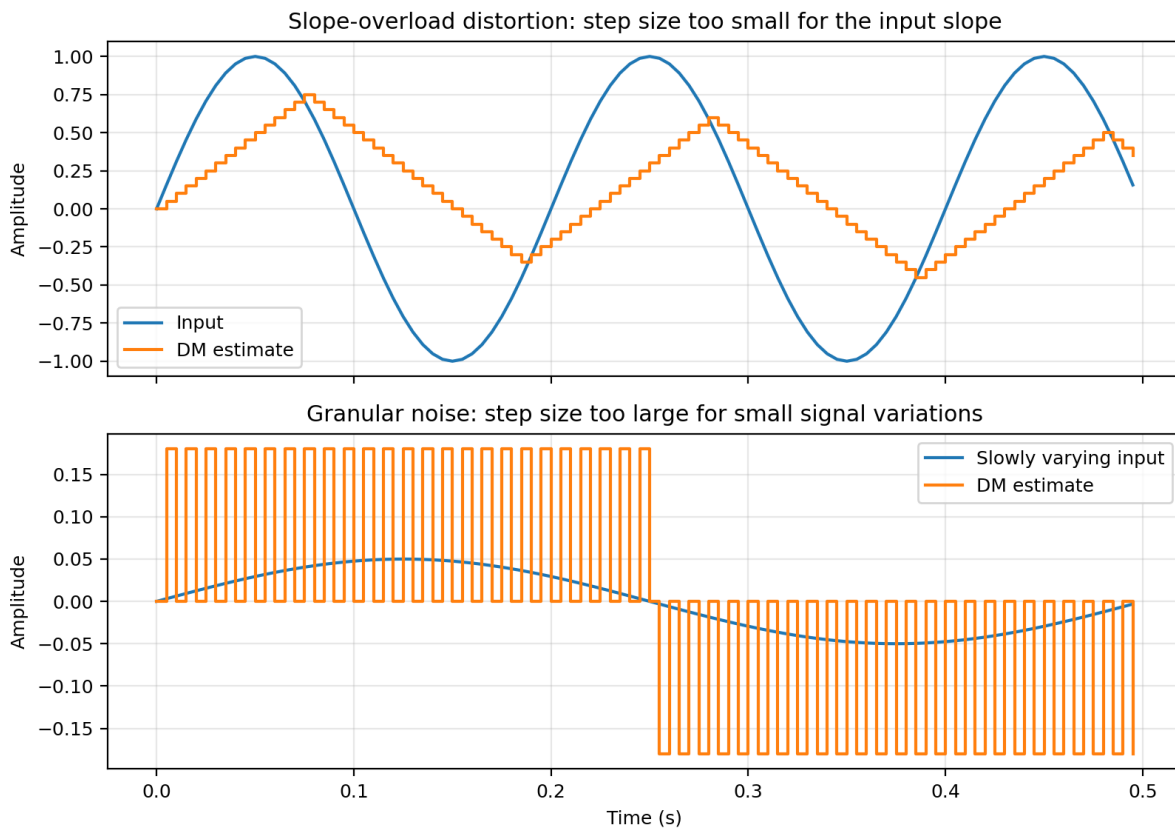


Fig. 5.3. Characteristic time-domain behavior of slope-overload distortion and granular noise.

2.6. Influence of Sampling Frequency

Increasing the sampling frequency reduces the sampling interval and increases the maximum available staircase slope Δf_s for a fixed step size. It also provides more decisions per message period. However, because one bit is transmitted per sample, a higher sampling frequency directly increases the DM bit rate. The selected sampling frequency and step size must therefore be considered together.

2.7. Adaptive Delta Modulation

Adaptive delta modulation changes the step size according to the recent behavior of the one-bit decision sequence. When several consecutive bits have the same value, the input is probably changing rapidly in one direction; the step size is increased to improve slope tracking. When the bit sequence alternates frequently, the staircase is probably hunting around a slowly varying input; the step size is reduced to limit granular noise.

The lecture notes describe ADM as a method that increases the step size for rapidly varying segments and reduces it for slowly varying segments. For the MATLAB implementation in this experiment, the following simple multiplicative rule is adopted:

$$\text{If } b[n] = b[n-1]: \Delta[n] = \min(\alpha_{up} \Delta[n-1], \Delta_{max})$$

$$\text{If } b[n] \neq b[n-1]: \Delta[n] = \max(\alpha_{down} \Delta[n-1], \Delta_{min})$$

Typical values used in this experiment are $\alpha_{up} > 1$ and $0 < \alpha_{down} < 1$. The lower and upper limits prevent the step size from becoming impractically small or excessively large.

Adaptive Delta Modulation with Step-Size Control

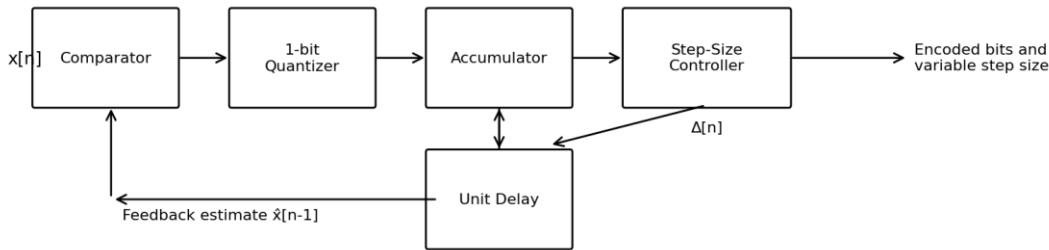


Fig. 5.4. Conceptual block diagram of adaptive delta modulation with a step-size controller.

2.8. Performance Measures

The numerical quality of a reconstructed waveform can be evaluated using the mean-square error (MSE) and reconstruction signal-to-noise ratio (SNR):

$$MSE = (1/N) \sum [x[n] - \hat{x}[n]]^2$$

$$SNR_{rec} = 10 \log_{10} \{ \sum x^2[n] / \sum [x[n] - \hat{x}[n]]^2 \} \text{ dB}$$

A lower MSE and a higher reconstruction SNR indicate improved waveform tracking. These measures do not replace visual inspection because slope-overload and granular errors may have different time-domain characteristics even when their global MSE values are similar.

3. Materials Used

- Computer
- MATLAB software
- MATLAB Script Editor
- Digital Communications lecture notes
- Laboratory worksheet
- Calculator

The principal MATLAB functions and operations used in this experiment include:

- sin, pi, zeros, ones, sign, diff, mean, sum, log10
- plot, stairs, stem, subplot, tiledlayout, grid, legend
- logical comparisons, for loops, indexing, min, max, movmean

4. Preliminary (PreLab)

Complete the following calculations and predictions before attending the laboratory.

4.1. Fixed-Step Delta-Modulation Calculations

Assume the sinusoidal message signal and sampling parameters

$$x(t) = 1.0 \sin(2\pi \cdot 5t), \quad f_s = 200 \text{ Hz}$$

For the three candidate step sizes $\Delta = 0.05, 0.18, \text{ and } 0.40$:

Calculate the sampling period T_s .

Calculate the maximum input slope $2\pi f_m A_m$.

Calculate the available staircase slope Δf_s for each step size.

Determine whether the theoretical slope-overload condition is satisfied.

Predict which step size will cause the strongest granular noise.

Calculate the DM bit rate.

Table 5.1. PreLab fixed-step delta-modulation calculations

Step size Δ	Sampling period T_s	Maximum input slope	Staircase slope Δf_s	Slope condition	Predicted dominant error
0.05					
0.18					
0.40					

4.2. Sampling-Frequency Calculations

For $\Delta = 0.18$ and sampling frequencies $f_s = 80, 200, \text{ and } 500 \text{ Hz}$, calculate the sampling period, available staircase slope, and DM bit rate. Predict how the reconstruction quality will change.

Table 5.2. PreLab sampling-frequency calculations

Sampling frequency	Sampling period	Δf_s	DM bit rate	Predicted tracking quality
80 Hz				
200 Hz				
500 Hz				

4.3. Adaptive Step-Size Prediction

For the adaptation parameters

$$\Delta_0 = 0.05, \alpha_{up} = 1.5, \alpha_{down} = 0.75, \Delta_{min} = 0.01, \Delta_{max} = 0.50$$

determine the step-size sequence for the decision bits shown below. Apply the adaptation after each decision, beginning with $\Delta[1] = \Delta_0$.

Decision sequence: 1 1 1 0 1 0 0 0

Table 5.3. PreLab adaptive-step calculation

Sample index n	Decision bit b[n]	Previous step size	Adaptation condition	New step size
1				
2				
3				
4				
5				
6				
7				
8				

4.4. Bit-Rate Comparison

Assume that an 8-bit PCM system and a one-bit DM system use the same sampling frequency $f_s = 8 \text{ kHz}$. Calculate the transmitted bit rate of each system and the ratio between them.

Table 5.4. PCM and DM bit-rate comparison

System	Bits per sample	Sampling frequency	Bit rate	Relative rate
8-bit PCM	8	8 kHz		

DM / ADM

1

8 kHz

5. Procedure

Part 1: Generate the Message Signal

Open MATLAB and create a new script file.

Define $A_m = 1$, $f_m = 5$ Hz, $f_s = 200$ Hz, and an observation interval of 1 second.

Generate the sampled sinusoidal message $x[n] = A_m \sin(2\pi f_m n T_s)$.

Plot the sampled message using both plot and stem.

Calculate the maximum numerical slope using $\text{diff}(x)/T_s$ and compare it with $2\pi f_m A_m$.

```
Am = 1; fm = 5; fs = 200;
Ts = 1/fs;
t = 0:Ts:1-Ts;
x = Am*sin(2*pi*fm*t);
maxSlopeNumerical = max(abs(diff(x)/Ts));
maxSlopeTheoretical = 2*pi*fm*Am;
figure;
subplot(2,1,1); plot(t,x,'LineWidth',1.1); grid on;
xlabel('Time (s)'); ylabel('Amplitude'); title('Message signal');
subplot(2,1,2); stem(t,x,'.'); grid on;
xlabel('Time (s)'); ylabel('Sample value'); title('Sampled message');
```

Part 2: Implement a Fixed-Step Delta Modulator

Set the initial staircase value to zero and choose $\Delta = 0.18$.

At every sample, compare the input with the previous staircase estimate.

Generate binary 1 for an upward step and binary 0 for a downward step.

Accumulate the signed steps to obtain $\hat{x}[n]$.

Plot $x[n]$, $\hat{x}[n]$, the decision bits, and the reconstruction error.

Calculate MSE and reconstruction SNR.

```
Delta = 0.18;
N = length(x);
bits = zeros(1,N);
xhat = zeros(1,N);
for n = 2:N
    if x(n) >= xhat(n-1)
        bits(n) = 1;
        xhat(n) = xhat(n-1) + Delta;
    else
        bits(n) = 0;
        xhat(n) = xhat(n-1) - Delta;
    end
end
```

```
err = x - xhat;  
mseDM = mean(err.^2);  
snrDM = 10*log10(sum(x.^2)/sum(err.^2));
```

Part 3: Investigate the Effect of Step Size

Repeat the fixed-step algorithm for $\Delta = 0.05, 0.18$, and 0.40 .

For each case, calculate Δf_s and compare it with the maximum input slope.

Plot the input and staircase approximation in separate subplots.

Calculate MSE, reconstruction SNR, and the proportion of logic-1 decisions.

Identify the case dominated by slope overload and the case dominated by granular noise.

```
DeltaList = [0.05 0.18 0.40];
for k = 1:length(DeltaList)
    Delta = DeltaList(k);
    bits = zeros(1,N); xhat = zeros(1,N);
    for n = 2:N
        bits(n) = x(n) >= xhat(n-1);
        xhat(n) = xhat(n-1) + (2*bits(n)-1)*Delta;
    end
    MSE(k) = mean((x-xhat).^2);
    SNRrec(k) = 10*log10(sum(x.^2)/sum((x-xhat).^2));
    oneRatio(k) = mean(bits);
end
```

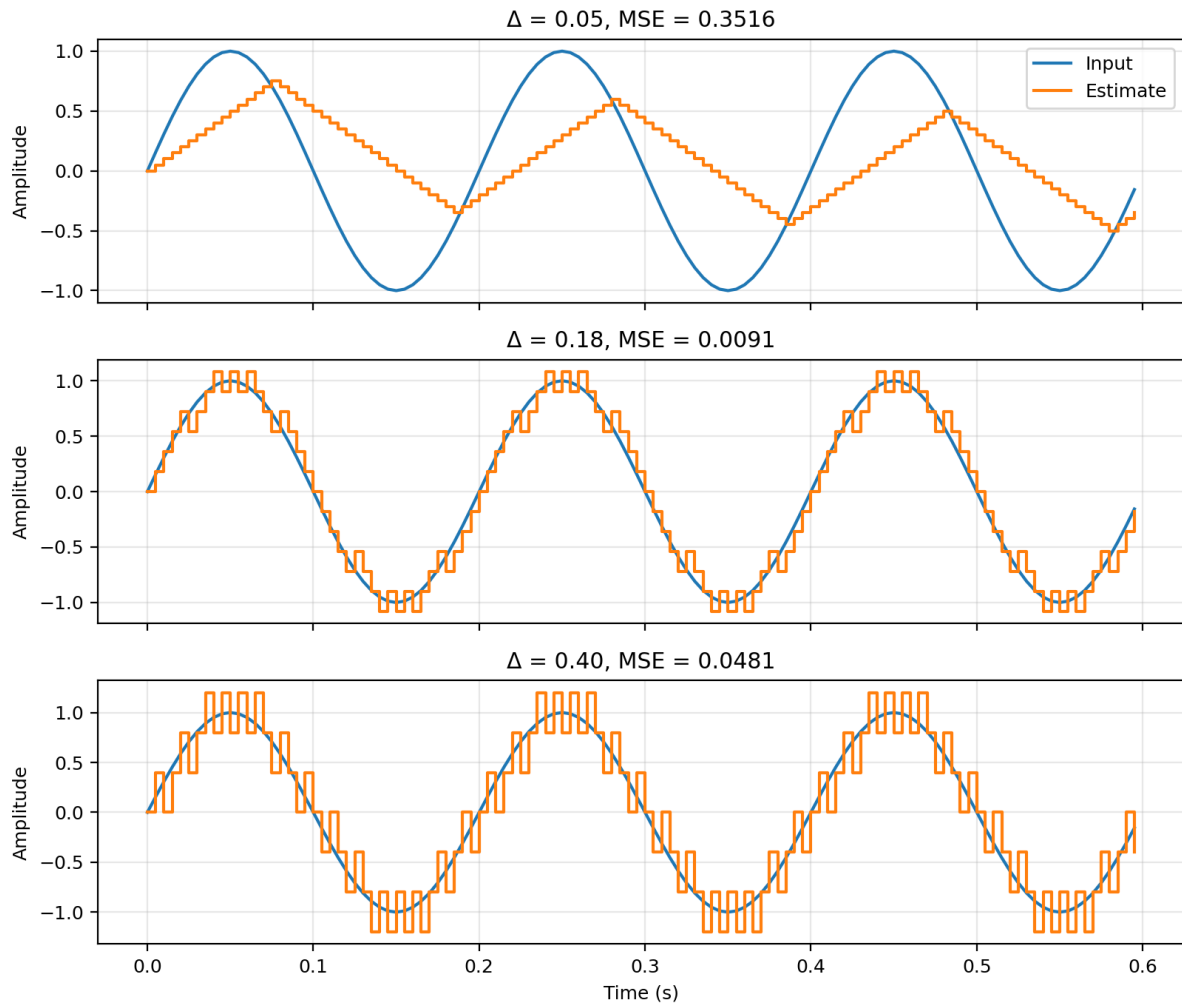


Fig. 5.5. Representative fixed-step DM results for small, intermediate, and large step sizes.

Part 4: Investigate the Effect of Sampling Frequency

Keep $\Delta = 0.18$ and repeat the simulation for $f_s = 80, 200$, and 500 Hz.

Regenerate the time vector and message signal for each sampling frequency.

Calculate the staircase slope Δf_s , DM bit rate, MSE, and reconstruction SNR.

Compare the number of decisions per message period.

Explain why increasing f_s improves tracking for the same step size but increases bit rate.

```
fsList = [80 200 500];
Delta = 0.18;
for k = 1:length(fsList)
    fs = fsList(k); Ts = 1/fs;
    t = 0:Ts:1-Ts;
    x = Am*sin(2*pi*fm*t);
    N = length(x); bits = zeros(1,N); xhat = zeros(1,N);
    for n = 2:N
        bits(n) = x(n) >= xhat(n-1);
        xhat(n) = xhat(n-1) + (2*bits(n)-1)*Delta;
    end
    MSEfs(k) = mean((x-xhat).^2);
    SNRfs(k) = 10*log10(sum(x.^2)/sum((x-xhat).^2));
end
```

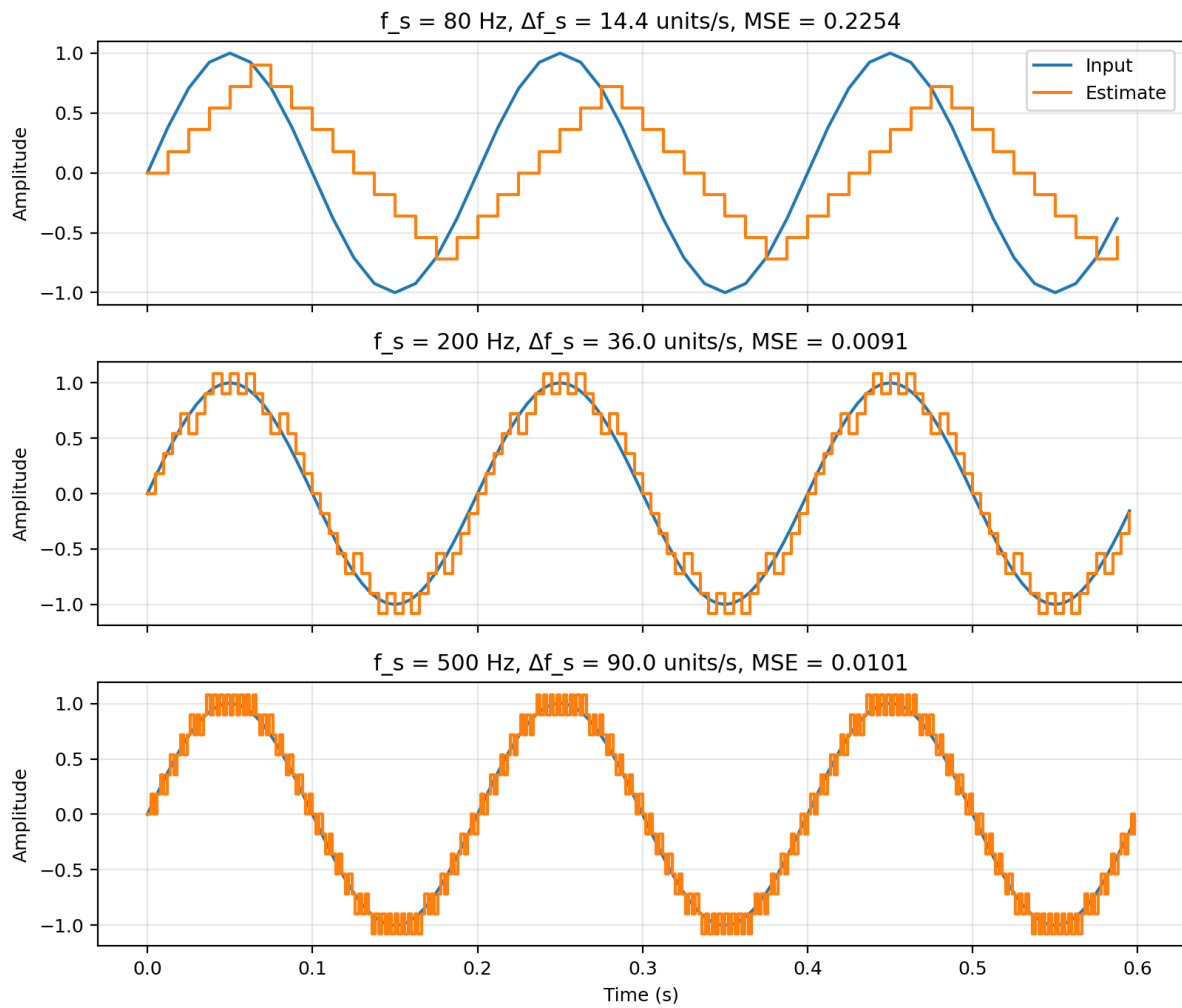


Fig. 5.6. Influence of sampling frequency on fixed-step delta-modulation tracking.

Part 5: Demonstrate Slope-Overload Distortion

Use $x(t) = \sin(2\pi \cdot 5t)$, $f_s = 200$ Hz, and $\Delta = 0.05$.

Calculate and report the maximum input slope and maximum staircase slope.

Locate the time intervals where the input has its greatest positive and negative slopes.

Observe long runs of identical decision bits in these intervals.

Increase Δ gradually until the theoretical slope condition is satisfied and record the minimum practical value.

Part 6: Demonstrate Granular Noise

Generate a low-amplitude, slowly varying signal $x(t) = 0.05 \sin(2\pi \cdot 2t)$.

Use $f_s = 200$ Hz and $\Delta = 0.18$.

Plot the input and staircase waveform.

Count the number of bit transitions in the generated decision sequence.

Reduce Δ and observe the change in granular oscillation and MSE.

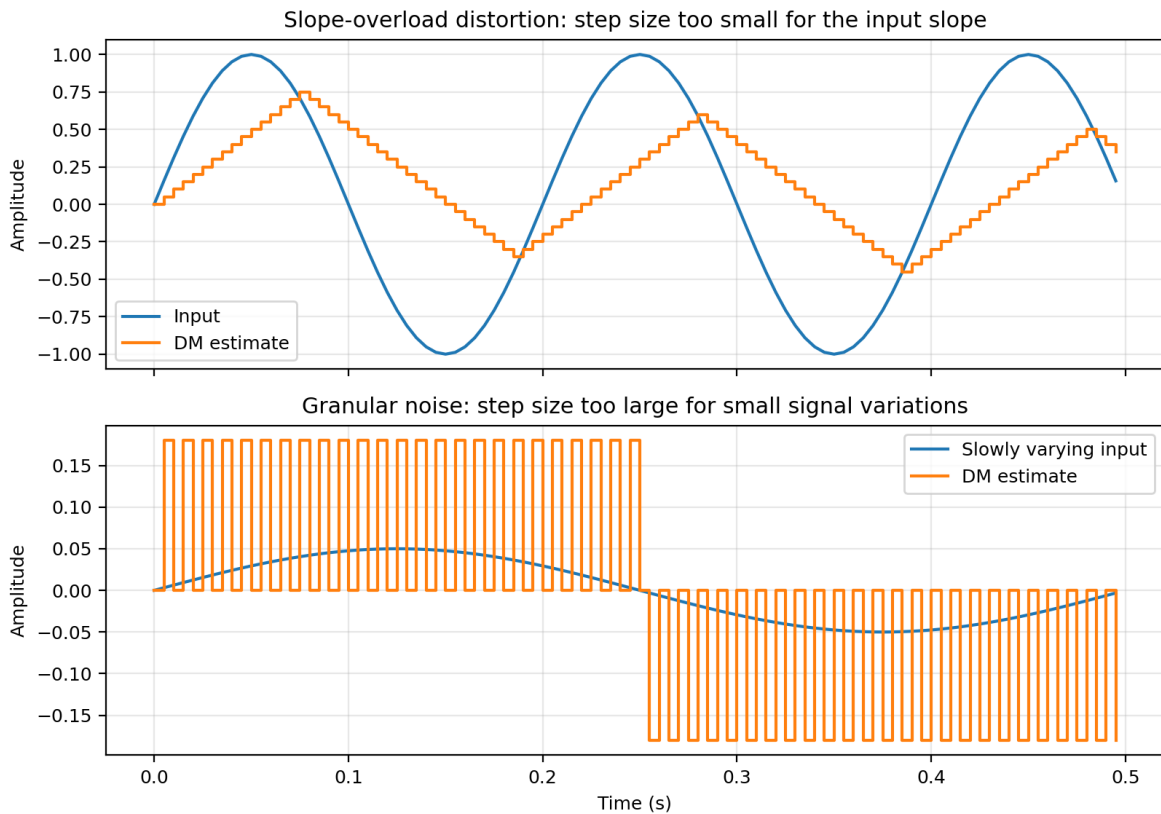


Fig. 5.7. Reference behavior to be reproduced in the slope-overload and granular-noise tests.

Part 7: Implement Adaptive Delta Modulation

Use the following piecewise message to produce low-amplitude, high-slope, and moderate-amplitude intervals within the same observation record:

$$0 \leq t < 0.3 \text{ s: } x(t) = 0.2 \sin(2\pi \cdot 5t)$$

$$0.3 \leq t < 0.7 \text{ s: } x(t) = 0.9 \sin(2\pi \cdot 12t)$$

$$0.7 \leq t < 1 \text{ s: } x(t) = 0.3 \sin(2\pi \cdot 5t)$$

Set $f_s = 500 \text{ Hz}$, $\Delta_0 = 0.05$, $\Delta_{\min} = 0.01$, and $\Delta_{\max} = 0.50$.

Use $\alpha_{\text{up}} = 1.5$ for consecutive equal decisions and $\alpha_{\text{down}} = 0.75$ for alternating decisions.

Generate the adaptive decision sequence, staircase waveform, and step-size history.

Plot the message, ADM reconstruction, fixed-step DM reconstruction, and $\Delta[n]$.

Calculate the MSE and reconstruction SNR for both methods.

Check that the adaptive step size increases in the high-slope interval and decreases in the low-amplitude intervals.

```
Delta0 = 0.05; DeltaMin = 0.01; DeltaMax = 0.50;
alphaUp = 1.5; alphaDown = 0.75;
Delta = Delta0*ones(1,N);
bitsADM = zeros(1,N); xhatADM = zeros(1,N);
for n = 2:N
    bitsADM(n) = x(n) >= xhatADM(n-1);
    xhatADM(n) = xhatADM(n-1) + (2*bitsADM(n)-1)*Delta(n-1);
    if n >= 3 && bitsADM(n) == bitsADM(n-1)
        Delta(n) = min(alphaUp*Delta(n-1),DeltaMax);
    else
        Delta(n) = max(alphaDown*Delta(n-1),DeltaMin);
    end
end
```

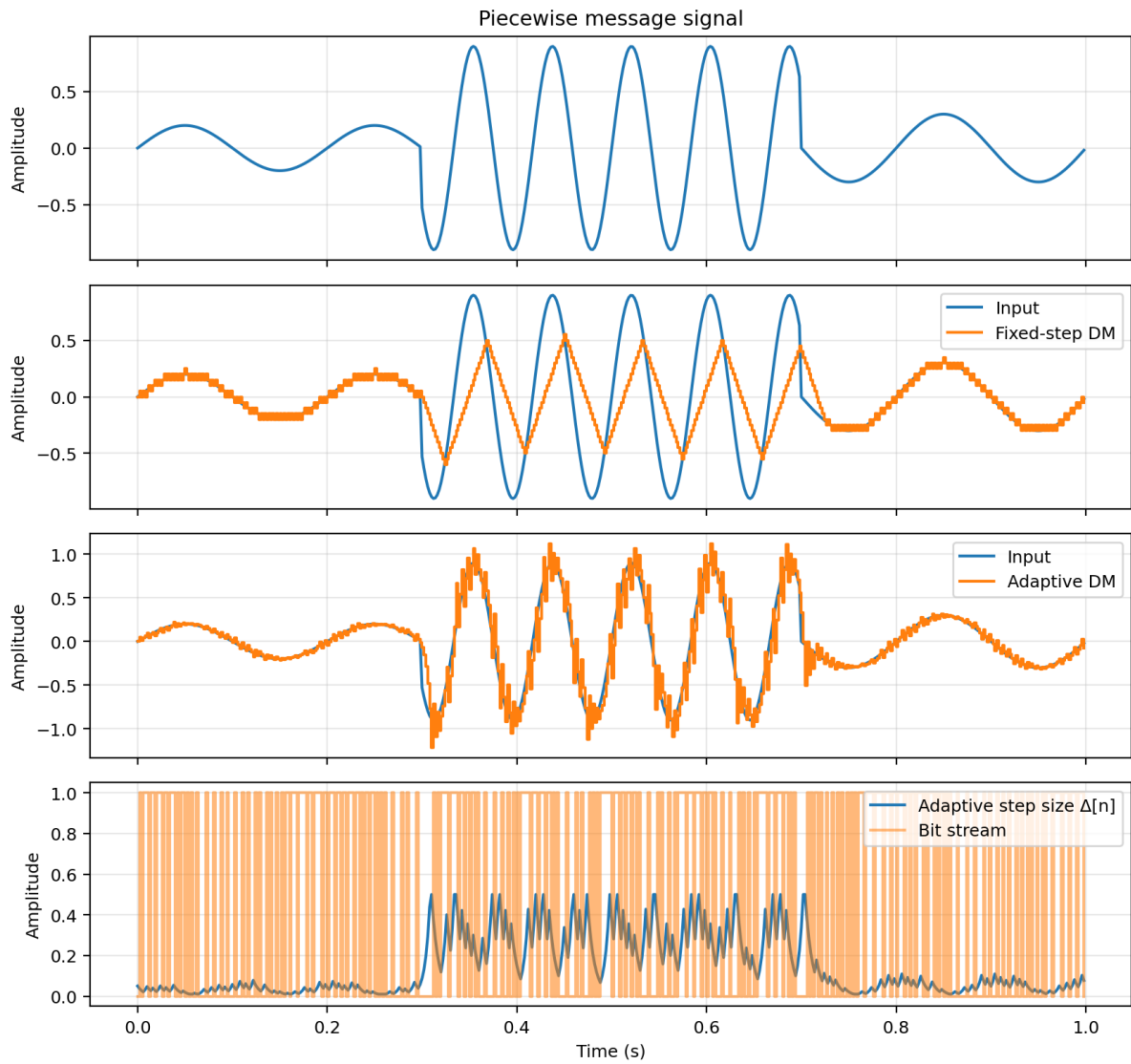


Fig. 5.8. Example comparison of fixed-step DM and ADM for a message with changing slope and amplitude.

Part 8: Reconstruction Filtering and Final Comparison

Apply a short moving-average filter to the fixed-step and adaptive staircase outputs using movmean.

Choose the averaging length carefully so that staircase ripple is reduced without excessive attenuation or delay.

Compare the unfiltered and filtered reconstructions.

Recalculate MSE and SNR after filtering.

Prepare a final comparison including step size behavior, dominant distortion, reconstruction quality, and bit rate.

```
L = 5; % moving-average length
xhatDM_f = movmean(xhatDM,L);
xhatADM_f = movmean(xhatADM,L);
mseDM_f = mean((x-xhatDM_f).^2);
mseADM_f = mean((x-xhatADM_f).^2);
snrDM_f = 10*log10(sum(x.^2)/sum((x-xhatDM_f).^2));
snrADM_f = 10*log10(sum(x.^2)/sum((x-xhatADM_f).^2));
```

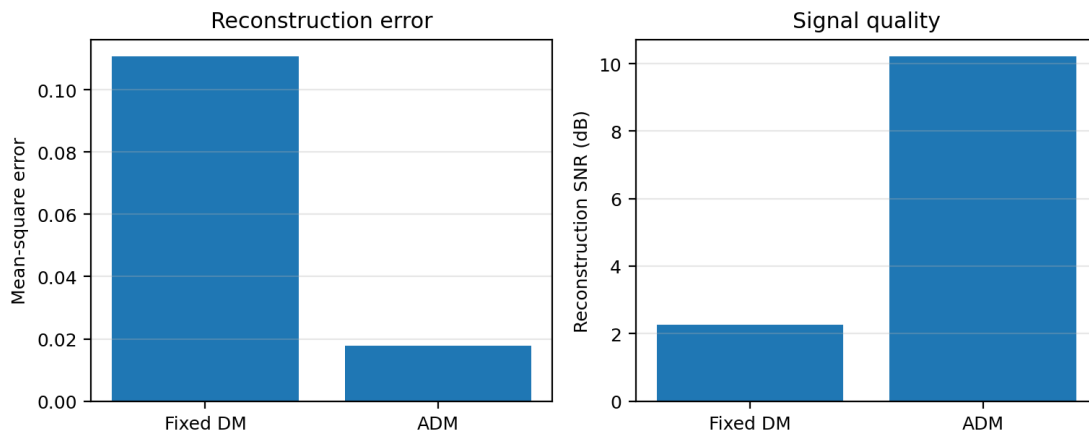


Fig. 5.9. Representative error and reconstruction-SNR comparison of fixed-step DM and ADM.

Save the completed MATLAB script, all figures, and the numerical results. Clearly label all axes and include units where applicable.

6. Evaluation of Results..... 17

Complete all tables using the values obtained during the experiment. Attach the requested plots and answer the questions following the tables.

Table 5.5. Fixed-step delta-modulation performance

Δ	Δf_s	Slope condition	MSE	SNR_rec (dB)	Logic-1 ratio	Dominant error
0.05						
0.18						
0.40						

Table 5.6. Sampling-frequency comparison

f_s	T_s	Samples/message period	DM bit rate	Δf_s	MSE	SNR_rec (dB)
80 Hz						
200 Hz						
500 Hz						

Table 5.7. Slope-overload and granular-noise observations

Test	Δ_m	f_m	f_s	Δ	Maximum input slope	Maximum staircase slope	Observation
Slope overload							
Granular noise							

Table 5.8. Fixed-step DM and ADM comparison

Method	Initial / fixed Δ	$\Delta_{\min}$	$\Delta_{\max}$	MSE	SNR_rec (dB)	Filtered MSE	Observation
Fixed-step DM		-	-				
Adaptive DM							

Table 5.9. Bit-rate comparison

System	Bits/sample	Sampling frequency	Calculated bit rate	Comments
8-bit PCM				
Delta modulation				
Adaptive delta modulation				

Answer the following questions:

What information is represented by binary 1 and binary 0 in a delta-modulation system?

Why does conventional DM require only one transmitted bit per sample?

Derive the maximum slope of the staircase approximation as Δf_s .

For a sinusoidal input, which parameters increase the risk of slope-overload distortion?

Why do repeated identical bits indicate that the staircase is attempting to follow a large slope?

Why does rapid alternation between 1 and 0 commonly indicate granular noise?

How did increasing the step size affect slope overload, granular noise, MSE, and reconstruction SNR?

How did increasing the sampling frequency affect tracking quality and transmitted bit rate?

Compare the calculated slope-overload condition with the waveform observed in MATLAB.

Why can a fixed step size not provide optimal performance for both low-amplitude and high-slope signal segments?

Explain how the selected ADM rule responds to repeated and alternating decisions.

In which portions of the piecewise message did the adaptive step size reach its largest values?

Did ADM always produce a lower MSE than fixed-step DM? Discuss the result based on the selected adaptation parameters.

How did moving-average filtering affect staircase ripple, delay, and signal amplitude?

Compare the bit rates of 8-bit PCM and DM when both use the same sampling frequency.

What practical disadvantages may result from using a very high sampling frequency in delta modulation?

Suggest one modification to the adaptive rule that could improve the observed reconstruction.

Summarize the trade-off among step size, sampling frequency, tracking capability, granular noise, and bit rate.

Submit the MATLAB .m file, all time-domain plots, completed result tables, and a concise discussion comparing conventional and adaptive delta modulation.

Experiment 6: ASK and FSK Modulation and Detection

1. Objective of the Experiment

The objective of this experiment is to investigate amplitude-shift keying (ASK) and binary frequency-shift keying (BFSK), two fundamental digital carrier-modulation techniques. Students will generate binary data waveforms, map the data onto sinusoidal carriers, examine the modulated signals in the time and frequency domains, and recover the transmitted bits by coherent and noncoherent detection methods.

The ASK section uses on-off keying, in which the carrier is transmitted for a binary 1 and suppressed for a binary 0. The FSK section represents binary 1 and binary 0 with two different carrier frequencies called the mark and space frequencies. Noise and carrier-phase conditions are varied to demonstrate the practical differences between amplitude-based and frequency-based signaling.

At the end of this experiment, students are expected to:

- Explain how binary information controls the amplitude or frequency of a sinusoidal carrier.
- Generate ASK/OOK and BFSK waveforms from a specified binary sequence in MATLAB.
- Identify the carrier, mark, space, and sideband components in FFT-based spectra.
- Calculate bit duration, baud rate, frequency deviation, and approximate transmission bandwidth.
- Implement coherent ASK detection using a product detector, integration, and threshold decision.
- Implement coherent and noncoherent BFSK detection using correlation and energy metrics.
- Examine the effects of additive noise and carrier-phase mismatch on the recovered bits.
- Compare ASK and BFSK with respect to envelope behavior, constant amplitude, bandwidth, and receiver requirements.

2. Theoretical Background

2.1. Digital Carrier Modulation

A digital modulator maps a sequence of binary symbols into one of a finite set of signal waveforms that can be transmitted through a physical channel. During each bit interval T_b , the selected waveform represents the current binary symbol. For binary modulation, one signaling element carries one information bit; therefore, the symbol rate and the bit rate are equal.

$$T_b = 1/R_b \quad \text{and} \quad R_s = R_b \text{ baud}$$

ASK changes the carrier amplitude while keeping its nominal frequency and phase unchanged. FSK keeps the amplitude approximately constant but selects one of two frequencies. These characteristics lead to different spectra and receiver structures.

2.2. Amplitude-Shift Keying and On-Off Keying

In binary ASK, the amplitude of the carrier is selected according to the input bit. The most common binary form is on-off keying (OOK). Using the lecture-note convention, a logic 1 passes the carrier and a logic 0 suppresses it. Over one bit interval, the transmitted signals may be written as

$$\begin{aligned} s_1(t) &= A_c \cos(2\pi f_c t), \quad 0 \leq t < T_b \\ s_0(t) &= 0, \quad 0 \leq t < T_b \end{aligned}$$

For a unipolar data waveform $b(t)$ that takes the values 0 and 1, the ASK signal is

$$s_{ASK}(t) = A_c b(t) \cos(2\pi f_c t)$$

A binary ASK signaling element lasts for one bit interval, so the baud rate is equal to the input bit rate. The lecture notes use the following approximate minimum bandwidth relationship for binary ASK:

$$B_{ASK,min} \approx R_b$$

Because the data sequence gates the carrier on and off, abrupt bit transitions produce spectral spreading around the carrier. The exact measured spectrum also depends on the data pattern, observation duration, and pulse shape used for the bit waveform.

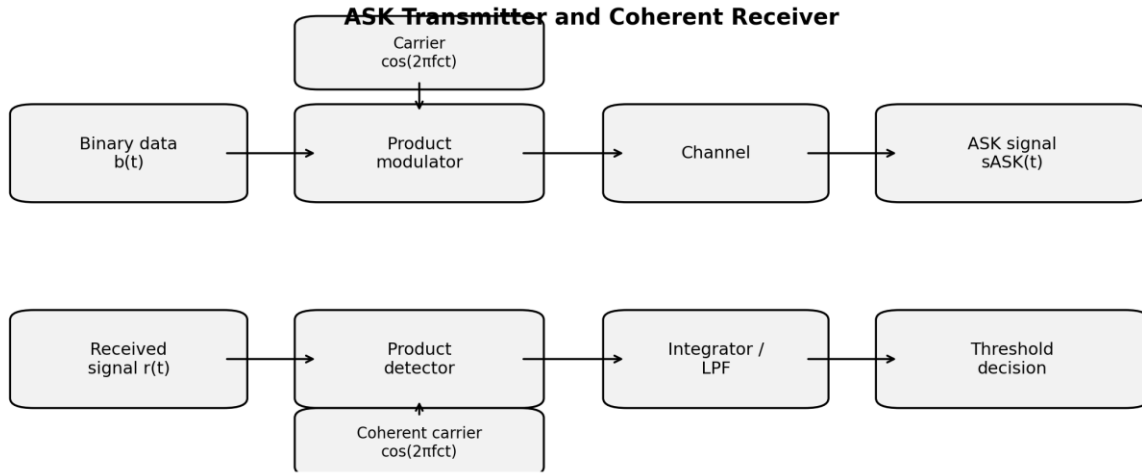


Fig. 6.1. ASK transmitter and coherent receiver structures.

2.3. Coherent Detection of ASK

A coherent ASK receiver multiplies the received waveform by a locally generated carrier having the same frequency and phase as the transmitter carrier. The product is integrated or low-pass filtered over each bit interval. For an ideal carrier reference, the decision statistic is approximately A_c for a transmitted 1 and zero for a transmitted 0 after suitable normalization.

$$z_k = (2/T_b) \int_{[(k-1)T_b \text{ to } kT_b]} r(t) \cos(2\pi f_c t) dt$$

The recovered bit is determined by comparing z_k with a threshold γ . For the normalized OOK waveform used in this experiment, $\gamma = A_c/2$ is a convenient threshold:

$$\hat{b}_k = 1, z_k \geq A_c/2; \quad \hat{b}_k = 0, z_k < A_c/2$$

Carrier-frequency or phase mismatch reduces the useful correlator output. Consequently, coherent ASK detection requires carrier synchronization.

2.4. Binary Frequency-Shift Keying

In BFSK, the carrier frequency changes between two discrete values. The frequency assigned to binary 1 is called the mark frequency f_M , and the frequency assigned to binary 0 is called the space frequency f_S . The two transmitted waveforms can be written as

$$s_1(t) = A_c \cos(2\pi f_M t), \quad 0 \leq t < T_b$$

$$s_0(t) = A_c \cos(2\pi f_S t), \quad 0 \leq t < T_b$$

The center frequency and peak frequency deviation are

$$f_c = (f_M + f_S)/2$$

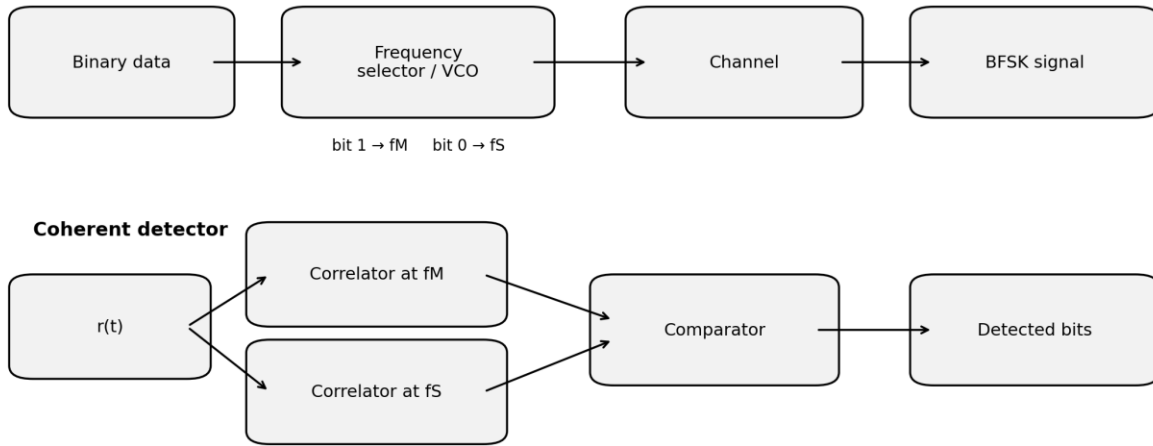
$$\Delta f = |f_M - f_S|/2$$

The separation between the two transmitted frequencies is $2\Delta f$. According to the relationship used in the lecture notes, the approximate minimum BFSK bandwidth is

$$B_{FSK,min} \approx 2(\Delta f + R_b)$$

The BFSK signal has an approximately constant envelope. Therefore, unlike OOK, the signal does not disappear during zero bits. The information is carried by frequency selection rather than by carrier presence.

Binary FSK Transmitter and Receiver Structures



Noncoherent implementation: two band-pass filters / quadrature energy detectors followed by a comparator.

Fig. 6.2. BFSK transmitter and coherent/noncoherent receiver structures.

2.5. Coherent and Noncoherent Detection of BFSK

A coherent BFSK receiver forms two correlations: one with a reference at the mark frequency and one with a reference at the space frequency. The receiver decides in favor of the frequency producing the larger correlation output. The carrier references must be synchronized with the received signal.

$$z_{M,k} = (2/T_b) \int r(t) \cos(2\pi f_M t) dt$$

$$z_{S,k} = (2/T_b) \int r(t) \cos(2\pi f_S t) dt$$

$$\hat{b}_k = 1 \text{ if } z_{M,k} > z_{S,k}; \text{ otherwise } \hat{b}_k = 0$$

In the noncoherent structure described in the lecture notes, the received signal is applied to two band-pass filters centered at f_M and f_S . Envelope detectors estimate the power in the two passbands, and a comparator selects the larger output. A MATLAB equivalent can be implemented with quadrature correlations. The phase-independent energy metrics are

$$E_{M,k} = (\int r(t) \cos(2\pi f_M t) dt)^2 + (\int r(t) \sin(2\pi f_M t) dt)^2$$

$$E_{S,k} = (\int r(t) \cos(2\pi f_s t) dt)^2 + (\int r(t) \sin(2\pi f_s t) dt)^2$$

$$\hat{b}_k = 1 \text{ if } E_{M,k} > E_{S,k}; \text{ otherwise } \hat{b}_k = 0$$

Because the squared sum removes dependence on an unknown carrier phase, noncoherent detection avoids explicit phase recovery, although its noise performance is generally poorer than that of an ideal coherent receiver.

2.6. Additive White Gaussian Noise

To examine receiver decisions under noise, the channel output is modeled as

$$r(t) = s(t) + n(t)$$

where $n(t)$ is additive white Gaussian noise. In MATLAB, a real Gaussian sequence can be scaled to obtain a desired signal-to-noise ratio. This experiment uses a fixed random seed so that all students obtain reproducible results. Detailed theoretical BER curves are reserved for a later matched-filter and error-probability experiment.

3. Materials Used

- Computer
- MATLAB software
- MATLAB Script Editor
- Digital Communications lecture notes
- Laboratory worksheet
- Calculator

The following MATLAB functions and operations may be used:

- `repelem` or `kron`, `reshape`, and logical indexing
- `cos`, `sin`, `randn`, `mean`, `sum`, and `sqrt`
- `fft`, `fftshift`, `fftfreq`-equivalent axis construction, and `abs`
- `plot`, `stairs`, `stem`, `subplot`, `grid`, `xlabel`, `ylabel`, `title`, and `legend`

4. Preliminary (PreLab)

Complete the following tasks before attending the laboratory. Use the parameter set below unless another value is explicitly stated.

Table 6.1. Parameters used in the preliminary study.

Parameter	Symbol	Value
Binary sequence	b	1 0 1 1 0 0 1 0
Bit rate	R_b	100 bit/s
Sampling frequency	F_s	20 kHz
ASK carrier frequency	f_c	1 kHz
Mark frequency	f_M	1.5 kHz
Space frequency	f_S	1.0 kHz
Carrier amplitude	A_c	1 V

4.1. ASK Calculations

1. Calculate the bit duration T_b and the number of MATLAB samples per bit N_s .
2. Determine the total duration of the eight-bit sequence.

3. Write the ASK waveform for binary 1 and binary 0.
4. Calculate the approximate minimum ASK bandwidth using the lecture-note relationship.
5. Determine the ideal normalized correlator outputs for bits 1 and 0 and state the decision threshold.

4.2. BFSK Calculations

1. Calculate the center frequency f_c and peak deviation Δf .
2. Calculate the separation between the mark and space frequencies.
3. Calculate the approximate minimum BFSK bandwidth.
4. Determine the baud rate.
5. Calculate the number of mark-frequency and space-frequency cycles contained in one bit interval.
6. Sketch the expected time-domain waveform for the specified bit sequence.

Table 6.2. Preliminary ASK and BFSK calculations.

Quantity	Theoretical value	Unit
Bit duration T_b		s
Samples per bit N_s		samples
ASK minimum bandwidth		Hz
BFSK center frequency		Hz
BFSK deviation Δf		Hz
BFSK minimum bandwidth		Hz
Baud rate		baud
Mark cycles per bit		cycles
Space cycles per bit		cycles

4.3. Receiver Truth Tables

Table 6.3. Ideal receiver decisions.

Modulation	Transmitted bit	Expected dominant statistic	Receiver decision
ASK	1	z_k above threshold	
ASK	0	z_k below threshold	
BFSK	1	mark metric > space metric	
BFSK	0	space metric > mark metric	

5. Procedure

Part 1: Define the Binary Data and Simulation Parameters

1. Open MATLAB and create a new script file.
2. Define the binary sequence, bit rate, bit duration, sampling frequency, and samples per bit.
3. Generate a unipolar NRZ data waveform by repeating each bit N_s times.
4. Construct the time vector and verify that the bit boundaries occur at integer multiples of T_b .
5. Plot the bit waveform using stairs or plot.

```
clear; close all; clc;
rng(18); % Reproducible noise
bits = [1 0 1 1 0 0 1 0];
```

```

Rb = 100; % bit/s
Tb = 1/Rb;
Fs = 20000; % samples/s
Ns = round(Fs/Rb); % samples per bit
b = repelem(bits,Ns);
t = (0:length(b)-1)/Fs;
figure;
stairs(t,b,'LineWidth',1.2); grid on;
xlabel('Time (s)'); ylabel('Bit value');
title('Unipolar NRZ data waveform'); ylim([-0.2 1.2]);

```

Part 2: Generate and Analyze the ASK/OOK Signal

1. Set the ASK carrier frequency to 1 kHz and the carrier amplitude to 1 V.
2. Generate the carrier waveform and multiply it by the unipolar bit waveform.
3. Plot the binary data, carrier, and ASK signal in the same figure.
4. Calculate a centered FFT and plot the two-sided ASK magnitude spectrum.
5. Identify the carrier region and explain why the spectrum spreads around the carrier.
6. Record the measured spectral extent using a consistent amplitude criterion.

```

Ac = 1;
fc = 1000;
carrier = cos(2*pi*fc*t);
sASK = Ac*b.*carrier;

figure;
subplot(3,1,1); stairs(t,b,'LineWidth',1.1); grid on;
ylabel('Bit'); ylim([-0.2 1.2]);
subplot(3,1,2); plot(t,carrier); grid on; ylabel('Carrier');
subplot(3,1,3); plot(t,sASK); grid on;
xlabel('Time (s)'); ylabel('ASK');

N = length(sASK);
SASK = fftshift(fft(sASK))/N;
f = (-N/2:N/2-1)*(Fs/N);
figure; plot(f,abs(SASK),'LineWidth',1.1); grid on;
xlim([-2500 2500]); xlabel('Frequency (Hz)');
ylabel('|S_{ASK}(f)|'); title('Two-sided ASK spectrum');

```

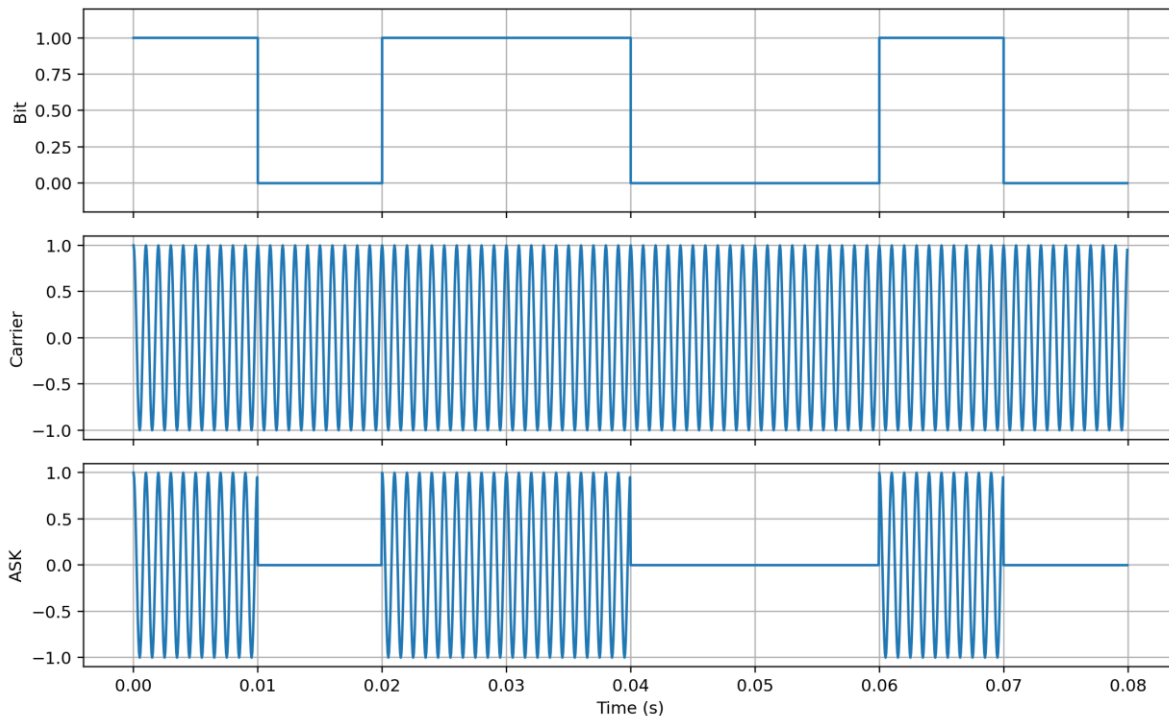


Fig. 6.3. Binary data, carrier, and ASK waveform for the specified bit sequence.

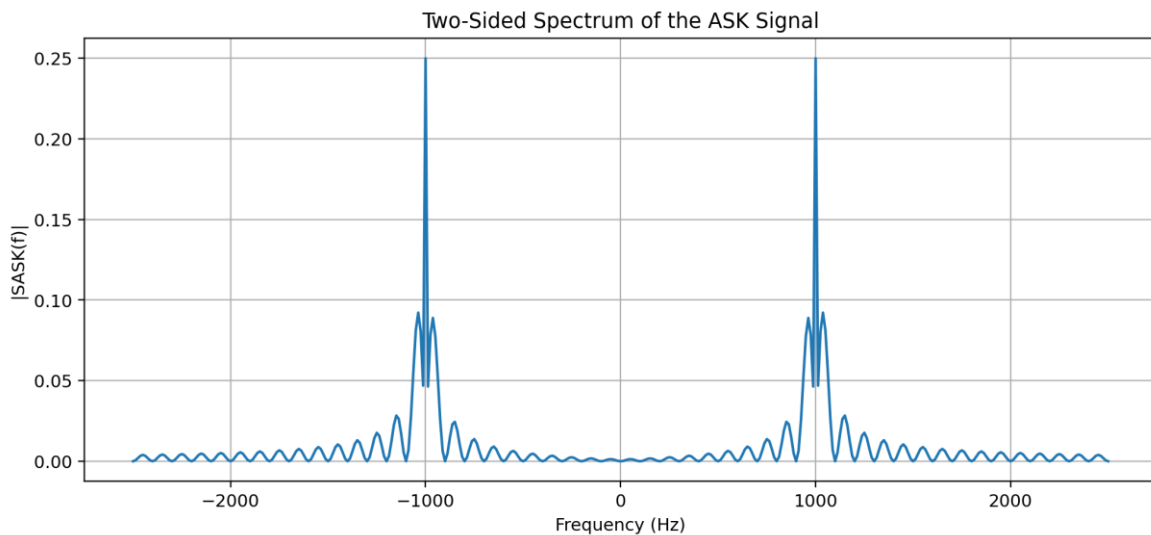


Fig. 6.4. Example two-sided spectrum of the ASK waveform.

Part 3: Coherent Detection of ASK

1. Add real Gaussian noise to the ASK signal at an SNR of 4 dB using the code below.
2. For each bit interval, multiply the received samples by the coherent carrier reference.
3. Sum and normalize the product to obtain the decision statistic z_k .
4. Use the threshold $\gamma = A_c/2$ and recover the bit sequence.
5. Calculate the number of bit errors and compare the clean and noisy decision statistics.
6. Repeat with a local carrier phase error of 60° and explain the change in the statistics.

```
SNRdB = 4;
Ps = mean(sASK.^2);
Pn = Ps/10^(SNRdB/10);
```

```

rASK = sASK + sqrt(Pn)*randn(size(sASK));

zASK = zeros(size(bits));
for k = 1:length(bits)
    idx = (k-1)*Ns + (1:Ns);
    ref = cos(2*pi*fc*t(idx));
    zASK(k) = 2*sum(rASK(idx).*ref)/Ns;
end
threshold = Ac/2;
bitsASK = zASK >= threshold;
errorsASK = sum(bitsASK ~= bits);

phi = deg2rad(60);
zPhase = zeros(size(bits));
for k = 1:length(bits)
    idx = (k-1)*Ns + (1:Ns);
    ref = cos(2*pi*fc*t(idx) + phi);
    zPhase(k) = 2*sum(rASK(idx).*ref)/Ns;
end

```

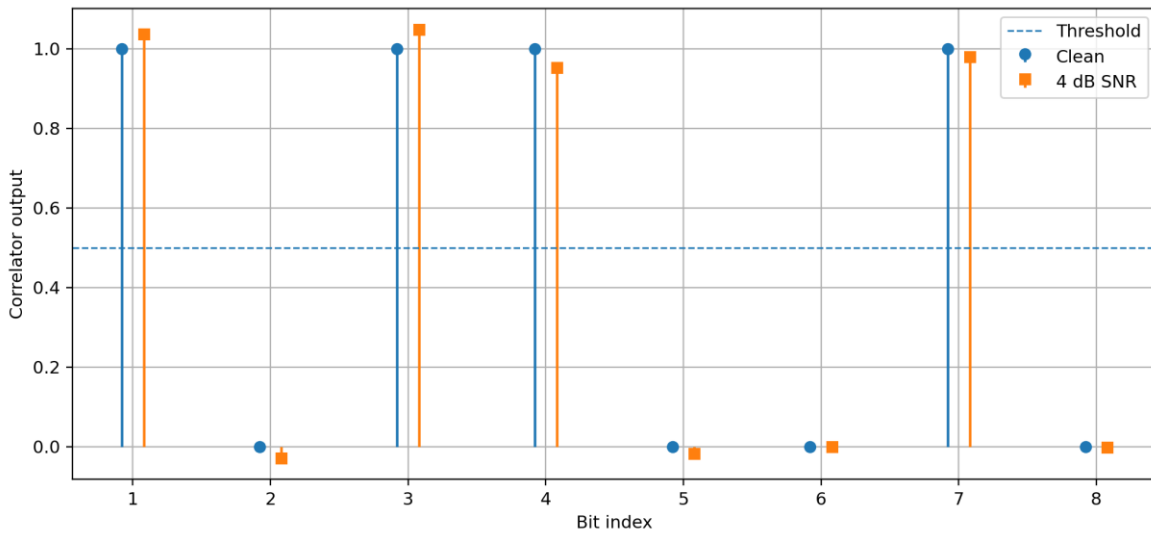


Fig. 6.5. ASK coherent-detector statistics and the decision threshold.

Part 4: Generate and Analyze the BFSK Signal

1. Set $f_M = 1.5$ kHz and $f_S = 1.0$ kHz.
2. Generate the BFSK signal bit by bit, selecting f_M for binary 1 and f_S for binary 0.
3. Plot the binary data, BFSK waveform, and selected instantaneous frequency.
4. Calculate the centered FFT and identify the mark and space spectral regions.
5. Calculate f_c , Δf , baud, and $B_{FSK,min}$ and compare them with the measured spectrum.

```

fM = 1500;
fS = 1000;
sFSK = zeros(size(t));
for k = 1:length(bits)
    idx = (k-1)*Ns + (1:Ns);
    if bits(k) == 1
        fk = fM;
    else
        fk = fS;
    end
    sFSK(idx) = Ac*cos(2*pi*fk*t(idx));
end

```



```

selectedFrequency = repelem(fS + bits*(fM-fS),Ns);
figure;
subplot(3,1,1); stairs(t,b); grid on; ylabel('Bit');
subplot(3,1,2); plot(t,sFSK); grid on; ylabel('BFSK');
subplot(3,1,3); stairs(t,selectedFrequency); grid on;
xlabel('Time (s)'); ylabel('Frequency (Hz)');

SFSK = fftshift(fft(sFSK))/length(sFSK);
figure; plot(f,abs(SFSK),'LineWidth',1.1); grid on;
xlim([-2500 2500]); xlabel('Frequency (Hz)');
ylabel('|S_{FSK}(f)|'); title('Two-sided BFSK spectrum');
    
```

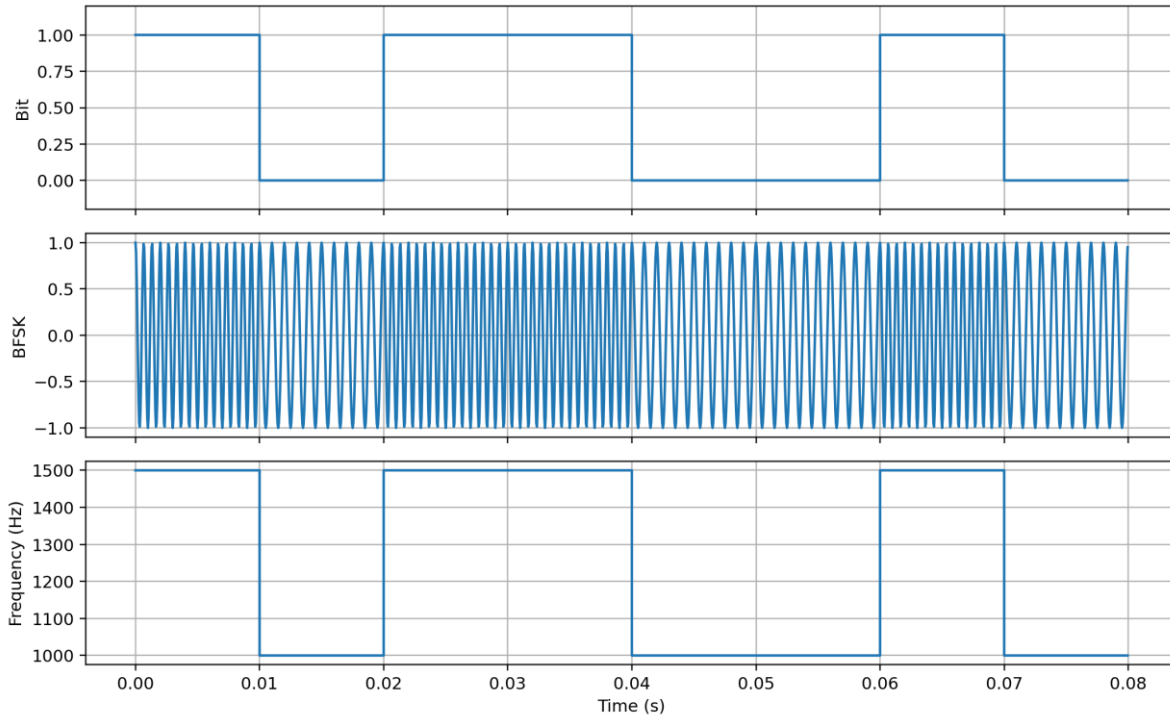


Fig. 6.6. BFSK waveform and the selected mark/space frequency.

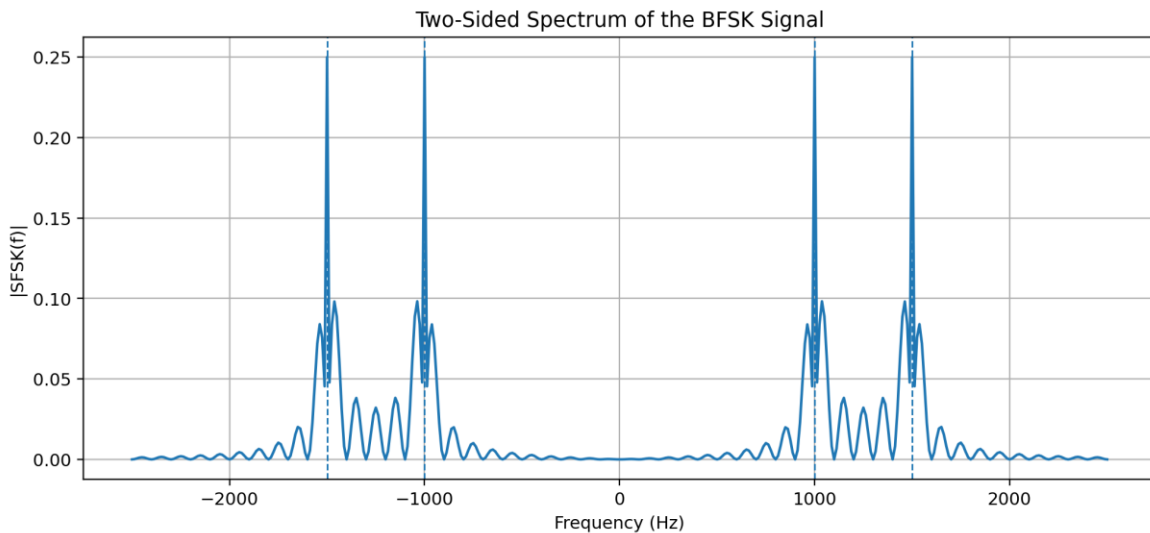


Fig. 6.7. Example two-sided BFSK spectrum showing mark and space regions.

Part 5: Coherent Detection of BFSK

1. Add Gaussian noise to the BFSK signal at 4 dB SNR.
2. For each bit interval, correlate the received samples with cosine references at f_M and f_S .
3. Decide binary 1 when the mark correlator output is larger than the space correlator output.
4. Compare the detected bit sequence with the transmitted bits and count the errors.
5. Repeat the coherent detector with a common 60° reference-phase error.

```
Ps = mean(sFSK.^2);
Pn = Ps/10^(SNRdB/10);
rFSK = sFSK + sqrt(Pn)*randn(size(sFSK));

zM = zeros(size(bits)); zS = zeros(size(bits));
for k = 1:length(bits)
    idx = (k-1)*Ns + (1:Ns);
    zM(k) = 2*sum(rFSK(idx).*cos(2*pi*fM*t(idx)))/Ns;
    zS(k) = 2*sum(rFSK(idx).*cos(2*pi*fS*t(idx)))/Ns;
end
bitsFSKcoh = zM > zS;
errorsFSKcoh = sum(bitsFSKcoh ~= bits);
```

Part 6: Noncoherent BFSK Detection

1. For each frequency, form in-phase and quadrature correlations using cosine and sine references.
2. Calculate the squared-sum energy metrics $E_{M,k}$ and $E_{S,k}$.
3. Decide according to the larger energy metric.
4. Compare coherent and noncoherent decisions at 4 dB SNR.
5. Apply an arbitrary phase shift to the transmitted BFSK waveform and verify that the quadrature energy detector remains phase independent.

```
EM = zeros(size(bits)); ES = zeros(size(bits));
for k = 1:length(bits)
    idx = (k-1)*Ns + (1:Ns);
    tt = t(idx);
    IM = sum(rFSK(idx).*cos(2*pi*fM*tt));
    QM = sum(rFSK(idx).*sin(2*pi*fM*tt));
    IS = sum(rFSK(idx).*cos(2*pi*fS*tt));
    QS = sum(rFSK(idx).*sin(2*pi*fS*tt));
    EM(k) = IM^2 + QM^2;
    ES(k) = IS^2 + QS^2;
end
bitsFSKnoncoh = EM > ES;
errorsFSKnoncoh = sum(bitsFSKnoncoh ~= bits);

figure;
subplot(2,1,1); plot(1:length(bits),zM,'o-', ...
    1:length(bits),zS,'s-'); grid on;
ylabel('Coherent metric'); legend('Mark','Space');
subplot(2,1,2); plot(1:length(bits),EM,'o-', ...
    1:length(bits),ES,'s-'); grid on;
xlabel('Bit index'); ylabel('Energy'); legend('Mark','Space');
```

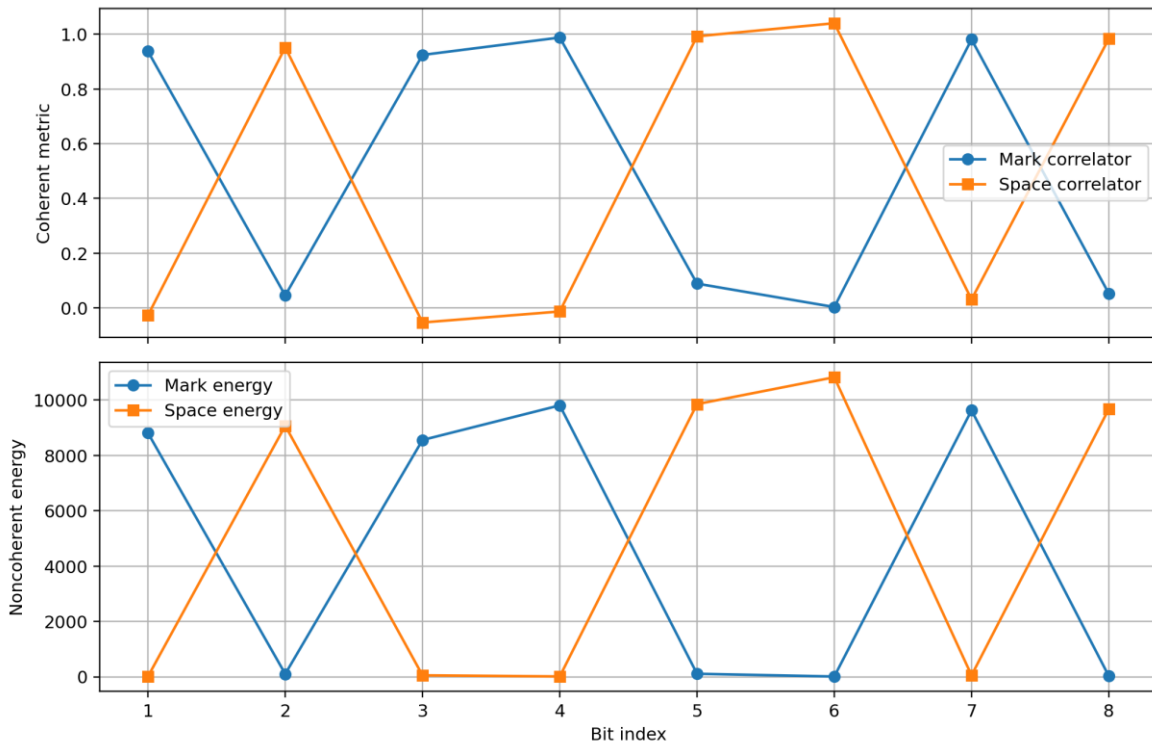


Fig. 6.8. Coherent and noncoherent BFSK decision metrics under additive noise.

Part 7: Parameter Investigation

1. Repeat the ASK and BFSK detectors for SNR values of 12, 8, 4, 0, and -4 dB.
2. For each SNR, transmit at least 10,000 random bits and calculate the empirical bit-error ratio. Do not derive theoretical BER expressions in this experiment.
3. For BFSK, repeat the simulation with frequency separations of 100, 300, and 500 Hz while keeping R_b constant.
4. Comment on the trade-off between frequency separation, measured bandwidth, and detection reliability.
5. Summarize the practical differences between ASK and BFSK.
6. Evaluation of Results

Tabulate the data obtained during the experiment and attach all MATLAB figures. Frequency values should be reported in hertz and decision outputs should be recorded with at least three decimal places.

Table 6.4. ASK generation and coherent detection results.

Quantity	Theoretical / expected	Measured
Carrier frequency		
Approximate minimum bandwidth		
Clean statistic for bit 1		
Clean statistic for bit 0		
Threshold		
Errors at 4 dB SNR		
Errors with 60° phase mismatch		

Table 6.5. ASK bit-by-bit detector output.

Bit index	Transmitted bit	z_k at 4 dB	Detected bit
1			
2			
3			
4			
5			
6			
7			
8			

Table 6.6. BFSK parameters and spectral measurements.

Quantity	Theoretical value	Measured value
Mark frequency f_M		
Space frequency f_S		
Center frequency f_c		
Frequency deviation Δf		
Baud rate		
Approximate minimum bandwidth		

Table 6.7(a). BFSK coherent-detector output.

Bit index	Transmitted bit	z_M	z_S	Detected bit
1				
2				
3				
4				
5				
6				
7				
8				

Table 6.7(b). BFSK noncoherent-detector output.

Bit index	Transmitted bit	E_M	E_S	Detected bit
1				
2				
3				
4				
5				
6				
7				
8				

Table 6.8. Noise and parameter investigation.

SNR (dB)	ASK errors / BER	Coherent BFSK errors / BER	Noncoherent BFSK errors / BER
12			

SNR (dB)	ASK errors / BER	Coherent BFSK errors / BER	Noncoherent BFSK errors / BER
8			
4			
0			
-4			

Questions

1. Why is binary ASK with a zero-amplitude symbol also called on-off keying?
2. Why does the ASK signal have a nonconstant envelope while the BFSK signal has an approximately constant envelope?
3. Why do abrupt changes in the binary waveform broaden the spectrum around the carrier?
4. What are the ASK coherent-detector outputs expected for ideal binary 1 and binary 0 intervals?
5. How does a carrier-phase error affect a coherent ASK detector?
6. Define mark frequency, space frequency, center frequency, and frequency deviation for BFSK.
7. Using the experiment parameters, compare the approximate minimum bandwidths of ASK and BFSK.
8. Why does the coherent BFSK receiver use two correlation branches?
9. Why is the squared sum of in-phase and quadrature correlations independent of carrier phase?
10. Which receiver, coherent or noncoherent BFSK, requires carrier-phase synchronization?
11. How does increasing the mark-space frequency separation affect the BFSK spectrum and receiver decisions?
12. At the same simulated SNR, which modulation method produced fewer errors in your implementation? Explain this observation using the measured statistics rather than a theoretical BER formula.
13. What limitations are introduced by finite observation time and FFT resolution when bandwidth is estimated from MATLAB plots?
14. Compare ASK and BFSK in terms of transmitter operation, receiver complexity, bandwidth, envelope, and sensitivity to amplitude variations.

Submit the following files:

- MATLAB script file with the .m extension
- Time-domain and frequency-domain figures for ASK and BFSK
- Completed Tables 6.1–6.8
- A short comparison of coherent and noncoherent detection results
- Written answers to the evaluation questions

Experiment 7: BPSK and Differential PSK

1. Objective of the Experiment

The objective of this experiment is to investigate binary phase-shift keying (BPSK) and differential phase-shift keying (DPSK/DBPSK), which convey digital information by controlling the phase of a sinusoidal carrier. Students will generate BPSK and DBPSK waveforms, examine their time-domain behavior, spectra, and signal-space representations, and recover the transmitted information using coherent and differential detection methods.

The BPSK section emphasizes the 0° and 180° carrier phases, polar symbol mapping, coherent product detection, and the effect of carrier-phase mismatch. The differential section demonstrates how information can be represented by the phase change between consecutive symbols, allowing detection without an explicit coherent carrier reference. The practical difference between absolute-phase and differential-phase reception is examined under additive noise and common phase offset.

At the end of this experiment, students are expected to:

- Explain how binary data are represented by two carrier phases separated by 180° .
- Generate polar NRZ data, BPSK, differentially encoded data, and DBPSK waveforms in MATLAB.
- Interpret the BPSK constellation and relate its two points to carrier phase.
- Calculate bit duration, baud rate, cycles per bit, and the approximate BPSK bandwidth.
- Implement a coherent BPSK receiver using multiplication, integration, and a zero threshold.
- Quantify the influence of local-carrier phase error on the coherent decision statistic.
- Implement differential encoding and delay-and-multiply DBPSK detection.
- Compare BPSK and DBPSK in terms of carrier synchronization, first-bit reference, and noise sensitivity.

2. Theoretical Background

2.1. Binary Phase-Shift Keying

Binary phase-shift keying uses two carrier phases to represent the two binary symbols. The lecture notes describe BPSK as a phase-reversal or biphase modulation method in which one bit is transmitted with a carrier phase of 0° and the other with a phase of 180° . A polar data sequence $a_k \in \{+1, -1\}$ therefore controls the sign of the carrier during each bit interval.

$$s_{BPSK}(t) = A_c a_k \cos(2\pi f_c t), \quad kT_b \leq t < (k+1)T_b$$

In this experiment, binary 1 is mapped to $a_k = +1$ and binary 0 is mapped to $a_k = -1$. Hence, the two transmitted waveforms are

$$s_1(t) = A_c \cos(2\pi f_c t)$$

$$s_0(t) = -A_c \cos(2\pi f_c t) = A_c \cos(2\pi f_c t + \pi)$$

Each BPSK signaling element carries one information bit. Consequently, the bit rate and symbol rate are equal:

$$R_s = R_b \quad \text{and} \quad T_b = 1/R_b$$

The lecture notes relate the worst-case BPSK spectrum to the highest fundamental component of the binary input. For binary signaling, the minimum double-sided Nyquist bandwidth is approximately equal to the input bit rate.

$$B_{BPSK,min} \approx R_b$$

BPSK Transmitter and Coherent Receiver

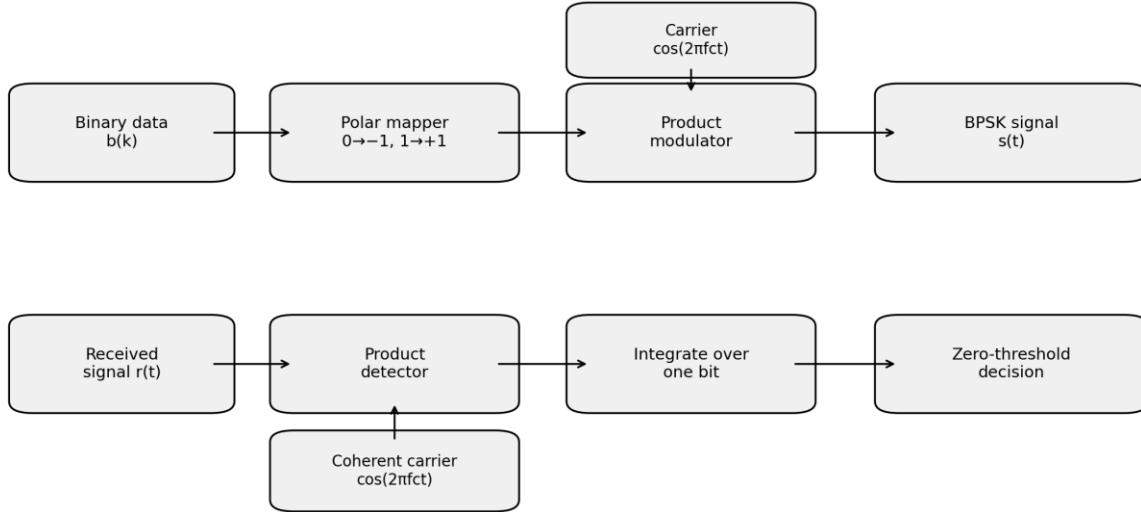


Fig. 7.1. BPSK transmitter and coherent receiver structures.

2.2. BPSK Signal Space and Constellation

Because both BPSK waveforms differ only by sign, they lie on a single signal-space axis and are antipodal. The two constellation points have equal amplitude and opposite phase. Their angular separation is 180°, which provides the largest possible separation for a binary constant-envelope modulation.

$$s_1 \leftrightarrow +\sqrt{E_b} \text{ and } s_0 \leftrightarrow -\sqrt{E_b}$$

The constellation makes the receiver decision rule clear: a positive correlator output is assigned to one bit and a negative output to the other. The bit labeling is a design convention, but the same mapping must be used at the transmitter and receiver.

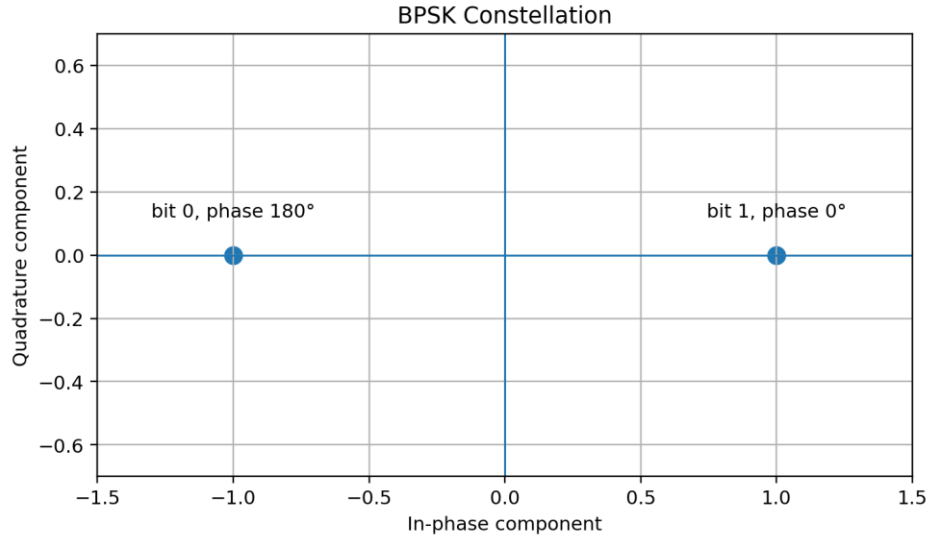


Fig. 7.2. BPSK constellation with antipodal signal points.

2.3. Coherent BPSK Detection

A coherent receiver reproduces a carrier that is synchronized in frequency and phase with the transmitter carrier. The received BPSK signal is multiplied by this reference and integrated over one bit interval. A normalized decision statistic for the k th bit is

$$z_k = (2/T_b) \int_{[kT_b \text{ to } (k+1)T_b]} r(t) \cos(2\pi f_c t) dt$$

Under ideal, noise-free synchronization, z_k is positive for $a_k = +1$ and negative for $a_k = -1$. A zero threshold is therefore used:

$$\hat{a}_k = +1, z_k \geq 0; \quad \hat{a}_k = -1, z_k < 0$$

If the local oscillator contains a phase error ϕ , the useful component of the coherent statistic is scaled approximately by $\cos\phi$. The receiver margin decreases as the error increases, and an ideal 90° mismatch removes the desired in-phase component. This sensitivity motivates accurate carrier recovery in an absolute-phase BPSK receiver.

$$z_k(\phi) \propto a_k \cos\phi$$

2.4. Differential Phase-Shift Keying

In differential phase-shift keying, information is represented by the phase relation between consecutive signaling intervals rather than by an absolute phase referenced to a local oscillator. The lecture notes describe a DBPSK transmitter that differentially encodes the input data before a BPSK modulator and a receiver that compares the received signal with a one-bit delayed version of itself.

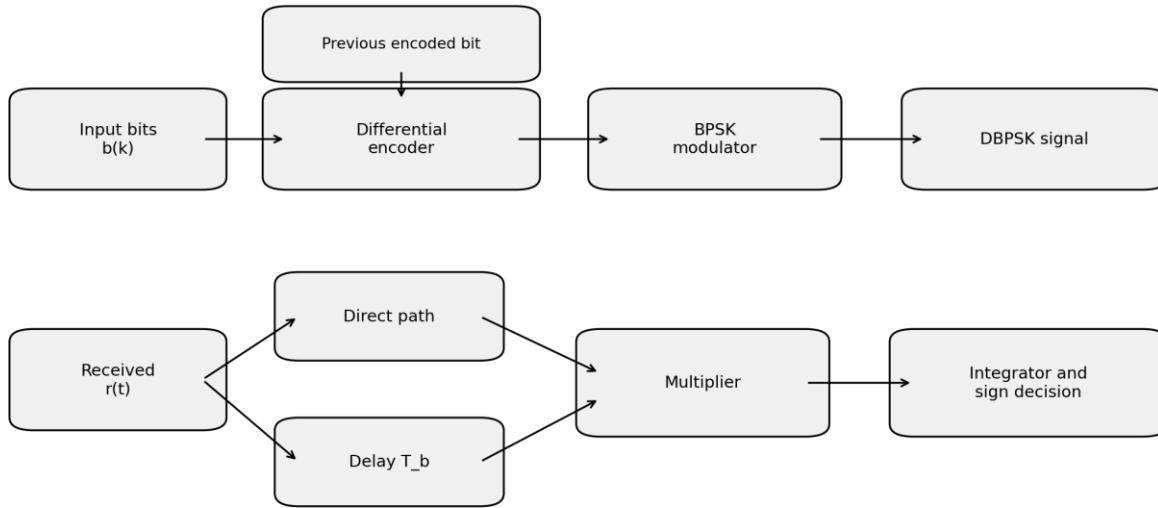
The exact assignment of a binary value to phase change is a convention. In this experiment, the following explicit convention is used: binary 0 produces no phase change and binary 1 produces a 180° phase reversal. With an initial reference state $d_{-1} = 1$, the encoded state is

$$d_k = d_{(k-1)}, b_k = 0; \quad d_k = 1 - d_{(k-1)}, b_k = 1$$

The lecture notes also describe an XNOR-based precoding representation. Although binary labels may differ with the chosen convention, the essential requirement is that the transmitter and receiver use the same differential rule.

$$s_{DBPSK}(t) = A_c (2d_k - 1) \cos(2\pi f_c t)$$

Differential PSK Transmitter and Delay-and-Multiply Receiver



The receiver compares the present symbol with the preceding symbol; an explicit carrier-recovery loop is not required.

Fig. 7.3. Differential encoder, DBPSK modulator, and delay-and-multiply receiver.

2.5. Differential Detection

A DBPSK receiver can multiply the current bit interval by the preceding bit interval and integrate the product. When the two phases are equal, the product metric is positive; when the phase differs by 180° , it is negative. With the convention used in this experiment, a positive metric is decoded as binary 0 and a negative metric as binary 1.

$$q_k = \int [kT_b \text{ to } (k+1)T_b] r(t) r(t-T_b) dt$$

$$\hat{b}_k = 0, q_k \geq 0; \quad \hat{b}_k = 1, q_k < 0$$

A constant phase rotation applied equally to consecutive symbols largely cancels in this comparison. Therefore, the differential receiver does not require an explicit carrier-recovery circuit. However, the first information bit requires an assumed reference state, and noise in two consecutive intervals contributes to each decision. The lecture notes state that DBPSK generally requires a higher signal-to-noise ratio than coherent BPSK for the same bit-error performance.

2.6. BPSK and DBPSK Comparison

Table 7.1. Conceptual comparison of BPSK and DBPSK.

Property	Coherent BPSK	DBPSK
Information representation	Absolute phase: 0° or 180°	Phase change between symbols
Carrier recovery	Required	Not required for delay detection
Decision reference	Local coherent carrier	Previous received symbol
First-bit handling	Directly detectable	Initial reference must be assumed
Effect of constant phase offset	Can strongly reduce the statistic	Common offset largely cancels
Noise performance in lecture notes	Better reference performance	Requires approximately 1–3 dB more SNR

3. Materials Used

- Computer
- MATLAB software
- MATLAB Script Editor
- Digital Communications lecture notes
- Laboratory worksheet
- Calculator

The following MATLAB functions and operations may be used:

- repelem, reshape, diff, xor, logical indexing, and loops
- cos, randn, mean, sum, sqrt, and deg2rad
- fft, fftshift, abs, angle, and complex-number operations
- plot, stairs, stem, scatter, subplot, grid, xlabel, ylabel, title, and legend

4. Preliminary (PreLab)

Complete the following calculations and tables before attending the laboratory. Use the parameter set below unless another value is stated.

Table 7.2. Parameters used in the preliminary study.

Parameter	Symbol	Value
Binary sequence	b	1 0 1 1 0 0 1 0 1 1
Bit rate	R _b	100 bit/s
Sampling frequency	F _s	20 kHz
Carrier frequency	f _c	1 kHz
Carrier amplitude	A _c	1 V
Initial DBPSK state	d ₋₁	1
Test SNR	SNR	2 dB

4.1. BPSK Calculations

1. Calculate the bit duration T_b and the number of samples per bit N_s .
2. Calculate the number of carrier cycles contained in one bit interval.
3. Write the polar sequence a_k corresponding to the given binary sequence.
4. Write the expected phase sequence using the mapping $1 \rightarrow 0^\circ$ and $0 \rightarrow 180^\circ$.
5. Calculate the baud rate and the approximate minimum BPSK bandwidth.
6. Calculate the ideal relative coherent-detector amplitudes at carrier phase errors of 0° , 30° , 60° , and 90° using $\cos\phi$.

Table 7.3. Preliminary BPSK calculations.

Quantity	Theoretical value	Unit
Bit duration T_b		s
Samples per bit N_s		samples
Carrier cycles per bit		cycles
Baud rate		baud
Approximate minimum bandwidth		Hz
$\cos 0^\circ$		-
$\cos 30^\circ$		-
$\cos 60^\circ$		-

Quantity	Theoretical value	Unit
$\cos 90^\circ$		-

4.2. Differential Encoding

1. Use $d_{-1} = 1$ and the experiment convention $b = 0 \rightarrow$ no phase change, $b = 1 \rightarrow$ phase reversal.
2. Calculate the encoded state d_k for every input bit.
3. Convert each encoded state to a carrier phase: $d_k = 1 \rightarrow 0^\circ$ and $d_k = 0 \rightarrow 180^\circ$.
4. Predict the sign of the delay-and-multiply statistic for each bit after the first reference interval.

Table 7.4. Differential encoding worksheet.

k	Input b_k	Previous state	Encoded state d_k	Carrier phase	Expected q_k sign
1	1				
2	0				
3	1				
4	1				
5	0				
6	0				
7	1				
8	0				
9	1				
10	1				

5. Procedure

Part 1: Define the Data and Generate the BPSK Waveform

1. Open MATLAB and create a new script file.
2. Define the binary sequence, bit rate, sampling frequency, carrier frequency, and carrier amplitude.
3. Generate a unipolar bit waveform and map it to a polar waveform with values +1 and -1.
4. Multiply the polar waveform by the sinusoidal carrier to obtain the BPSK signal.
5. Plot the binary data, polar data, carrier, and BPSK waveform in one figure.
6. Identify each 180° phase reversal and relate it to a change in the polar symbol.

```
clear; close all; clc;
rng(27);
bits = [1 0 1 1 0 0 1 0 1 1];
Rb = 100; Tb = 1/Rb;
Fs = 20000; Ns = round(Fs/Rb);
Ac = 1; fc = 1000;

b = repelem(bits,Ns);
a = 2*b - 1;           % 1 -> +1, 0 -> -1
t = (0:length(a)-1)/Fs;
carrier = cos(2*pi*fc*t);
sBPSK = Ac*a.*carrier;

figure;
subplot(4,1,1); stairs(t,b); grid on; ylabel('Bit'); ylim([-0.2 1.2]);
subplot(4,1,2); stairs(t,a); grid on; ylabel('Polar'); ylim([-1.3 1.3]);
```

```
subplot(4,1,3); plot(t,carrier); grid on; ylabel('Carrier');
subplot(4,1,4); plot(t,sBPSK); grid on; ylabel('BPSK'); xlabel('Time (s)');
```

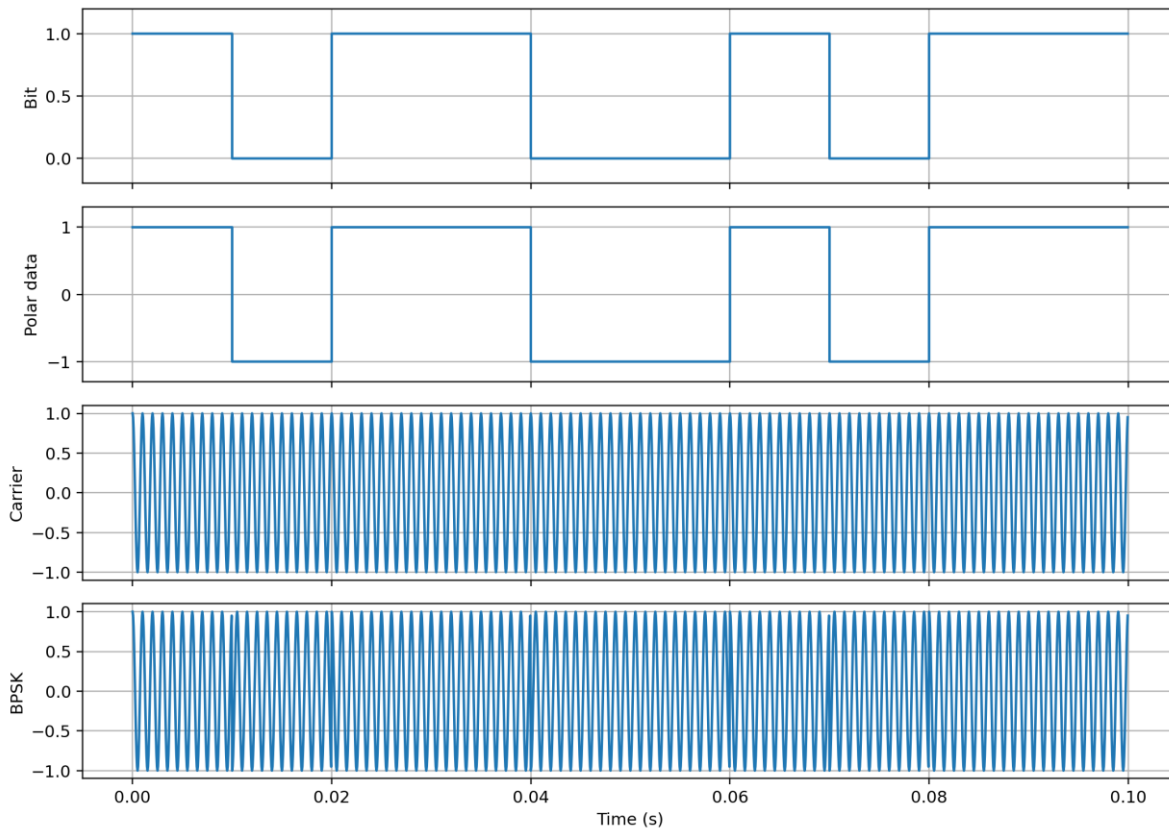


Fig. 7.4. Binary data, polar waveform, carrier, and BPSK signal.

Part 2: Frequency Spectrum and Constellation

1. Calculate a normalized, centered FFT of the BPSK waveform.
2. Plot the two-sided magnitude spectrum and identify the spectral region around $\pm f_c$.
3. Compare the measured spectral extent with the approximate bandwidth obtained in the PreLab.
4. Create the ideal BPSK constellation by plotting the two symbol values at $(+1,0)$ and $(-1,0)$.
5. Explain why the quadrature coordinate is zero for the ideal carrier reference used here.

```
N = length(sBPSK);
S = fftshift(fft(sBPSK))/N;
f = (-N/2:N/2-1)*(Fs/N);
figure; plot(f,abs(S),'LineWidth',1.1); grid on;
xlim([-2200 2200]); xlabel('Frequency (Hz)');
ylabel('|S_{BPSK}(f)|'); title('Two-sided BPSK spectrum');

figure; scatter([-1 1],[0 0],80,'filled'); grid on;
xline(0); yline(0); axis([-1.5 1.5 -0.7 0.7]);
xlabel('In-phase'); ylabel('Quadrature'); title('BPSK constellation');
```

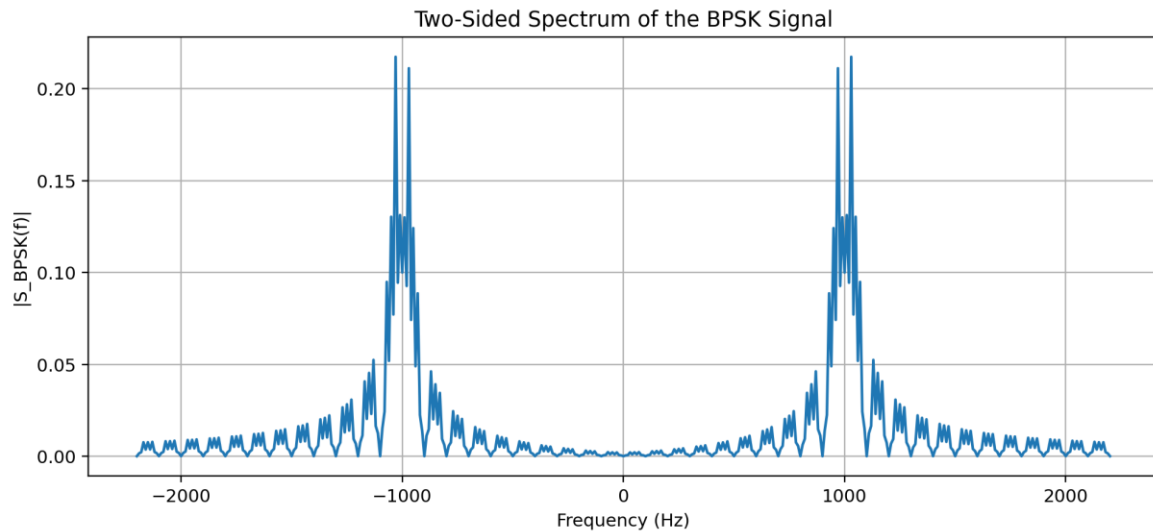


Fig. 7.5. Example two-sided spectrum of the BPSK waveform.

Part 3: Coherent BPSK Detection

1. Add real Gaussian noise to the BPSK signal at an SNR of 2 dB.
2. For each bit interval, multiply the received samples by a coherent cosine reference.
3. Integrate and normalize the product to obtain z_k .
4. Use a zero threshold to recover the polar symbols and binary bits.
5. Count the bit errors and record each decision statistic.
6. Repeat the detector for phase errors of 30° , 60° , and 90° .
7. Compare the measured statistic reduction with the expected $\cos\phi$ relationship.

```

SNRdB = 2;
Ps = mean(sBPSK.^2);
Pn = Ps/10^(SNRdB/10);
r = sBPSK + sqrt(Pn)*randn(size(sBPSK));

phaseDeg = [0 30 60 90];
Z = zeros(length(phaseDeg),length(bits));
Bhat = zeros(length(phaseDeg),length(bits));
for p = 1:length(phaseDeg)
    phi = deg2rad(phaseDeg(p));
    for k = 1:length(bits)
        idx = (k-1)*Ns + (1:Ns);
        ref = cos(2*pi*fc*t(idx) + phi);
        Z(p,k) = 2*sum(r(idx).*ref)/Ns;
    end
    Bhat(p,:) = Z(p,:) >= 0;
end
errors = sum(Bhat ~= bits,2);

```

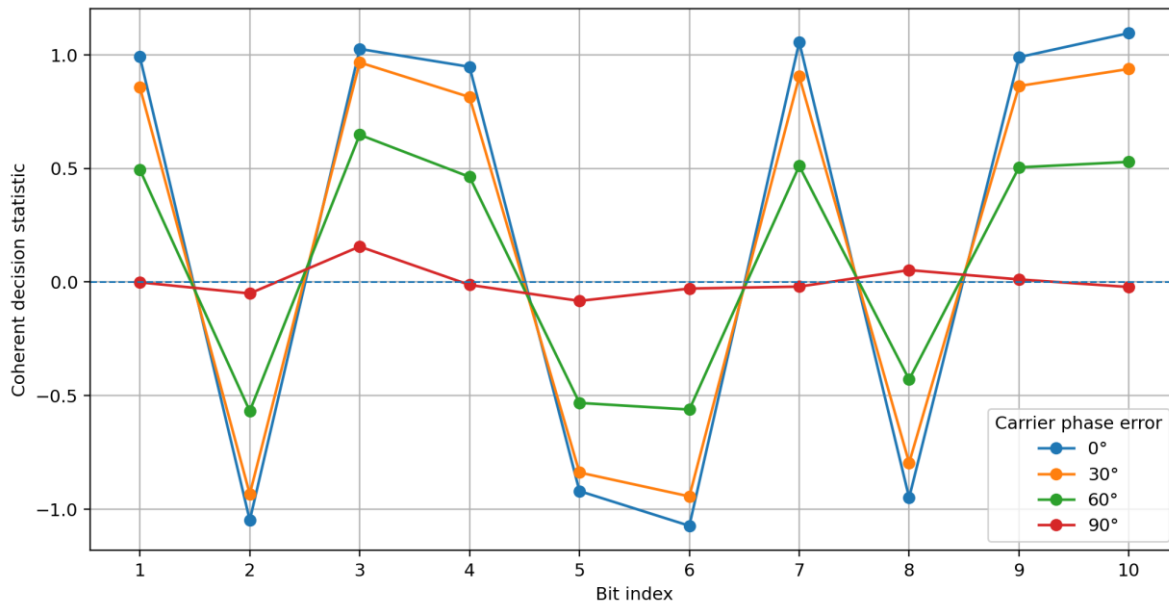


Fig. 7.6. Coherent BPSK decision statistics for different carrier-phase errors.

Part 4: Differential Encoding and DBPSK Generation

1. Set the initial differential reference state to 1.
2. Encode the input sequence using $b = 0 \rightarrow$ no state change and $b = 1 \rightarrow$ state inversion.
3. Map the encoded states to +1 and -1 and multiply them by the carrier.
4. Plot the input bits, differentially encoded states, and DBPSK waveform.
5. Complete Table 7.4 and verify every calculated state using MATLAB.

```
d = zeros(size(bits));
previous = 1;
for k = 1:length(bits)
    if bits(k) == 0
        d(k) = previous;      % no phase change
    else
        d(k) = 1 - previous;  % phase reversal
    end
    previous = d(k);
end

dPolar = 2*d - 1;
dWave = repelem(dPolar,Ns);
sDBPSK = Ac*dWave.*carrier;

figure;
subplot(3,1,1); stairs(t,b); grid on; ylabel('Input bit');
subplot(3,1,2); stairs(t,dWave); grid on; ylabel('Encoded state');
subplot(3,1,3); plot(t,sDBPSK); grid on; ylabel('DBPSK'); xlabel('Time (s)');
```

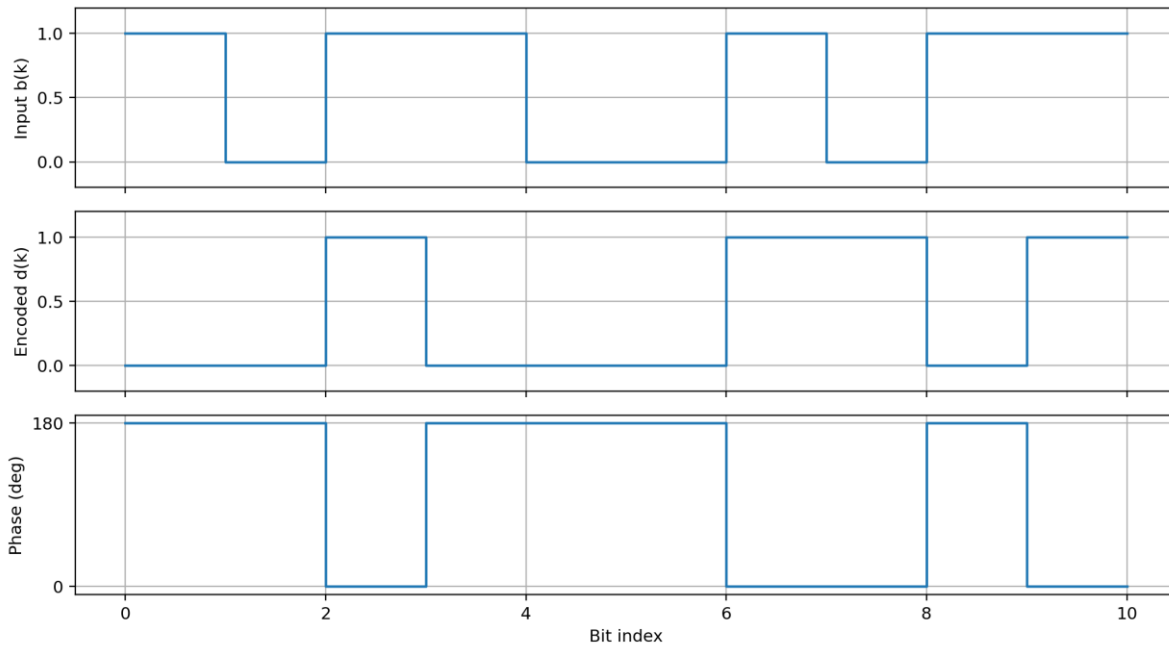


Fig. 7.7. Example input bits, differential states, and transmitted phase sequence.

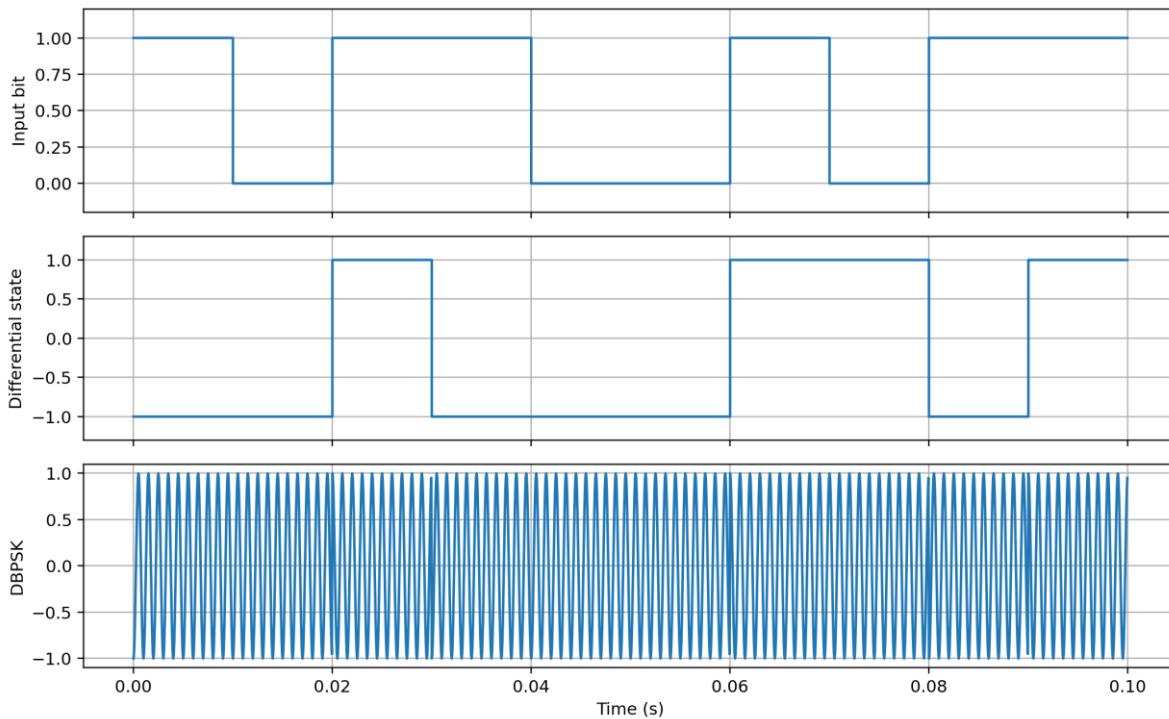


Fig. 7.8. DBPSK waveform generated from the differential state sequence.

Part 5: Delay-and-Multiply DBPSK Detection

1. Add Gaussian noise to the DBPSK waveform at 2 dB SNR.
2. Apply a constant phase offset of 75° to the received carrier to represent an unknown common phase.
3. For each bit after the first interval, multiply the current interval by the preceding interval and sum the product.
4. Decode a positive metric as binary 0 and a negative metric as binary 1.

- Compare the recovered bits with input bits 2 through N and count the errors.
- Explain why the common phase offset does not create the same loss as in the coherent BPSK detector.

```
phi0 = deg2rad(75);
sShifted = Ac*dWave.*cos(2*pi*fc*t + phi0);
Ps = mean(sShifted.^2);
Pn = Ps/10^(SNRdB/10);
rD = sShifted + sqrt(Pn)*randn(size(sShifted));

q = zeros(1,length(bits)-1);
for k = 2:length(bits)
    idxNow = (k-1)*Ns + (1:Ns);
    idxPrev = (k-2)*Ns + (1:Ns);
    q(k-1) = 2*sum(rD(idxNow).*rD(idxPrev))/Ns;
end
bitsDhat = q < 0; % negative product -> phase reversal -> bit 1
errorsD = sum(bitsDhat ~= bits(2:end));
```

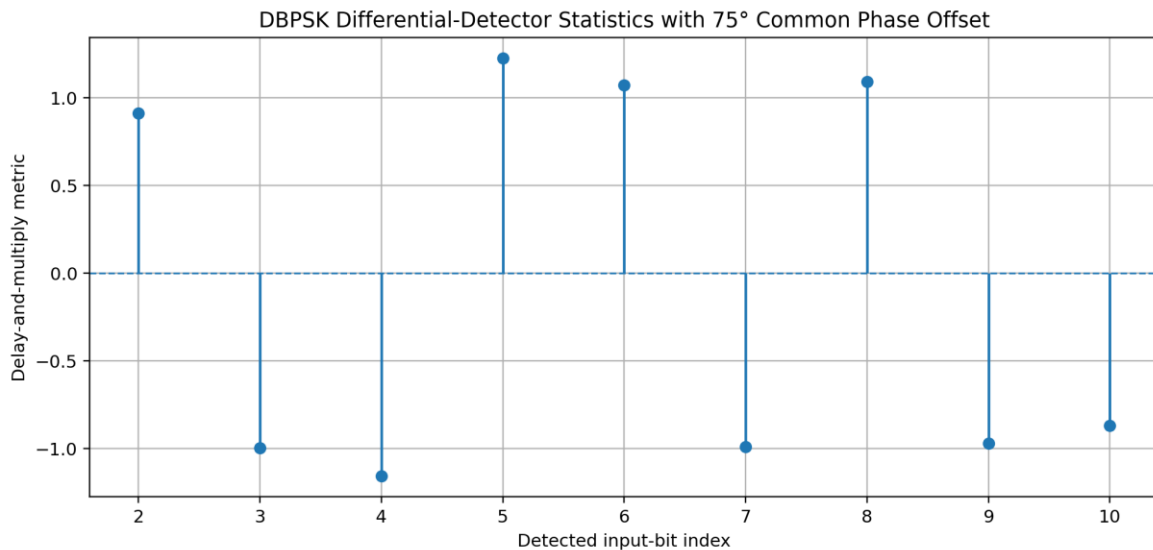


Fig. 7.9. Differential-detector metrics in the presence of noise and a common phase offset.

Part 6: Receiver Comparison

- Run the coherent BPSK detector and differential DBPSK detector for SNR values -4 , 0 , 4 , and 8 dB.
- For each SNR, use at least 1000 randomly generated bits to obtain a meaningful empirical error count.
- Apply a 60° phase error to the coherent BPSK reference and a common 60° phase offset to the DBPSK waveform.
- Record the error rates, receiver synchronization requirement, and first-bit treatment.
- Do not interpret the short simulation as a complete theoretical BER study; detailed matched-filter BER analysis will be performed in a later experiment.

```
% Suggested loop structure for the comparison
SNRlist = [-4 0 4 8];
Nbits = 1000;
% 1) Generate random bits and corresponding BPSK/DBPSK waveforms.
% 2) Add noise at each SNR value.
% 3) Detect coherent BPSK with a selected carrier phase error.
```


% 4) Detect DBPSK by one-bit delay and multiplication.
 % 5) Store empirical BER = number_of_errors / number_of_tested_bits.

6. Evaluation of Results

Record the calculated and measured values in the following tables. Use the same bit mapping and differential convention throughout the report.

Table 7.5. BPSK parameter verification.

Quantity	Theoretical value	MATLAB result	Percentage error
Bit duration T_b			
Samples per bit N_s			
Carrier cycles per bit			
Baud rate			
Approximate bandwidth			

Table 7.6. Coherent BPSK decisions at 2 dB SNR.

Bit index	Transmitted bit	z_k at 0°	Detected bit	Correct / Error
1	1			
2	0			
3	1			
4	1			
5	0			
6	0			
7	1			
8	0			
9	1			
10	1			

Table 7.7. Effect of local-carrier phase error.

Phase error	Expected scale $\cos\phi$	Mean $ z_k $	Detected errors	Interpretation
0°				
30°				
60°				
90°				

Table 7.8. DBPSK differential detection results.

Detected bit index	Transmitted bit	q_k	Detected bit	Correct / Error
2	0			
3	1			
4	1			
5	0			
6	0			
7	1			
8	0			
9	1			
10	1			

Table 7.9. Empirical receiver comparison.

SNR (dB)	BPSK BER, ideal phase	BPSK BER, 60° error	DBPSK BER, common 60° offset
-4			
0			
4			
8			

Table 7.10. Final comparison.

Criterion	BPSK	DBPSK
Information variable		
Carrier synchronization		
Decision threshold / rule		
First-bit treatment		
Response to constant phase offset		
Observed noise performance		

Questions

1. Why are the two BPSK carrier phases separated by 180° ?
2. Why does BPSK have a constant envelope even though its phase reverses?
3. How are bit rate and baud rate related for binary PSK?
4. What information is provided by the BPSK constellation diagram?
5. Why is a zero threshold suitable for the ideal coherent BPSK detector?
6. Explain the $\cos\phi$ dependence of the coherent receiver output.
7. What happens to an ideal coherent BPSK receiver at a 90° phase error?
8. How does differential encoding convert input bits into phase transitions?
9. Why must an initial reference state be selected for DBPSK?
10. Why does a constant carrier-phase offset largely cancel in delay-and-multiply detection?
11. Why can DBPSK have poorer noise performance than coherent BPSK?
12. Compare the spectra of BPSK and DBPSK generated with the same bit rate.
13. Explain any differences between the manually calculated differential sequence and the MATLAB result.
14. Based on the experiment, state one practical advantage and one practical disadvantage of DBPSK.

Experiment 8: QPSK Modulation, Constellation and Coherent Detection

1. Objective of the Experiment

The objective of this experiment is to investigate quadrature phase-shift keying (QPSK), a digital phase-modulation technique that transmits two information bits during each signaling interval. Students will divide a serial binary sequence into in-phase and quadrature branches, generate the corresponding orthogonal carrier components, combine them to obtain a QPSK waveform, and recover the original data using coherent I/Q detection.

The experiment emphasizes the relationship among dibits, carrier phase, constellation points, symbol rate, and bandwidth. It also demonstrates how the two product detectors in the coherent receiver separately recover the I and Q data streams and how an error in the recovered carrier phase rotates the received constellation relative to the fixed decision axes.

At the end of this experiment, students are expected to:

- Explain why QPSK transmits two bits per symbol and identify its four possible phase states.
- Split a serial binary sequence into I and Q branches and map binary values to polar coefficients.
- Generate QPSK using two orthogonal carrier signals in MATLAB.
- Construct and interpret the QPSK truth table, phasor diagram, and constellation diagram.
- Calculate bit rate, symbol rate, symbol duration, and carrier cycles per symbol.
- Compare QPSK and BPSK in terms of bits per symbol and signaling rate.
- Implement coherent QPSK detection using I and Q product detectors, integration, and zero thresholds.
- Recombine detected I and Q bits into the original serial sequence.
- Examine the effect of common carrier-phase error on constellation rotation and symbol decisions.

2. Theoretical Background

2.1. Quadrature Phase-Shift Keying

The lecture notes define QPSK as a phase-shift-keying technique with four possible carrier phases. The serial input is grouped into dibits, so each signaling element represents two binary digits. Because two bits are conveyed per symbol, the QPSK symbol rate is one-half of the input bit rate.

$$M = 4, \quad \log_2 M = 2 \text{ bits/symbol}$$
$$R_s = R_b/2, \quad T_s = 1/R_s = 2T_b$$

The serial-to-parallel converter separates every dibit into an in-phase bit I and a quadrature bit Q. Each branch is converted into a polar coefficient in the set $\{-1, +1\}$. The I branch modulates a sine carrier, while the Q branch modulates a cosine carrier that is 90° out of phase with the sine carrier. The two modulated branches are then added.

$$s_{\text{QPSK}}(t) = (A_c/\sqrt{2})[I_k \sin(2\pi f_c t) + Q_k \cos(2\pi f_c t)]$$

The normalization factor $1/\sqrt{2}$ keeps the resultant symbol energy constant when both branch coefficients have magnitude one. The lecture-note expressions use the same I/Q addition without explicitly showing this normalization; the MATLAB implementation uses it to obtain a unit-amplitude constellation.

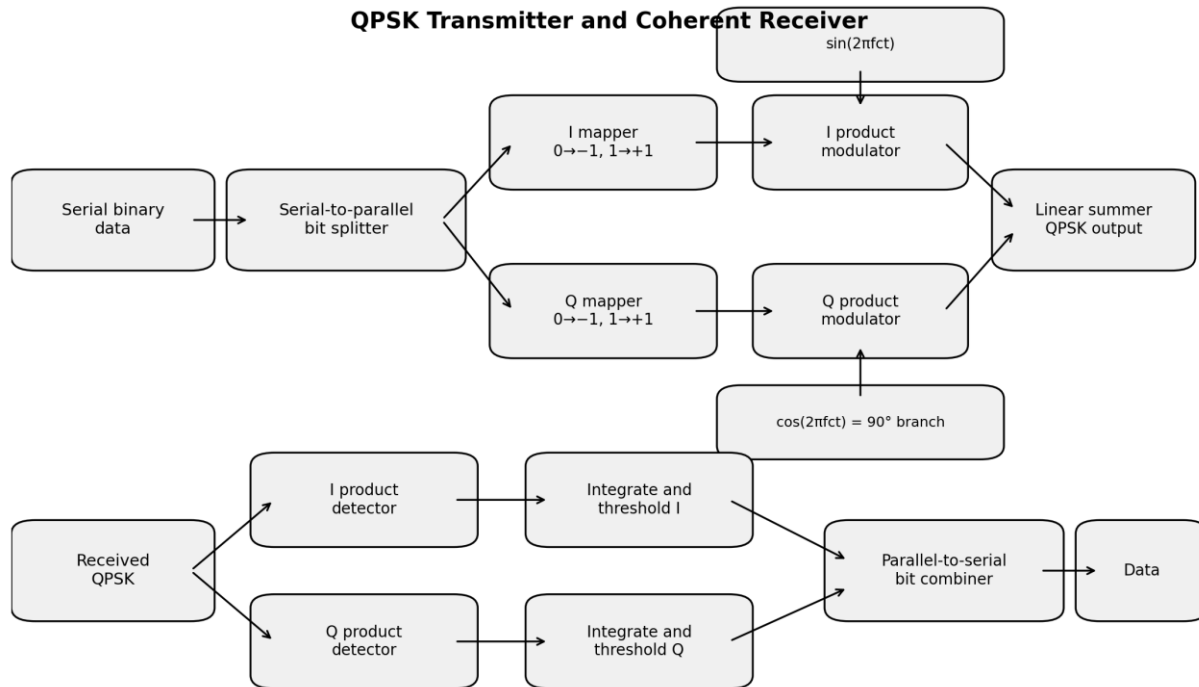


Fig. 8.1. QPSK transmitter and coherent receiver structures.

2.2. Dibit Mapping and Carrier Phases

Using the lecture-note convention, the first bit of each dibit is assigned to the Q branch and the second bit is assigned to the I branch. A logic 1 is mapped to +1 and a logic 0 is mapped to -1. The resultant phase is determined by the signs of the two orthogonal components.

Table 8.1. QPSK mapping used in this experiment.

Dibit (Q I)	I coefficient	Q coefficient	Resultant expression	Phase
00	-1	-1	$-\sin(\omega ct) - \cos(\omega ct)$	-135°
01	+1	-1	$+\sin(\omega ct) - \cos(\omega ct)$	-45°
11	+1	+1	$+\sin(\omega ct) + \cos(\omega ct)$	$+45^\circ$
10	-1	+1	$-\sin(\omega ct) + \cos(\omega ct)$	$+135^\circ$

Adjacent QPSK phases are separated by 90° . The lecture notes state that a phase displacement approaching $\pm 45^\circ$ can be tolerated before a point crosses a neighboring decision boundary in an ideal noise-free receiver. In practice, noise reduces this margin.

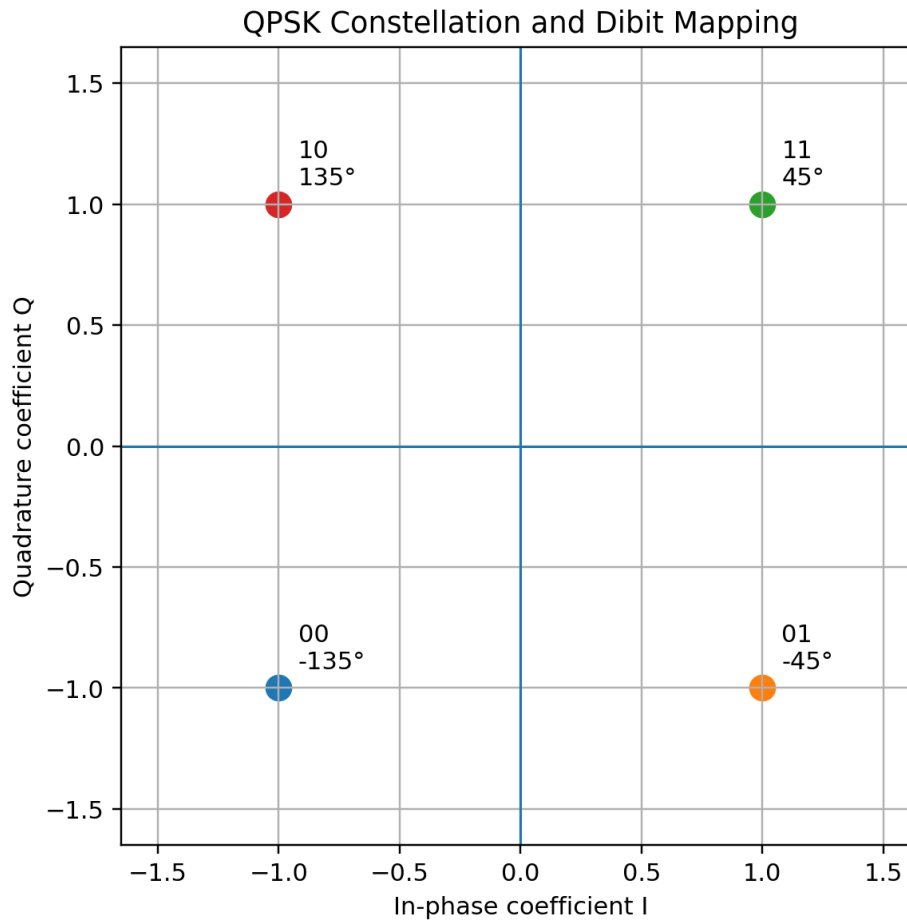


Fig. 8.2. QPSK constellation and dibit-to-phase assignment.

2.3. QPSK Constellation and Signal Space

The ideal QPSK constellation contains four points with equal distance from the origin. Therefore, all symbols have equal amplitude and the information is carried by phase. The horizontal coordinate represents the I coefficient and the vertical coordinate represents the Q coefficient. The two axes are orthogonal because the sine and cosine carriers are orthogonal over an integer number of carrier cycles.

$$\int [kT_s \text{ to } (k+1)T_s] \sin(2\pi f_c t) \cos(2\pi f_c t) dt = 0$$

Each quadrant corresponds to one dibit. A coherent receiver decides the sign of each coordinate separately. The detected I and Q signs are converted to binary values and then recombined in the same order used by the transmitter.

2.4. Symbol Rate and Bandwidth Considerations

QPSK transmits two bits per symbol, whereas BPSK transmits one bit per symbol. For the same input bit rate, the I and Q branches each operate at one-half of the bit rate. Thus, QPSK requires a lower signaling rate than BPSK. The lecture notes use this property to explain the bandwidth advantage of QPSK.

$$R_{s,QPSK} = R_b/2$$

For rectangular symbol pulses, the null-to-null spectral width is determined by the symbol rate. The exact occupied bandwidth depends on pulse shaping; in this experiment, no additional pulse-shaping filter is applied, so the FFT spectrum is interpreted qualitatively rather than as a regulatory occupied-bandwidth measurement.

2.5. Coherent QPSK Receiver

The coherent QPSK receiver uses two product detectors. One branch multiplies the received signal by the recovered sine carrier and the other branch uses the recovered cosine carrier. Each product is integrated over one symbol interval. Under ideal synchronization, the normalized decision variables are proportional to the transmitted I and Q coefficients.

$$z_I(k) = (2\sqrt{2}/T_s) \int r(t) \sin(2\pi f_c t) dt$$

$$z_Q(k) = (2\sqrt{2}/T_s) \int r(t) \cos(2\pi f_c t) dt$$

$$\hat{I}_k = +1 \text{ for } z_I \geq 0, \text{ otherwise } -1; \quad \hat{Q}_k = +1 \text{ for } z_Q \geq 0, \text{ otherwise } -1$$

The parallel decisions are mapped back to binary values and converted to a serial sequence. Correct recovery requires the local carriers to be coherent with the transmitter carriers in both frequency and phase.

2.6. Effect of Carrier-Phase Error

A common phase error rotates the received QPSK constellation relative to the receiver decision axes. If the received coordinates are I and Q and the local reference contains a phase error ϕ , the receiver observes a rotated pair of coordinates. A convenient rotation model is

$$[I_r \ Q_r]^T = [[\cos\phi, -\sin\phi], [\sin\phi, \cos\phi]] [I \ Q]^T$$

For phase errors below 45° in the ideal noise-free case, every point remains in its original quadrant. At 45° , the points lie on decision boundaries; beyond 45° , some dibits are assigned to neighboring quadrants. This experiment visualizes the rotation and measures the resulting decision errors.

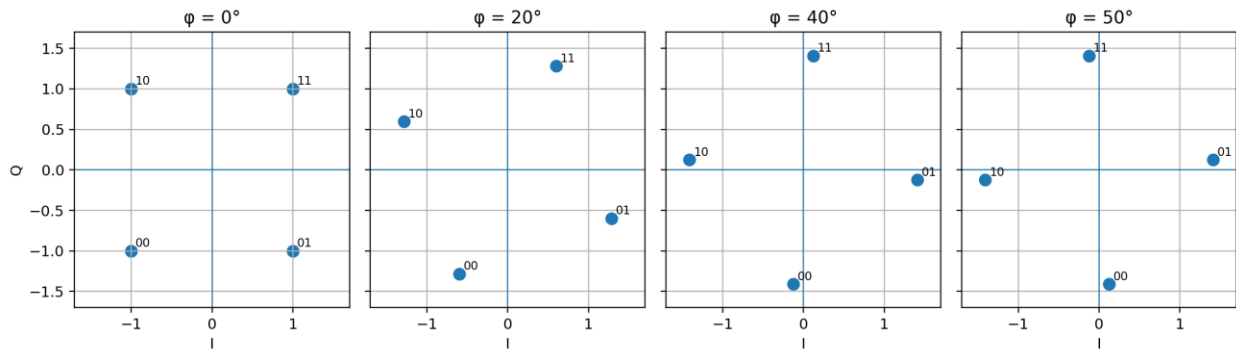


Fig. 8.3. Rotation of the QPSK constellation for selected carrier-phase errors.

3. Materials Used

- Computer
- MATLAB software
- MATLAB Script Editor
- Digital Communications lecture notes

- Laboratory worksheet
- Calculator

The following MATLAB functions and commands will be useful:

- repelem
- reshape
- sin
- cos
- fft
- fftshift
- scatter
- subplot
- stairs
- randn
- sum
- mean
- plot
- xline
- yline
- legend
- grid

4. Preliminary (PreLab)

Complete all calculations before the laboratory session. Use the dibit convention in Table 8.1 throughout the experiment.

4.1. System Parameters

For the laboratory, use the following values:

Table 8.2. Assigned QPSK parameters.

Parameter	Symbol	Value
Bit rate	R_b	200 bit/s
Carrier frequency	f_c	1000 Hz
Sampling frequency	F_s	20000 Hz
Carrier amplitude	A_c	1
Input bits	b	00 01 11 10 01 10

1. Calculate the bit duration T_b .
2. Calculate the QPSK symbol rate R_s .
3. Calculate the symbol duration T_s .
4. Calculate the number of samples per symbol N_s .
5. Calculate the number of carrier cycles within one symbol.
6. Determine the number of QPSK symbols formed by the assigned bit sequence.
7. Write the Q-bit and I-bit sequences obtained by serial-to-parallel conversion.
8. Use Table 8.1 to determine the I coefficient, Q coefficient, and phase for every dibit.

Table 8.3. Preliminary parameter calculations.

Quantity	Expression	Calculated value
Bit duration		
Symbol rate		
Symbol duration		
Samples per symbol		
Carrier cycles per symbol		
Number of symbols		

Table 8.4. Manual serial-to-parallel conversion and phase assignment.

Symbol index	Dibit (Q I)	Q coefficient	I coefficient	Phase
1	00			
2	01			
3	11			
4	10			
5	01			
6	10			

4.2. Conceptual Questions

1. Why does QPSK transmit two bits during one symbol interval?
2. Why are the I and Q carrier waveforms described as orthogonal?
3. Why do all four ideal QPSK constellation points have the same magnitude?
4. How does the QPSK symbol rate compare with the BPSK symbol rate for the same bit rate?
5. What happens to the decision margin when the carrier-phase error approaches 45°?

5. Procedure

Part 1: Serial-to-Parallel Conversion

1. Open MATLAB and create a new script file.
2. Define the input bit sequence, bit rate, symbol rate, sampling frequency, carrier frequency, and amplitude.
3. Confirm that the number of input bits is even.
4. Assign the first bit of each dibit to the Q branch and the second bit to the I branch.
5. Map binary 0 to -1 and binary 1 to +1.
6. Plot the serial input and the resulting I and Q branch sequences.
7. Compare the MATLAB sequences with Table 8.4.

<pre> bits = [0 0 0 1 1 1 1 0 0 1 1 0]; Rb = 200; % bit/s Rs = Rb/2; % QPSK symbol/s Fs = 20000; % sample/s fc = 1000; % Hz Ac = 1; Ns = round(Fs/Rs); Nsym = length(bits)/2; t = (0:Nsym*Ns-1)/Fs; Qbits = bits(1:2:end); % first bit of each dibit Ibits = bits(2:2:end); % second bit of each dibit </pre>	
---	--


```

Qsym = 2*Qbits - 1;
Isym = 2*Ibits - 1;
Qwave = repelem(Qsym,Ns);
Iwave = repelem(Isym,Ns);

```

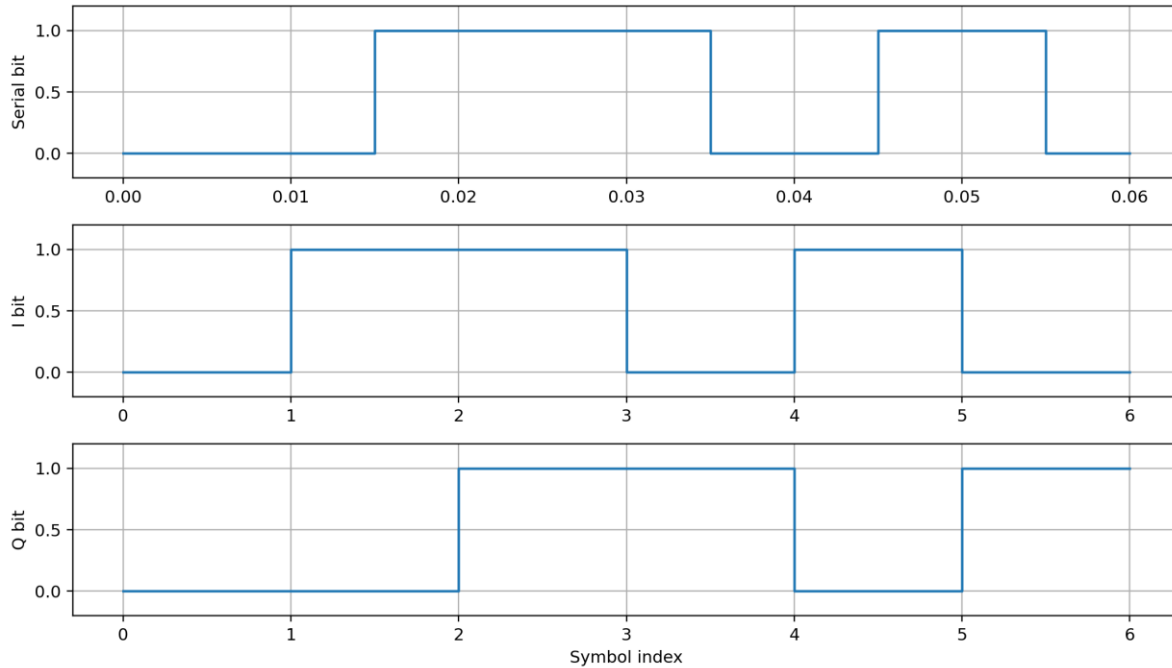


Fig. 8.4. Example serial input and the resulting I/Q branch sequences.

Part 2: QPSK Signal Generation

1. Generate the sine carrier for the I branch and the cosine carrier for the Q branch.
2. Multiply each polar data branch by its corresponding carrier.
3. Add the two branches and divide by $\sqrt{2}$ to obtain the normalized QPSK signal.
4. Plot the I data, Q data, I-modulated branch, Q-modulated branch, and total QPSK waveform.
5. Identify the phase associated with each symbol interval and verify it against Table 8.1.
6. Observe that the QPSK envelope remains constant except for numerical effects at symbol transitions.

```

carrierI = sin(2*pi*fc*t);
carrierQ = cos(2*pi*fc*t);
sI = Iwave.*carrierI;
sQ = Qwave.*carrierQ;
sQPSK = (sI + sQ)/sqrt(2);

figure;
subplot(5,1,1); stairs(t,Iwave); grid on; ylabel('I data');
subplot(5,1,2); stairs(t,Qwave); grid on; ylabel('Q data');
subplot(5,1,3); plot(t,sI); grid on; ylabel('I branch');
subplot(5,1,4); plot(t,sQ); grid on; ylabel('Q branch');
subplot(5,1,5); plot(t,sQPSK); grid on; ylabel('QPSK'); xlabel('Time (s)');

```

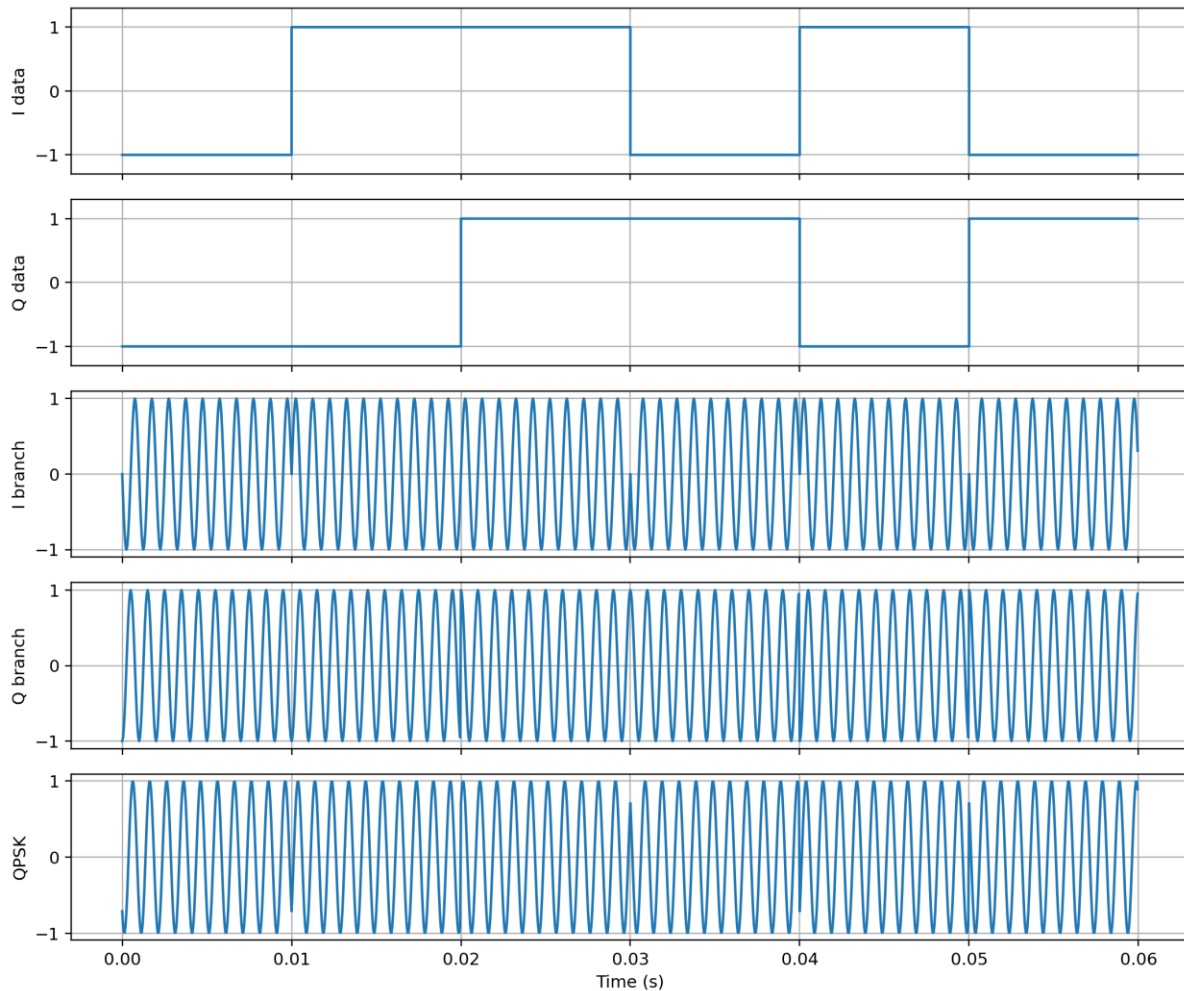


Fig. 8.5. I/Q branches and the resulting QPSK waveform.

Part 3: Constellation and Spectrum

1. Plot the four ideal constellation points using the I coefficient on the horizontal axis and the Q coefficient on the vertical axis.
2. Label every point with its dibit and phase.
3. Calculate the FFT of the QPSK waveform and construct a centered two-sided frequency axis.
4. Plot the magnitude spectrum around the carrier frequency.
5. Compare the observed spectral extent with the QPSK symbol rate.
6. Compare qualitatively with the BPSK spectrum obtained in Experiment 7 for the same input bit rate.

```
figure;
scatter([-1 1 1 -1],[-1 -1 1 1],90,'filled');
xline(0); yline(0); grid on; axis equal;
xlabel('I coefficient'); ylabel('Q coefficient');
title('QPSK constellation');
text([-1 1 1 -1]+0.08,[-1 -1 1 1]+0.08,{'00','01','11','10'});

N = length(sQPSK);
S = fftshift(fft(sQPSK))/N;
f = (-N/2:N/2-1)*(Fs/N);
figure; plot(f,abs(S),'LineWidth',1.1); grid on;
xlim([-1800 1800]); xlabel('Frequency (Hz)'); ylabel('|S(f)|');
```

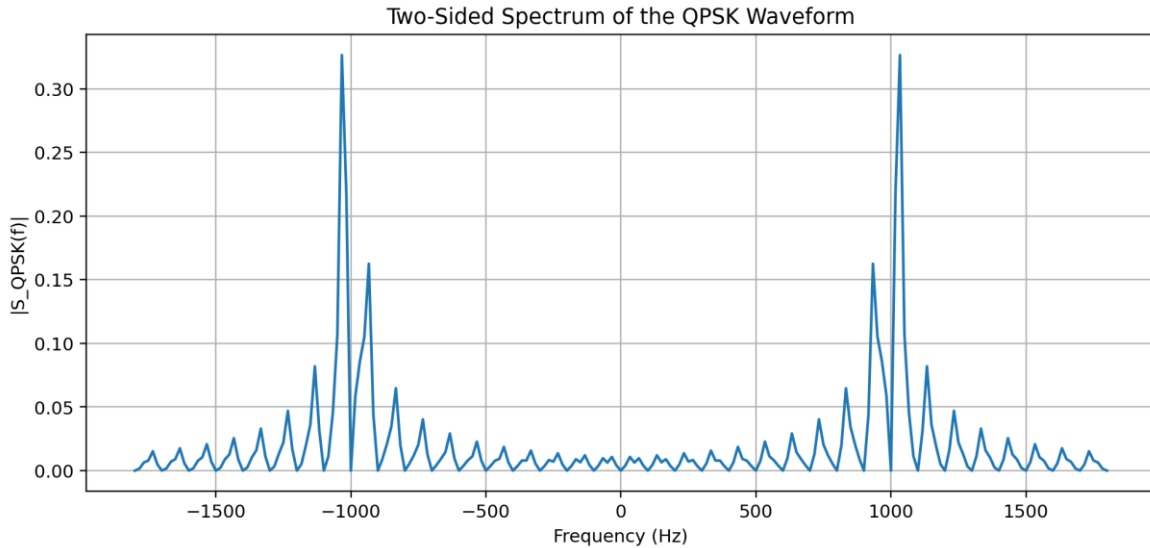


Fig. 8.6. Example two-sided spectrum of the QPSK signal.

Part 4: Coherent QPSK Detection

1. Add real Gaussian noise to the QPSK waveform at an SNR of 4 dB.
2. For each symbol interval, multiply the received signal by the local sine and cosine references.
3. Integrate and normalize both products to obtain z_I and z_Q .
4. Use zero thresholds to decide the I and Q coefficients.
5. Convert the detected coefficients back to binary values.
6. Recombine the Q and I bits into a serial sequence in the same order used at the transmitter.
7. Count the bit errors and plot the decision statistics on the I-Q plane.

```
SNRdB = 4;
Ps = mean(sQPSK.^2);
Pn = Ps/10^(SNRdB/10);
r = sQPSK + sqrt(Pn)*randn(size(sQPSK));

zI = zeros(1,Nsym); zQ = zeros(1,Nsym);
for k = 1:Nsym
    idx = (k-1)*Ns + (1:Ns);
    zI(k) = 2*sqrt(2)*sum(r(idx).*carrierI(idx))/Ns;
    zQ(k) = 2*sqrt(2)*sum(r(idx).*carrierQ(idx))/Ns;
end
Ihat = zI >= 0;
Qhat = zQ >= 0;
bitsHat = reshape([Qhat; Ihat],1,[]);
errors = sum(bitsHat ~= bits);

figure; scatter(zI,zQ,80,'filled'); xline(0); yline(0);
grid on; axis equal; xlabel('I statistic'); ylabel('Q statistic');
```

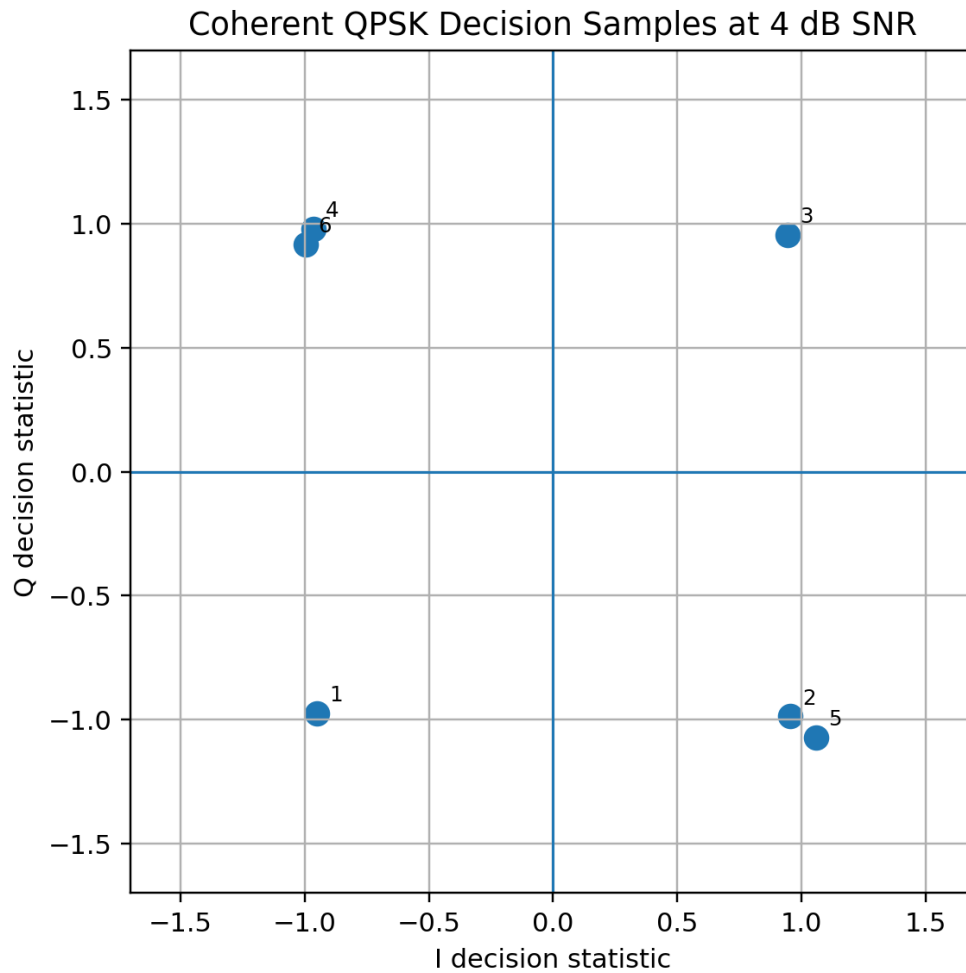


Fig. 8.7. Coherent QPSK decision samples at 4 dB SNR.

Part 5: Carrier-Phase Error

1. Repeat the coherent receiver with common phase errors of 0° , 20° , 40° , and 50° .
2. Plot the received decision samples for each phase error.
3. Record the number of bit errors for each case.
4. Explain why the ideal noise-free decision begins to fail when the phase error exceeds approximately 45° .
5. Repeat with 4 dB SNR and comment on why errors may occur below 45° when noise is present.

```

phaseDeg = [0 20 40 50];
for p = 1:length(phaseDeg)
    phi = deg2rad(phaseDeg(p));
    % equivalent constellation rotation before sign decisions
    zIrot = zI*cos(phi) - zQ*sin(phi);
    zQrot = zI*sin(phi) + zQ*cos(phi);
    Ihat = zIrot >= 0;
    Qhat = zQrot >= 0;
    bitsHat = reshape([Qhat; Ihat],1,[]);
    fprintf('Phase error %g deg: %d errors\n',phaseDeg(p),sum(bitsHat~=bits));
end

```

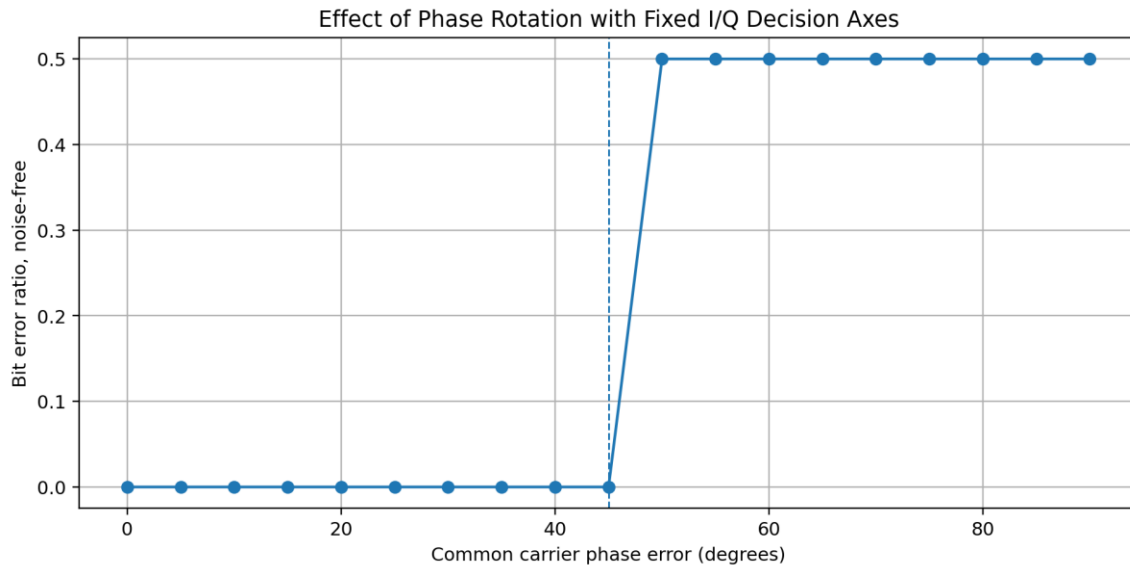


Fig. 8.8. Noise-free QPSK bit decisions as a function of common phase rotation.

Part 6: QPSK and BPSK Signaling-Rate Comparison

1. Use the same input bit rate for BPSK and QPSK.
2. Calculate the BPSK and QPSK symbol rates.
3. Generate BPSK and QPSK waveforms with rectangular symbol pulses.
4. Compare their symbol durations, carrier cycles per symbol, and FFT spectra.
5. Explain why QPSK can transmit the same bit rate with half the symbol rate.
6. Do not use this short comparison as a complete BER analysis; matched-filter and BER performance will be studied in Experiment 9.

```
% Signaling-rate comparison
RsBPSK = Rb;
RsQPSK = Rb/2;
TsBPSK = 1/RsBPSK;
TsQPSK = 1/RsQPSK;
fprintf('BPSK symbol rate = %.1f baud\n',RsBPSK);
fprintf('QPSK symbol rate = %.1f baud\n',RsQPSK);
```

6. Evaluation of Results

Record all calculated and MATLAB-obtained values. Use the same Q-I dibit order throughout the report.

Table 8.5. QPSK parameter verification.

Quantity	Theoretical value	MATLAB result	Percentage error
Bit duration T_b			
Symbol rate R_s			
Symbol duration T_s			
Samples per symbol N_s			
Carrier cycles per symbol			

Table 8.6. Generated QPSK symbol sequence.

Symbol	Dibit (Q I)	I coefficient	Q coefficient	Theoretical phase	Observed phase
1	00	-1	-1		
2	01	1	-1		
3	11	1	1		
4	10	-1	1		
5	01	1	-1		
6	10	-1	1		

Table 8.7. Coherent receiver decision results at 4 dB SNR.

Symbol	Transmitted dibit	z_I	z_Q	Detected dibit	Correct / Error
1	00				
2	01				
3	11				
4	10				
5	01				
6	10				

Table 8.8. Effect of common carrier-phase error.

Phase error	Ideal quadrant condition	Measured bit errors	Measured symbol errors	Interpretation
0°				
20°				
40°				
50°				

Table 8.9. BPSK-QPSK comparison for the same bit rate.

Criterion	BPSK	QPSK
Bits per symbol		
Symbol rate		
Symbol duration		
Number of constellation points		
Adjacent phase separation		
Receiver branches		
Qualitative bandwidth observation		

Questions

1. Why is QPSK called a quadrature modulation technique?
2. Why must the number of serial input bits be even for the direct dibit implementation?
3. How does the serial-to-parallel converter reduce the signaling rate in each branch?
4. Explain the phase produced by the dibit 10 using the I and Q branch signs.
5. Why are adjacent QPSK constellation points separated by 90°?
6. Why do all ideal QPSK symbols have the same amplitude?
7. How are the I and Q bits recovered independently in the coherent receiver?
8. Why must the transmitter and receiver use the same dibit order and mapping?

9. Explain how a common phase error rotates the received constellation.
10. Why is 45° the ideal decision-boundary limit for the mapping used in this experiment?
11. Why can additive noise cause errors even when the phase error is less than 45° ?
12. Compare the QPSK symbol rate with the BPSK symbol rate for the same input bit rate.
13. What is the role of the parallel-to-serial converter at the receiver?
14. Comment on any differences between the theoretical phase sequence and the MATLAB waveform.

Experiment 9: Optimum Baseband Receiver, Matched Filter, Correlator and BER**1. Objective of the Experiment**

The objective of this experiment is to investigate the reception of binary digital signals in the presence of additive white Gaussian noise (AWGN). Students will construct an integrate-and-dump receiver, derive and implement a matched filter for a rectangular signaling pulse, implement the equivalent correlation receiver, and verify that the matched-filter and correlator decision samples are identical when they use the same signal template and sampling instant.

The experiment also introduces the statistical decision process used in digital communication receivers. Students will observe how noise changes the sampled decision variable, apply a zero threshold to antipodal binary signals, estimate the bit error rate by Monte Carlo simulation, and compare the obtained BPSK error rate with the theoretical expression. Finally, the error-probability expressions for ASK, coherent and noncoherent FSK, BPSK, and QPSK presented in the lecture notes will be compared.

At the end of this experiment, students are expected to:

- Explain the purpose of the baseband receiver and the role of the decision threshold.
- Generate antipodal rectangular binary pulses and transmit them through an AWGN channel.
- Implement an integrate-and-dump receiver and identify the optimum sampling instants.
- Derive the matched-filter impulse response by time reversing and delaying the signal pulse.
- Implement matched filtering using convolution in MATLAB.
- Implement a correlation receiver using multiplication and integration over each bit interval.
- Verify the equivalence of matched-filter and correlator samples.
- Interpret the Gaussian distributions of the receiver decision variable.
- Estimate BPSK BER by Monte Carlo simulation and compare it with the theoretical result.
- Compare the lecture-note error-probability expressions of ASK, FSK, BPSK, and QPSK.

2. Theoretical Background**2.1. Binary Baseband Reception in Noise**

The lecture notes consider a binary encoded baseband signal represented by two voltage levels. During one bit interval T , one binary symbol is represented by a positive waveform and the other by a negative waveform. The received signal is the sum of the transmitted signal and channel noise.

$$r(t) = s_i(t) + n(t), \quad i \in \{0,1\}, \quad 0 \leq t \leq T$$

For antipodal rectangular signaling, the two possible waveforms may be expressed as

$$s_1(t) = +A p(t), \quad s_0(t) = -A p(t)$$

where $p(t)$ is a rectangular pulse limited to one bit interval. The AWGN process $n(t)$ has zero mean and a two-sided power spectral density $N_0/2$. The receiver does not need to preserve every detail of the noisy waveform; it must produce a reliable binary decision at the end of each bit interval.

Binary Baseband Receiver Structures

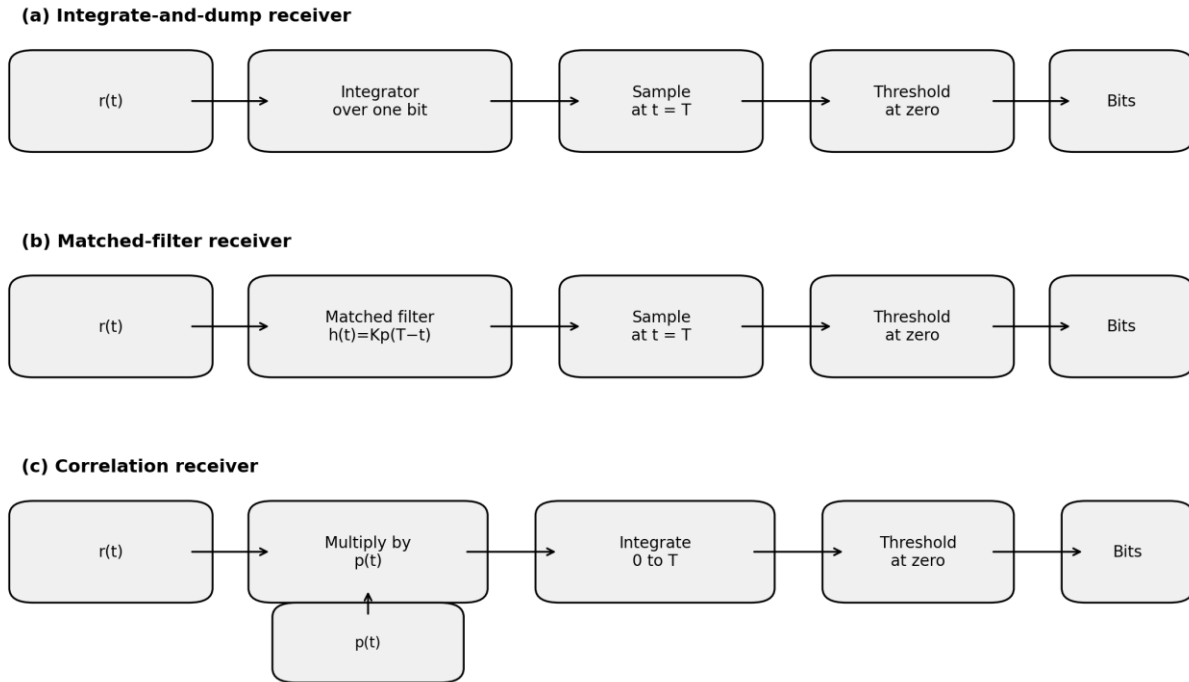


Fig. 9.1. Integrate-and-dump, matched-filter, and correlation receiver structures.

2.2. Integrate-and-Dump Receiver

For a rectangular baseband pulse, an integrate-and-dump receiver accumulates the received waveform during one bit interval and samples the accumulated value at the end of that interval. The integrator is then reset before the next bit. When the binary symbols are antipodal and equally likely, a zero threshold is used.

$$z_k = \int_{kT}^{(k+1)T} r(t)p(t-kT)dt$$

If z_k is positive, the receiver decides in favor of the positive signal; if z_k is negative, it decides in favor of the negative signal. The integration emphasizes the signal contribution while averaging the zero-mean noise over the bit interval.

2.3. Optimum Filter Criterion

The optimum linear receiver filter is selected to maximize the output signal-to-noise ratio at the required sampling time. In the white-noise case, the lecture notes show that the optimum filter is the matched filter. Its frequency response is proportional to the complex conjugate of the signal spectrum with a delay term.

$$H(f) = K P^*(f) e^{-j2\pi fT}$$

The corresponding impulse response is a time-reversed and delayed replica of the signal pulse.

$$h(t) = K p(T-t)$$

For a real rectangular pulse, time reversal does not change the pulse shape, but the delay establishes the correct sampling time. The matched-filter output reaches its maximum signal value at the end of the bit interval.

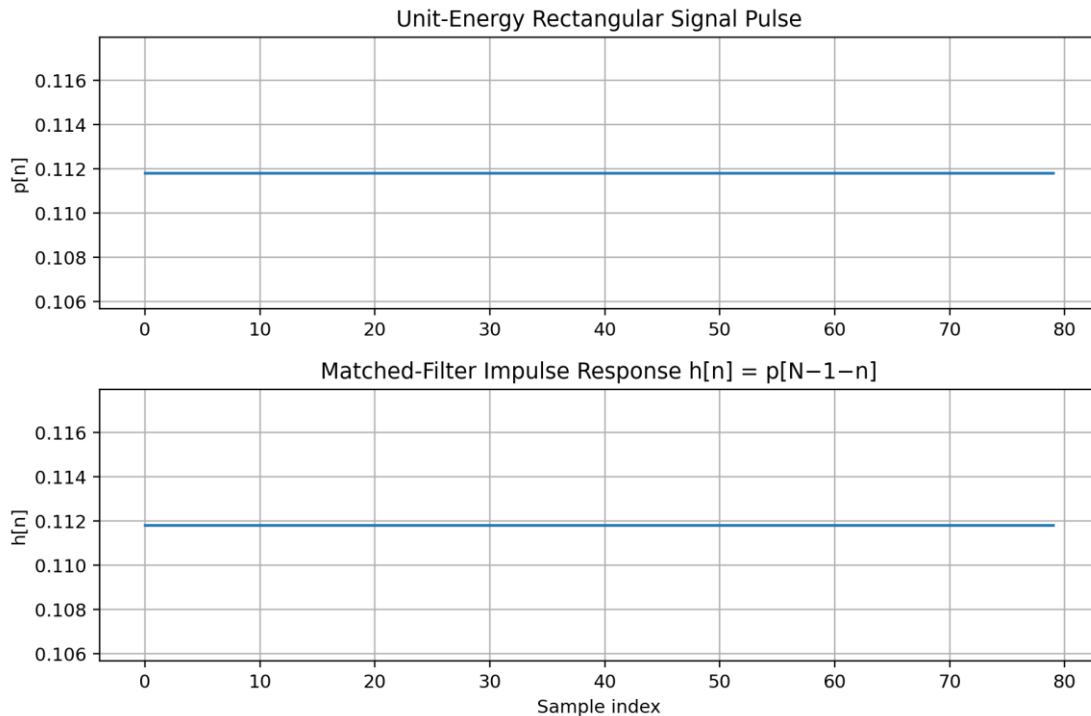


Fig. 9.2. Rectangular signal pulse and its matched-filter impulse response.

2.4. Matched-Filter and Correlator Equivalence

A correlation receiver multiplies the received signal by the known signal template and integrates the product over the symbol interval. The lecture notes demonstrate that the sampled output of this correlator is identical to the output of the matched filter at the optimum sampling time. They are therefore two realizations of the same optimum receiver.

$$z_{corr}(T) = \int[0 \text{ to } T] r(t)p(t)dt$$

$$z_{MF}(T) = (r*h)(T) = \int[0 \text{ to } T] r(\tau)h(T-\tau)d\tau$$

Substituting $h(T-\tau)=Kp(\tau)$ gives the same decision statistic, apart from the constant K . A positive scale factor does not change a zero-threshold decision.

2.5. Decision Variable and Probability of Error

At the sampling instant, the receiver output contains a deterministic signal contribution and a random Gaussian noise contribution. For equally likely antipodal signals, the optimum decision boundary is the midpoint between the two mean values, which is zero. An error occurs when noise moves the decision variable across this boundary.

$$\text{Decide 1 if } z \geq 0; \quad \text{decide 0 if } z < 0$$

The complementary error function and the Q-function are related by

$$Q(x) = 1/2 \operatorname{erfc}(x/\sqrt{2})$$

For coherent binary PSK, the lecture notes give

$$P_{b,BPSK} = 1/2 \operatorname{erfc}(\sqrt{E_b/N_0}) = Q(\sqrt{2E_b/N_0})$$

As E_b/N_0 increases, the two Gaussian decision distributions become more separated relative to their standard deviation and the bit error probability decreases rapidly.

2.6. Error-Probability Expressions in the Lecture Notes

The following expressions are used in this experiment to compare the receiver performance discussed in Unit II. The energy symbol follows the convention used in each lecture-note derivation. In particular, the ASK expression uses E as the energy of the transmitted carrier burst for binary 1.

Table 9.1. Error-probability expressions used in the experiment.

Scheme and receiver	Error-probability expression
Coherent ASK with matched filter	$P_e = 1/2 \operatorname{erfc}(\sqrt{E/(4N_0)})$
Coherent orthogonal BFSK	$P_b = 1/2 \operatorname{erfc}(\sqrt{E_b/(2N_0)})$
Noncoherent orthogonal BFSK	$P_b = 1/2 \exp(-E_b/(2N_0))$
Coherent BPSK	$P_b = 1/2 \operatorname{erfc}(\sqrt{E_b/N_0})$
Gray-coded coherent QPSK	$\text{BER} = 1/2 \operatorname{erfc}(\sqrt{E_b/N_0})$

The lecture notes also describe coherent QPSK as two coherent BPSK systems operating in parallel on orthogonal carriers. With Gray coding, its average bit error probability is therefore the same as coherent BPSK for the same E_b/N_0 , while two bits are transmitted per symbol.

2.7. Monte Carlo BER Estimation

In a Monte Carlo simulation, a large random bit sequence is generated, transmitted through a noise model, detected, and compared with the original sequence. The estimated BER is

$$\text{BER}_{est} = N_{error} / N_{total}$$

A sufficiently large number of transmitted bits is required to estimate small BER values. If no errors are observed, the simulation does not prove that the theoretical BER is zero; it only shows that the chosen simulation length was insufficient to observe a rarer event.

3. Materials Used

- Computer
- MATLAB software
- MATLAB Script Editor
- Digital Communications lecture notes
- Laboratory worksheet
- Calculator

The principal MATLAB commands and operations used in this experiment are:

- rand
- randn
- repelem
- conv
- fliplr
- cumsum
- dot or sum
- erfc
- semilogy

- histogram
- mean
- subplot
- plot
- stem
- grid
- legend

4. Preliminary (PreLab)

Complete the following tasks before attending the laboratory. Use the discrete-time normalization specified below unless otherwise stated.

4.1. Signaling Parameters

Use the following parameters:

Table 9.2. Parameters for the preliminary calculations.

Parameter	Value
Bit rate R_b	1000 bit/s
Bit duration T_b	1 ms
Samples per bit N_s	80
Sampling frequency F_s	80 kHz
Pulse $p[n]$	$1/\sqrt{N_s}$ s for $0 \leq n < N_s$
Pulse energy	$\sum p^2[n] = 1$
Decision threshold	0

1. Calculate the bit duration and verify the sampling frequency.
2. Write the discrete-time matched-filter impulse response $h[n]$ for the rectangular pulse.
3. Calculate the ideal correlator output for transmitted symbols +1 and -1.
4. Explain why zero is the optimum threshold for equally likely antipodal symbols.
5. Sketch the expected matched-filter output for one positive rectangular pulse.
6. Calculate the theoretical BPSK BER at $E_b/N_0 = 0, 2, 4, 6, 8,$ and 10 dB.
7. Using the formulas in Table 9.1, calculate the corresponding coherent ASK and coherent BFSK error probabilities.
8. Explain why the matched-filter and correlator decisions must be identical when both are implemented correctly.

Table 9.3. PreLab theoretical error probabilities.

Ratio (dB)	Coherent ASK	Coherent BFSK	BPSK	Gray-coded QPSK BER
0				
2				
4				
6				
8				
10				

5. Procedure

Part 1: Generate the Binary Baseband Signal

1. Open MATLAB and create a new script.
2. Define $R_b = 1000$ bit/s, $N_s = 80$ samples per bit, and $F_s = R_b \times N_s$.
3. Use the bit sequence [1 0 1 1 0 0 1 0 1 1 1 0].
4. Map logic 1 to +1 and logic 0 to -1.
5. Create the unit-energy rectangular pulse $p[n] = 1/\sqrt{N_s}$.
6. Construct the transmit waveform by multiplying each polar symbol by $p[n]$.
7. Plot the first six bits and the corresponding clean baseband waveform.

```
clear; close all; clc;
Rb = 1000;
Tb = 1/Rb;
Ns = 80;
Fs = Rb*Ns;
bits = [1 0 1 1 0 0 1 0 1 1 1 0];
a = 2*bits - 1;
p = ones(1,Ns)/sqrt(Ns);
s = kron(a,p);
t = (0:length(s)-1)/Fs;

figure;
subplot(2,1,1); stairs((0:length(bits))*Tb,[bits bits(end)]); grid on;
ylim([-0.2 1.2]); ylabel('Bit');
subplot(2,1,2); plot(t,s); grid on;
xlabel('Time (s)'); ylabel('s(t)');
```

Part 2: Add AWGN and Observe the Received Waveform

1. Set E_b/N_0 to 4 dB and convert it to a linear ratio.
2. Because the discrete pulse has unit energy, set $N_0 = 1/(E_b/N_0)$.
3. Generate real Gaussian noise with variance $N_0/2$ per sample.
4. Add the noise to the transmit waveform.
5. Plot the clean and noisy waveforms for the first six bit intervals.
6. Comment on why individual noisy samples may be much larger than the pulse amplitude although the receiver can still make reliable decisions after integration.

```
EbN0dB = 4;
gamma = 10^(EbN0dB/10);
N0 = 1/gamma;
noise = sqrt(N0/2)*randn(size(s));
r = s + noise;

figure;
subplot(2,1,1); plot(t,s); grid on; xlim([0 6*Tb]); ylabel('s(t)');
subplot(2,1,2); plot(t,r); grid on; xlim([0 6*Tb]); ylabel('r(t)'); xlabel('Time (s)');
```

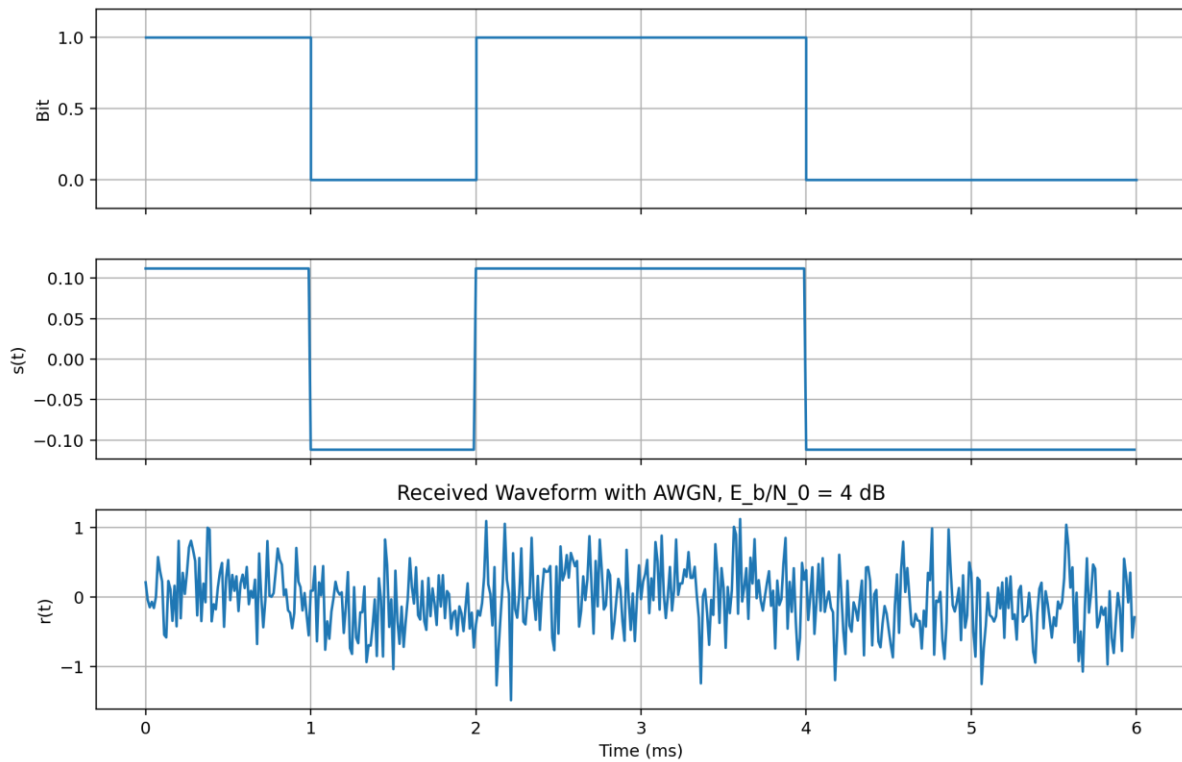


Fig. 9.3. Binary data, clean antipodal waveform, and noisy received waveform.

Part 3: Integrate-and-Dump Reception

1. Process each bit interval independently.
2. Multiply the received samples by $p[n]$ and calculate the cumulative sum inside the bit interval.
3. Record the final value of each interval as the decision statistic.
4. Apply the zero threshold and convert the decisions back to bits.
5. Compare the detected bits with the transmitted sequence.
6. Plot the integrator output and the sampled decision values.

```
Nbits = length(bits);
zID = zeros(1,Nbits);
yID = zeros(size(r));
for k = 1:Nbits
    idx = (k-1)*Ns + (1:Ns);
    yID(idx) = cumsum(r(idx).*p);
    zID(k) = yID(idx(end));
end
bitsID = zID >= 0;
errorsID = sum(bitsID ~= bits);

figure;
subplot(2,1,1); plot(t,yID); grid on; ylabel('Integrator output');
subplot(2,1,2); stem(0:Nbits-1,zID,'filled'); yline(0); grid on;
xlabel('Bit index'); ylabel('Decision statistic');
```

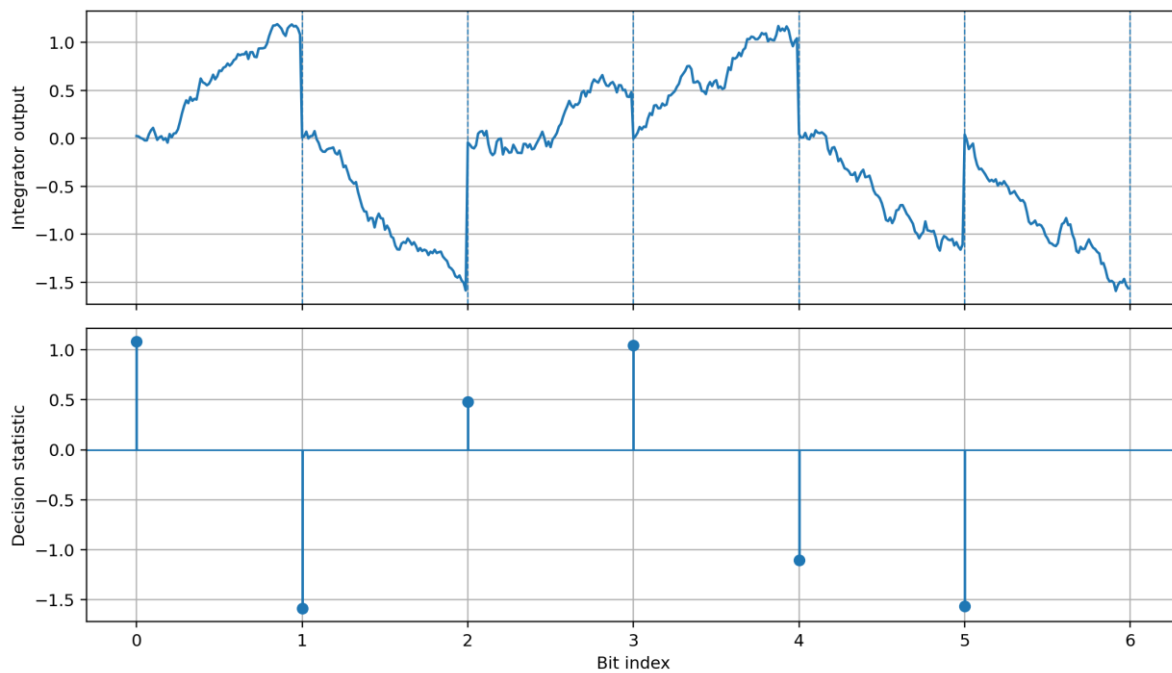


Fig. 9.4. Integrate-and-dump output and decision samples.

Part 4: Matched-Filter Receiver

1. Construct $h[n]$ by reversing $p[n]$.
2. Convolve the received waveform with the matched-filter impulse response.
3. Determine the end-of-bit sample indices, accounting for the filter delay.
4. Extract the matched-filter decision samples.
5. Apply a zero threshold and determine the detected bit sequence.
6. Plot the matched-filter output and mark the sampling instants.

```
h = flip1r(p);
yMF = conv(r,h);
sampleIndex = (1:Nbits)*Ns;
zMF = yMF(sampleIndex);
bitsMF = zMF >= 0;
errorsMF = sum(bitsMF ~= bits);

tMF = (0:length(yMF)-1)/Fs;
figure; plot(tMF,yMF); hold on;
plot(tMF(sampleIndex),zMF,'o'); yline(0); grid on;
xlabel('Time (s)'); ylabel('Matched-filter output');
```

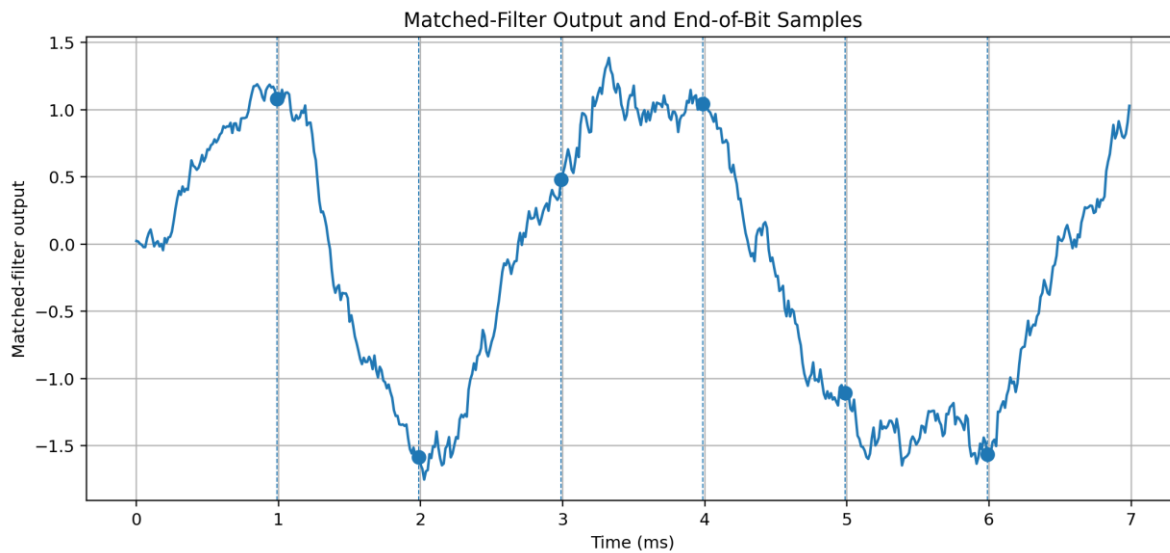


Fig. 9.5. Matched-filter output and optimum end-of-bit samples.

Part 5: Correlation Receiver and Equivalence Test

1. For each bit interval, multiply the received segment by $p[n]$ and sum the result.
2. Compare the obtained correlator samples with the matched-filter samples.
3. Calculate the maximum absolute difference between the two sample vectors.
4. Verify that the detected bits and error counts are identical.
5. Plot both sample vectors on the same axes.

```
zCorr = zeros(1,Nbits);
for k = 1:Nbits
    idx = (k-1)*Ns + (1:Ns);
    zCorr(k) = sum(r(idx).*p);
end
bitsCorr = zCorr >= 0;
maxDifference = max(abs(zMF-zCorr));
fprintf('Maximum difference = %.3e\n',maxDifference);

figure;
plot(0:Nbits-1,zCorr,'o-'); hold on;
plot(0:Nbits-1,zMF,'x--'); yline(0); grid on;
legend('Correlator','Matched filter'); xlabel('Bit index'); ylabel('Decision statistic');
```

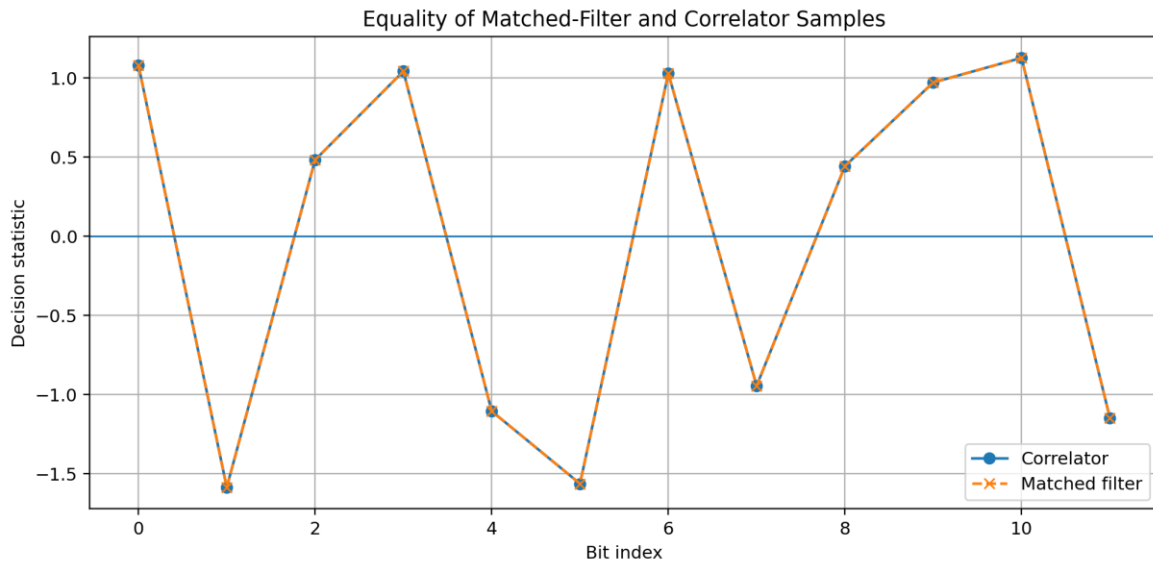



Fig. 9.6. Comparison of matched-filter and correlator decision samples.

Part 6: Decision Distributions

1. Generate at least 6000 equally likely antipodal symbols.
2. Use $E_b/N_0 = 2$ dB.
3. Generate the corresponding noisy decision variables directly as $z=a+n$, where $a \in \{-1, +1\}$.
4. Plot separate normalized histograms for symbols -1 and $+1$.
5. Mark the zero decision threshold.
6. Explain the relationship between the overlapping region and the error probability.

```
M = 6000;
b = randi([0 1],1,M);
a = 2*b - 1;
gamma = 10^(2/10);
z = a + sqrt(1/(2*gamma))*randn(1,M);

figure;
histogram(z(a== -1),55,'Normalization','pdf'); hold on;
histogram(z(a==1),55,'Normalization','pdf');
xline(0); grid on; xlabel('Decision variable z'); ylabel('Estimated density');
legend('Symbol -1','Symbol +1');
```

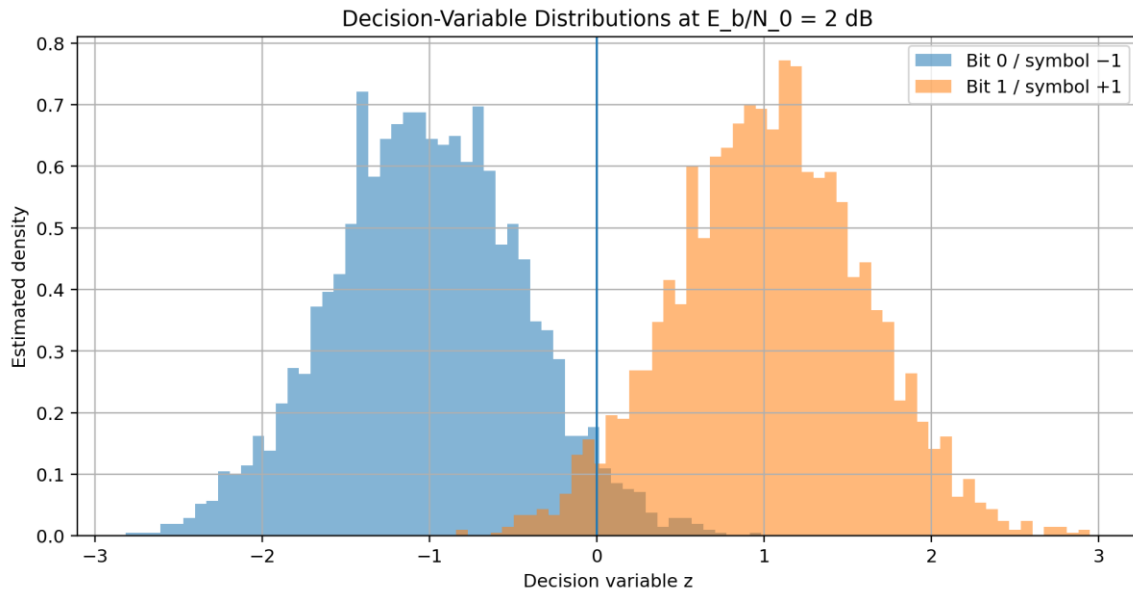


Fig. 9.7. Gaussian decision-variable distributions and the zero threshold.

Part 7: BPSK BER Simulation

1. Generate 250000 random bits.
2. Evaluate E_b/N_0 from 0 to 10 dB.
3. For each value, generate the noisy BPSK decision variable.
4. Apply a zero threshold and calculate the simulated BER.
5. Calculate the theoretical BER using $0.5\text{erfc}(\sqrt{E_b/N_0})$.
6. Plot simulated and theoretical results on a semilogarithmic graph.
7. Explain why the simulated curve becomes irregular when only a small number of errors are observed.

```

EbNdB = 0:10;
N = 250000;
b = randi([0 1],1,N);
a = 2*b - 1;
BERsim = zeros(size(EbNdB));
for m = 1:length(EbNdB)
    gamma = 10^(EbNdB(m)/10);
    z = a + sqrt(1/(2*gamma))*randn(size(a));
    bhat = z >= 0;
    BERsim(m) = mean(bhat ~= b);
end
BERtheory = 0.5*erfc(sqrt(10.^(EbNdB/10)));
figure; semilogy(EbNdB,BERtheory,'-'); hold on;
semilogy(EbNdB,BERSim,'o'); grid on;
xlabel('E_b/N_0 (dB)'); ylabel('BER'); legend('Theory','Simulation');

```

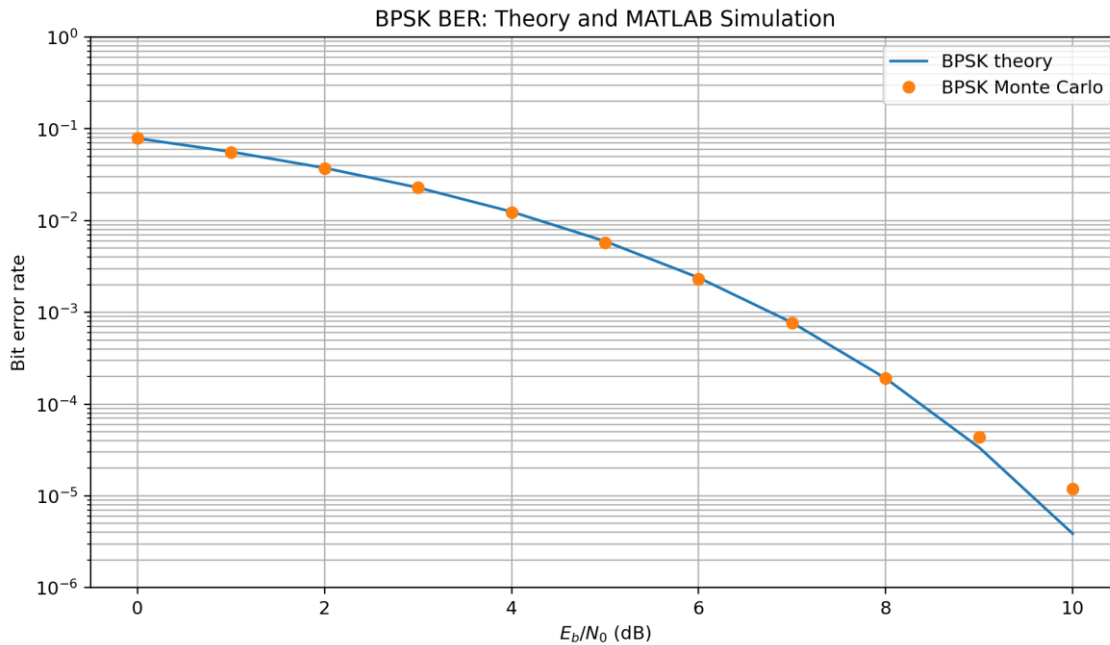


Fig. 9.8. Theoretical and simulated coherent BPSK BER.

Part 8: Compare Lecture-Note Error-Probability Expressions

1. Create an energy-to-noise ratio vector from 0 to 15 dB.
2. Calculate the coherent ASK, coherent BFSK, noncoherent BFSK, BPSK, and QPSK BER expressions given in Table 9.1.
3. Plot all curves using semilogy.
4. Identify which curves coincide and explain why.
5. For a target error probability of 10^{-3} , estimate the required ratio for each curve from the graph.
6. When interpreting ASK, retain the energy convention stated in Section 2.6.

```

rati dB = 0:15;
g = 10.^(rati dB/10);
PeASK = 0.5*erfc(sqrt(g/4));
PeCFSK = 0.5*erfc(sqrt(g/2));
PeNCFSK = 0.5*exp(-g/2);
PeBPSK = 0.5*erfc(sqrt(g));
PeQPSK = PeBPSK;

figure;
semilogy(rati dB,PeASK,rati dB,PeCFSK,rati dB,PeNCFSK, ...
         rati dB,PeBPSK,rati dB,PeQPSK,'--');
grid on; xlabel('Energy-to-noise ratio (dB)'); ylabel('Error probability');
legend('ASK','Coherent FSK','Noncoherent FSK','BPSK','QPSK BER');

```

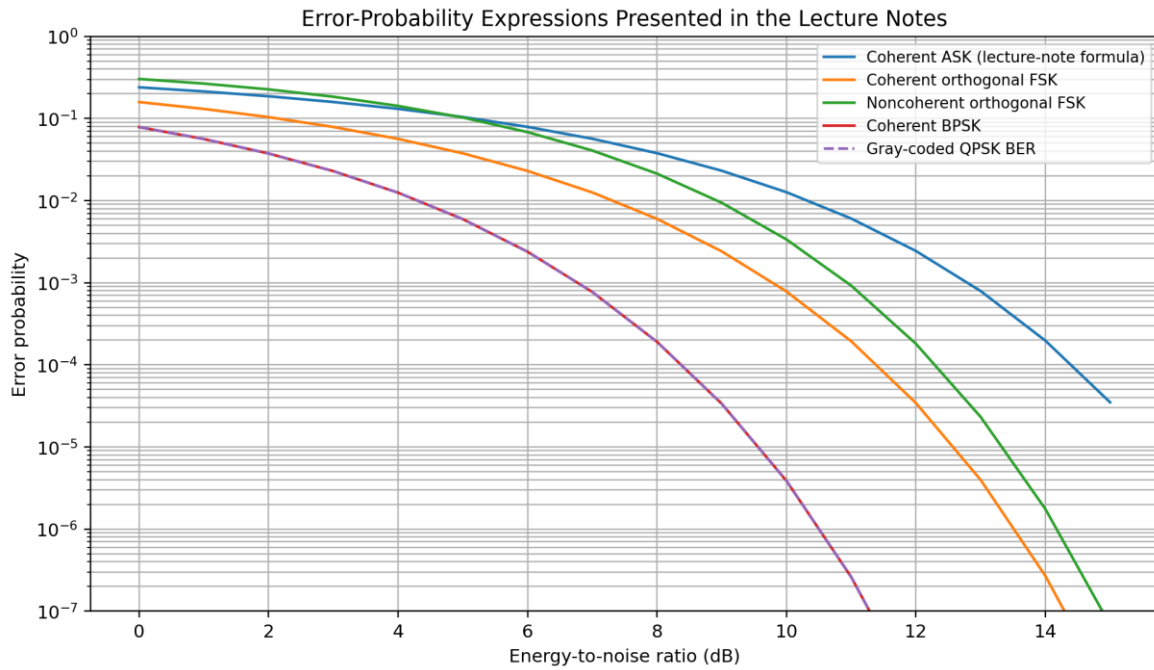


Fig. 9.9. Comparison of the error-probability expressions presented in the lecture notes.

6. Evaluation of Results

Record the calculated and MATLAB-obtained values in the following tables. Include the requested plots in the laboratory report.

Table 9.4. Receiver parameter verification.

Quantity	Theoretical value	MATLAB result	Percentage error
Bit duration T_b			
Sampling frequency F_s			
Pulse energy			
Ideal positive decision sample			
Ideal negative decision sample			

Table 9.5. Decision results for the specified bit sequence.

Bit index	Transmitted bit	Integrator sample	Matched-filter sample	Correlator sample	Detected bit
1	1				
2	0				
3	1				
4	1				
5	0				
6	0				
7	1				
8	0				
9	1				
10	1				
11	1				
12	0				

Table 9.6. Matched-filter and correlator equivalence.

Criterion	Result
Maximum $ z_{MF} - z_{corr} $	
Matched-filter bit errors	
Correlator bit errors	
Are detected sequences identical?	
Explanation of any numerical difference	

Table 9.7. BPSK theoretical and simulated BER.

E b/N ₀ (dB)	Theoretical BER	Simulated BER	Number of errors
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

Table 9.8. Error-probability comparison.

Ratio (dB)	ASK	Coherent BFSK	Noncoherent BFSK	BPSK	QPSK BER
0					
3					
6					
9					
12					
15					

Table 9.9. Approximate ratio required for P_e or BER = 10^{-3} .

Technique	Required ratio from graph	Comment
Coherent ASK		
Coherent BFSK		
Noncoherent BFSK		
BPSK		
Gray-coded QPSK BER		

Questions

1. What information must a baseband receiver recover from the noisy waveform?
2. Why is an integrate-and-dump receiver suitable for a rectangular baseband pulse?
3. What is the optimum sampling time for the matched-filter output?
4. How is the matched-filter impulse response obtained from the transmitted pulse?
5. Why does the matched filter maximize output SNR in white noise?

6. Why are matched-filter and correlator samples identical?
7. Why does a positive scaling constant in the matched filter not change the zero-threshold decision?
8. Explain why the decision variable is Gaussian when the channel noise is Gaussian.
9. What part of the histogram corresponds to decision errors?
10. How does increasing E_b/N_0 change the two decision distributions?
11. Why must BER simulations use more bits at high E_b/N_0 ?
12. Why do the BPSK and Gray-coded QPSK BER curves coincide?
13. Compare coherent and noncoherent BFSK based on the plotted curves.
14. Why must the energy convention be stated explicitly when comparing ASK with other schemes?
15. Comment on any difference between the theoretical and simulated BPSK BER.

Experiment 10: Information Theory, Channel Capacity and Source Coding**1. Objective of the Experiment**

The objective of this experiment is to introduce the quantitative measures used to describe discrete information sources and communication channels. Students will calculate the self-information of source symbols, entropy, information rate, joint entropy, conditional entropy, and mutual information using MATLAB. The experiment also examines the Shannon-Hartley relation and the tradeoff between channel bandwidth and signal-to-noise ratio.

The second part of the experiment focuses on source coding. Students will implement the Shannon-Fano and binary Huffman procedures described in the lecture notes, construct prefix code tables, calculate average codeword length, and compare coded representations with fixed-length binary coding. The relationship between source entropy and average codeword length will be evaluated using two probability distributions taken from the course material.

At the end of this experiment, students are expected to:

- Determine the self-information associated with source symbols of different probabilities.
- Calculate the entropy and information rate of a discrete memoryless source.
- Explain the maximum-entropy condition for a binary source.
- Calculate joint entropy, conditional entropy, and mutual information from a joint probability table.
- Interpret mutual information as the reduction in uncertainty produced by knowledge of another random variable.
- Calculate channel capacity using the Shannon-Hartley equation.
- Examine the bandwidth-SNR tradeoff for a noisy channel.
- Construct Shannon-Fano and Huffman prefix codes.
- Calculate average codeword length, coding efficiency, and redundancy.
- Encode and decode a random source sequence using the generated prefix codes.

2. Theoretical Background**2.1. Discrete Memoryless Source and Self-Information**

The lecture notes describe a discrete memoryless source as a source that generates symbols from a finite alphabet with a fixed probability assignment. The present symbol is statistically independent of previous source symbols. If a symbol x_i occurs with probability p_i , its self-information is defined as

$$I(x_i) = -\log_2(p_i)$$

A deterministic event with probability one conveys zero information. Less probable events convey more information than more probable events. The logarithm base determines the unit; base two gives bits, the natural logarithm gives nats, and base ten gives Hartleys.

Information-Theory and Source-Coding Analysis Chain

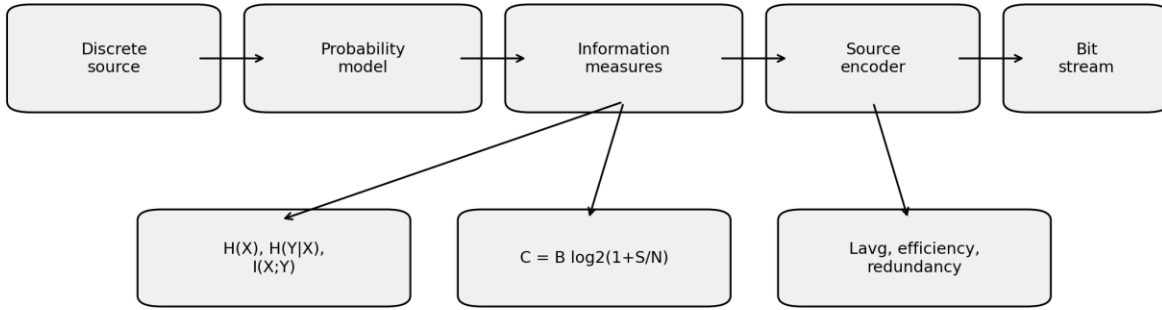


Fig. 10.1. Information-theory and source-coding analysis chain used in the experiment.

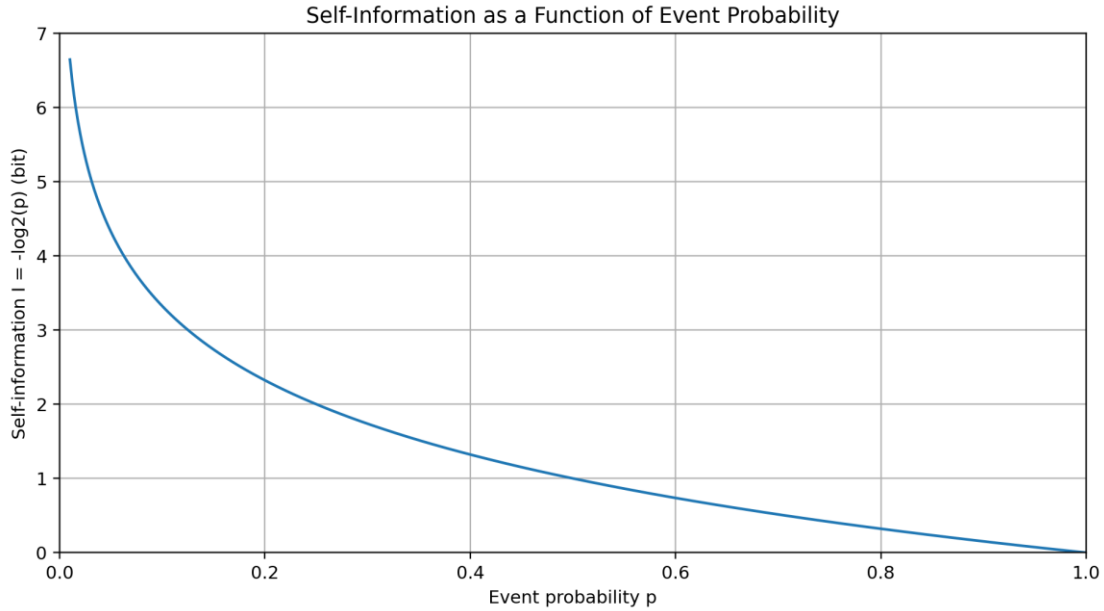


Fig. 10.2. Relationship between event probability and self-information.

2.2. Entropy and Information Rate

Entropy is the average information generated by a discrete memoryless source. For a source X with M symbols, it is

$$H(X) = -\sum_{i=1}^M p_i \log_2(p_i) \quad \text{bit/symbol}$$

The entropy is nonnegative and is bounded by

$$0 \leq H(X) \leq \log_2(M)$$

The maximum entropy is obtained when all M symbols are equally probable. For a binary source with probabilities p and $1-p$,

$$H_b(p) = -p \log_2(p) - (1-p) \log_2(1-p)$$

The binary entropy is zero when one outcome is certain and reaches one bit per symbol when the two outcomes are equally likely. If the source generates r_s symbols per second, the information rate is

$$R = r_s H(X) \text{ bit/s}$$

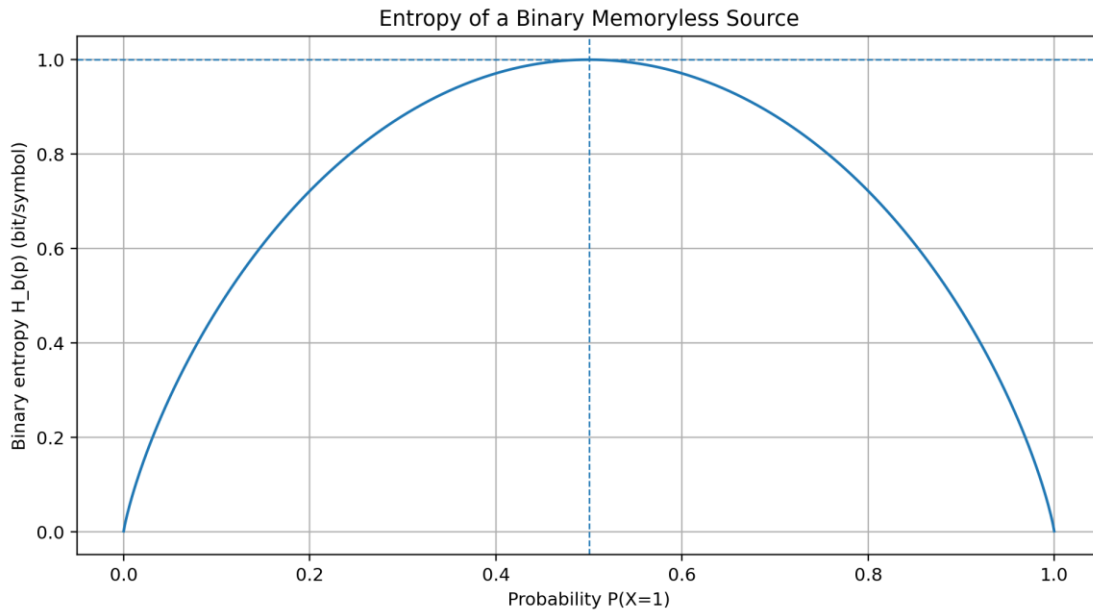


Fig. 10.3. Entropy of a binary memoryless source.

2.3. Joint, Conditional, and Mutual Information

For two discrete random variables X and Y with joint probability $p(x,y)$, the joint entropy is

$$H(X,Y) = -\sum_x \sum_y p(x,y) \log_2 p(x,y)$$

The conditional entropy measures the remaining uncertainty of one random variable after the other is known. The chain rule gives

$$H(X,Y) = H(X) + H(Y|X) = H(Y) + H(X|Y)$$

The lecture notes define mutual information as the reduction in uncertainty of one variable due to knowledge of the other variable.

$$I(X;Y) = H(Y) - H(Y|X)$$

$$I(X;Y) = H(X) + H(Y) - H(X,Y) = I(Y;X)$$

Mutual information is zero when the variables are independent and increases as their statistical dependence increases.

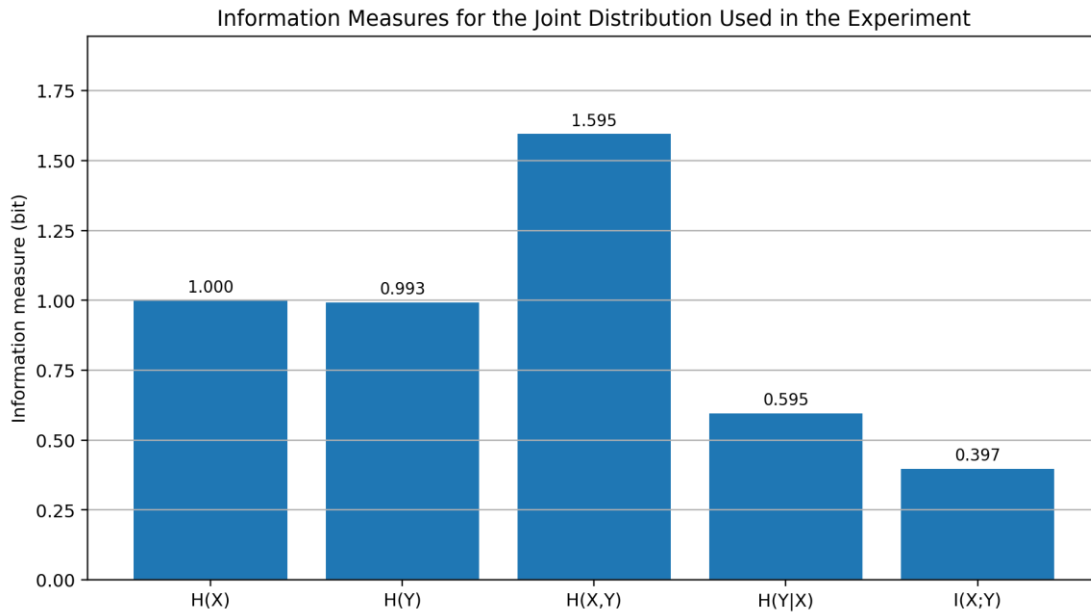


Fig. 10.4. Entropy and mutual-information values for the joint distribution used in the experiment.

2.4. Shannon-Hartley Channel Capacity

The Shannon-Hartley relation gives the upper bound on the information rate that can be transmitted through a bandwidth-limited channel in the presence of Gaussian noise. The channel capacity is

$$C = B \log_2(1 + S/N) \quad \text{bit/s}$$

where B is the channel bandwidth in hertz and S/N is the signal-to-noise power ratio in linear form. The lecture notes emphasize that an increase in bandwidth or SNR increases capacity and that bandwidth can be traded against SNR while maintaining a required information rate. The SNR value in decibels must be converted to a linear ratio before it is used in the equation.

$$SNR_{linear} = 10^{(SNR_{dB}/10)}$$

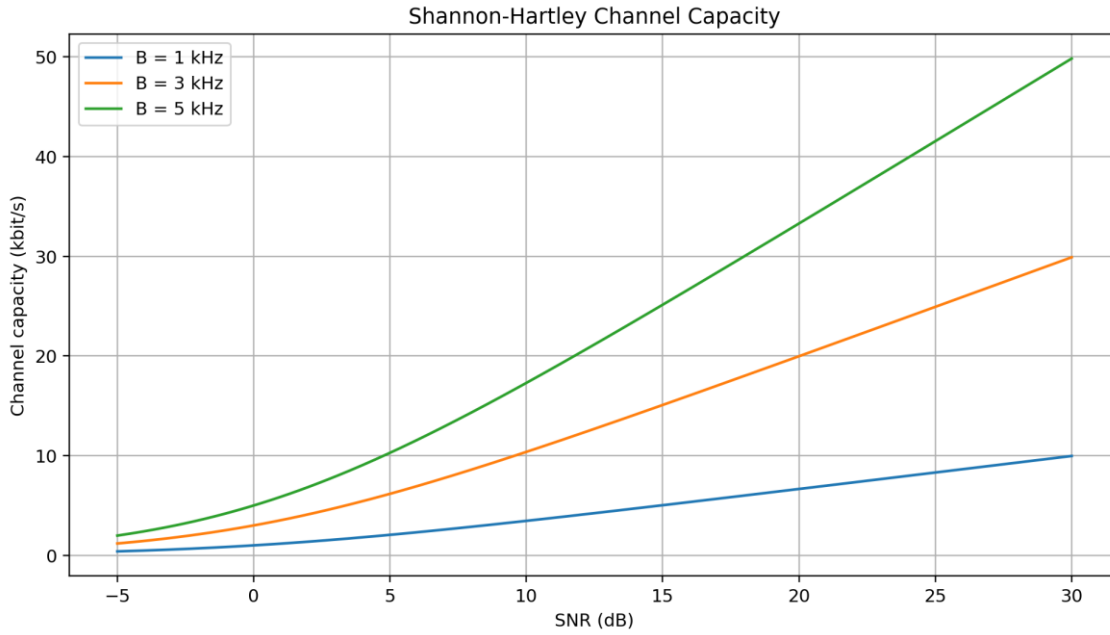


Fig. 10.5. Shannon-Hartley channel-capacity curves for different bandwidths.

2.5. Source Coding and Prefix Condition

The purpose of source coding is to represent source symbols efficiently and reduce the average number of transmitted bits. More probable symbols are generally assigned shorter codewords, while less probable symbols are assigned longer codewords. The code must also permit unambiguous decoding. A prefix code satisfies this requirement because no valid codeword is the prefix of another valid codeword.

The average codeword length is

$$L_{avg} = \sum_{i=1}^M p_i l_i \text{ bit/symbol}$$

where l_i is the length of the codeword assigned to symbol x_i . In this experiment, coding efficiency and redundancy are evaluated by

$$\eta = H(X)/L_{avg}$$

$$\text{Redundancy} = 1 - \eta$$

A good source code has an average length close to the entropy while preserving unique decodability.

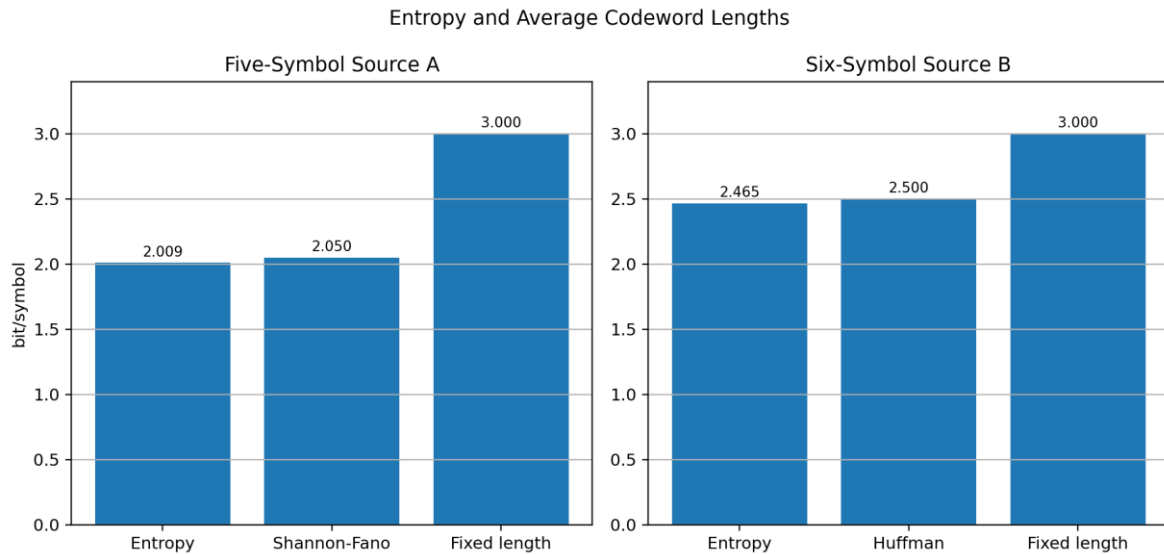


Fig. 10.6. Entropy and average lengths for the source-coding examples.

2.6. Shannon-Fano Coding

Shannon-Fano coding is a top-down procedure. The source symbols are first sorted from the largest probability to the smallest probability. The list is divided into two groups with total probabilities as close as possible. The first group is assigned 0 and the second group is assigned 1. The division is repeated recursively until each group contains one symbol. The lecture notes state that Shannon-Fano coding produces a prefix code but does not always produce the minimum possible average length.

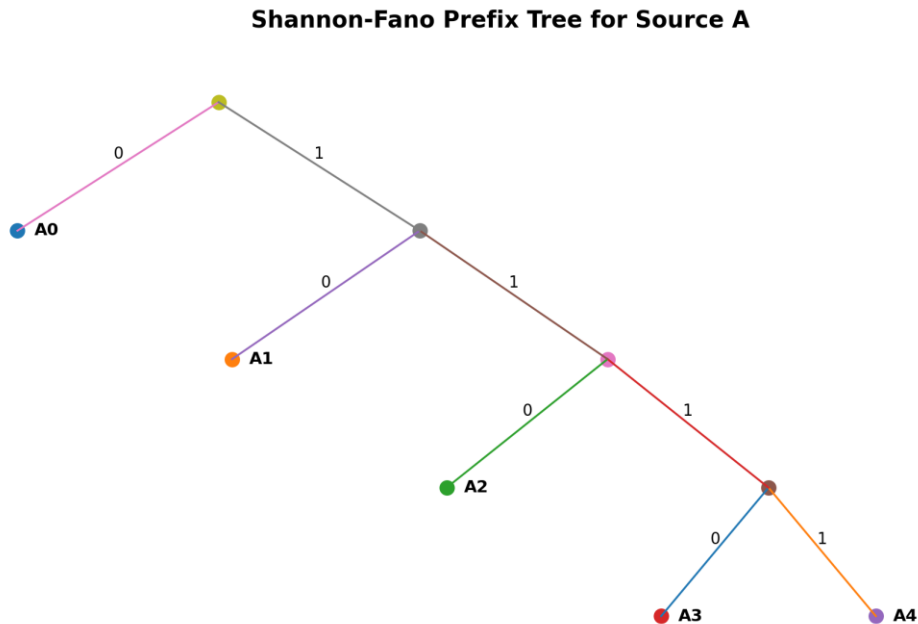


Fig. 10.7. Shannon-Fano prefix tree for the five-symbol source in the lecture notes.

2.7. Binary Huffman Coding

Binary Huffman coding is a bottom-up procedure. At each step, the two symbols or combined nodes with the smallest probabilities are merged. One branch is assigned 0 and the other branch is

assigned 1. The merging process continues until one root node remains. Reading the branches from the root to each source symbol gives the corresponding codeword. The lecture notes present Huffman coding as an optimum variable-length prefix-coding method.

Huffman Prefix Tree for Source B

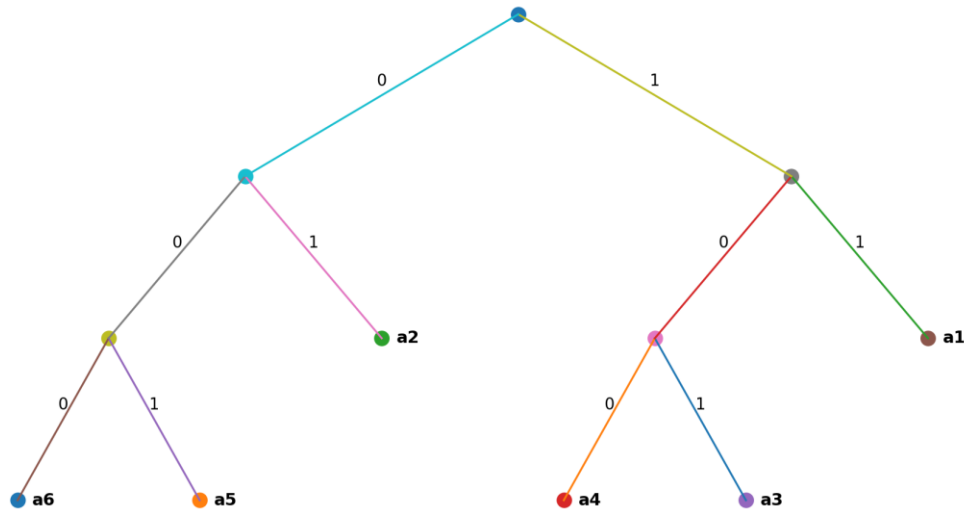


Fig. 10.8. Huffman prefix tree for the six-symbol source in the lecture notes.

3. Materials Used

- Computer
- MATLAB software
- MATLAB Script Editor
- Digital Communications lecture notes
- Laboratory worksheet
- Calculator

The principal MATLAB commands and operations used in this experiment are:

- log2
- sum
- cumsum
- sort
- cell
- struct
- bar
- plot
- stairs
- rand
- find
- strlength or length
- table

- disp

4. Preliminary (PreLab)

Complete the following calculations before attending the laboratory.

4.1. Source A: Information and Shannon-Fano Coding

Table 10.1. Probability distribution of Source A.

Symbol	A0	A1	A2	A3	A4
Probability	0.40	0.30	0.15	0.10	0.05

1. Calculate the self-information of each source symbol.
2. Calculate the source entropy $H(A)$.
3. Calculate the information rate when the source produces 1000 symbols per second.
4. Determine the number of bits required by fixed-length binary coding.
5. Construct the Shannon-Fano code using the top-down partitioning procedure.
6. Calculate the Shannon-Fano average length, efficiency, and redundancy.
7. Check the prefix condition for the obtained codewords.

Table 10.2. PreLab calculations for Source A.

Symbol	Probability	I(symbol) (bit)	Shannon-Fano code	Code length
A0	0.40			
A1	0.30			
A2	0.15			
A3	0.10			
A4	0.05			

4.2. Joint Probability Distribution

For the binary random variables X and Y, use the following joint probability table:

Table 10.3. Joint probability $p(x,y)$.

X \ Y	Y=0	Y=1	P(X=x)
X=0	0.45	0.05	
X=1	0.10	0.40	
P(Y=y)			1.00

1. Calculate the marginal probabilities $P(X=x)$ and $P(Y=y)$.
2. Calculate $H(X)$, $H(Y)$, and $H(X,Y)$.
3. Calculate $H(Y|X)$ and $H(X|Y)$.
4. Calculate $I(X;Y)$ using two equivalent expressions.
5. Determine whether X and Y are statistically independent.

4.3. Channel Capacity

1. Calculate the channel capacity for $B = 4$ kHz and $S/N = 7$.
2. Calculate the channel capacity for $B = 3$ kHz and $S/N = 15$.
3. Compare the two capacity values and explain the bandwidth-SNR tradeoff.

- Calculate the capacity for $B = 5$ kHz and $\text{SNR} = 10$ dB. Remember to convert dB to a linear ratio.

Table 10.4. PreLab channel-capacity calculations.

Bandwidth	SNR specification	SNR linear	Capacity (bit/s)
4 kHz	$S/N = 7$		
3 kHz	$S/N = 15$		
5 kHz	10 dB		

4.4. Source B: Huffman Coding

Table 10.5. Probability distribution of Source B.

Symbol	a1	a2	a3	a4	a5	a6
Probability	0.30	0.20	0.15	0.15	0.12	0.08

- Calculate the entropy of Source B.
- Construct the binary Huffman tree by repeatedly combining the two least probable nodes.
- Determine the codeword assigned to each source symbol.
- Calculate the average codeword length, coding efficiency, and redundancy.
- Compare the Huffman average length with the three-bit fixed-length representation.

Table 10.6. PreLab Huffman calculations for Source B.

Symbol	Probability	Huffman code	Code length
a1	0.30		
a2	0.20		
a3	0.15		
a4	0.15		
a5	0.12		
a6	0.08		

5. Procedure

Part 1: Self-Information, Entropy, and Information Rate

- Open MATLAB and create a new script file.
- Define the symbol probabilities of Source A.
- Check that the probabilities sum to one.
- Calculate the self-information of each symbol using $-\log_2(p)$.
- Calculate the source entropy.
- Calculate the information rate for 1000 symbols per second.
- Display the results in a MATLAB table.
- Plot self-information as a function of probability from 0.01 to 1.

```
clear; close all; clc;
symbolsA = ["A0" "A1" "A2" "A3" "A4"];
pA = [0.40 0.30 0.15 0.10 0.05];
assert(abs(sum(pA)-1) < 1e-12, 'Probabilities must sum to one.');
```

```
IA = -log2(pA);
HA = -sum(pA.*log2(pA));
rs = 1000;
```

```

informationRate = rs*HA;
T_A = table(symbolsA.',pA.',IA.', 'VariableNames', ...
    {'Symbol','Probability','SelfInformation_bit'});
disp(T_A);
fprintf('Entropy = %.6f bit/symbol\n',HA);
fprintf('Information rate = %.3f bit/s\n',informationRate);

p = linspace(0.001,1,500);
figure; plot(p,-log2(p)); grid on;
xlabel('Probability p'); ylabel('Self-information (bit)');
title('Self-Information');

```

Part 2: Binary Entropy

1. Create a probability vector from 0.001 to 0.999.
2. Calculate the binary entropy function.
3. Plot $H_b(p)$ versus p .
4. Determine the probability at which entropy is maximum.
5. Explain why the entropy approaches zero near $p=0$ and $p=1$.

```

p = linspace(0.001,0.999,1000);
Hb = -p.*log2(p) - (1-p).*log2(1-p);
figure; plot(p,Hb); grid on;
xlabel('P(X=1)'); ylabel('H_b(p) (bit/symbol)');
title('Binary Entropy Function');
[Hmax,idx] = max(Hb);
fprintf('Maximum entropy = %.6f at p = %.4f\n',Hmax,p(idx));

```

Part 3: Joint, Conditional, and Mutual Information

1. Define the joint probability matrix given in Table 10.3.
2. Calculate the marginal distributions of X and Y.
3. Calculate $H(X)$, $H(Y)$, and $H(X,Y)$.
4. Calculate $H(Y|X)$ and $H(X|Y)$ by using the chain rule.
5. Calculate $I(X;Y)$ using $H(X)+H(Y)-H(X,Y)$.
6. Verify the result using $H(Y)-H(Y|X)$.
7. Display all values and compare them with the PreLab calculations.

```

Pxy = [0.45 0.05; 0.10 0.40];
px = sum(Pxy,2);
py = sum(Pxy,1);
entropyBits = @(q) -sum(q(q>0).*log2(q(q>0)));
HX = entropyBits(px);
HY = entropyBits(py);
HXY = entropyBits(Pxy(:));
HY_X = HXY - HX;
HX_Y = HXY - HY;
Ixy = HX + HY - HXY;
Icheck = HY - HY_X;

fprintf('H(X) = %.6f bit\n',HX);
fprintf('H(Y) = %.6f bit\n',HY);
fprintf('H(X,Y) = %.6f bit\n',HXY);

```



```
fprintf('H(Y|X) = %.6f bit
',HY_X);
fprintf('H(X|Y) = %.6f bit
',HX_Y);
fprintf('I(X;Y) = %.6f bit, check = %.6f bit
',Ixy,Icheck);
```

Part 4: Shannon-Hartley Capacity and the Bandwidth-SNR Tradeoff

1. Create an SNR vector from -5 dB to 30 dB.
2. Convert the SNR values to linear form.
3. Calculate capacity for bandwidths of 1 kHz, 3 kHz, and 5 kHz.
4. Plot all three capacity curves in the same figure.
5. Calculate the capacity values specified in Table 10.4.
6. Compare the two cases $B=4$ kHz, $S/N=7$ and $B=3$ kHz, $S/N=15$.

```
snr_dB = -5:0.1:30;
snrLinear = 10.^(snr_dB/10);
Blist = [1000 3000 5000];
figure; hold on;
for B = Blist
    C = B*log2(1+snrLinear);
    plot(snr_dB,C/1000,'DisplayName',sprintf('B = %.0f kHz',B/1000));
end
grid on; legend('Location','northwest');
xlabel('SNR (dB)'); ylabel('Capacity (kbit/s)');
title('Shannon-Hartley Capacity');

C1 = 4000*log2(1+7);
C2 = 3000*log2(1+15);
C3 = 5000*log2(1+10^(10/10));
fprintf('C1 = %.2f bit/s, C2 = %.2f bit/s, C3 = %.2f bit/s
',C1,C2,C3);
```

Part 5: Shannon-Fano Coding

1. Use the Source A probabilities and call the shannonFano function given below.
2. Display each source symbol, probability, codeword, and codeword length.
3. Calculate the average codeword length.
4. Calculate coding efficiency and redundancy.
5. Compare the generated code with the code tree in Fig. 10.7.
6. Verify that no codeword is the prefix of another codeword.

```
codesSF = shannonFano(pA);
lengthsSF = cellfun(@length,codesSF);
Lsf = sum(pA.*lengthsSF);
etaSF = HA/Lsf;
redundancySF = 1-etaSF;
T_SF = table(symbolsA.',pA.',string(codesSF).',lengthsSF.', ...
'VariableNames',{'Symbol','Probability','Code','Length'});
disp(T_SF);
fprintf('Lavg = %.6f bit/symbol
',Lsf);
fprintf('Efficiency = %.4f %%
',100*etaSF);
fprintf('Redundancy = %.4f %%
',100*redundancySF);
```

Part 6: Huffman Coding

1. Define the Source B probabilities.
2. Call the binaryHuffman function given below.
3. Display the Huffman code table.
4. Calculate the entropy, average codeword length, efficiency, and redundancy.
5. Compare the generated codeword lengths with the tree shown in Fig. 10.8.
6. Compare Huffman coding with three-bit fixed-length coding.

```

symbolsB = ["a1" "a2" "a3" "a4" "a5" "a6"];
pB = [0.30 0.20 0.15 0.15 0.12 0.08];
HB = -sum(pB.*log2(pB));
codesH = binaryHuffman(pB);
lengthsH = cellfun(@length,codesH);
Lh = sum(pB.*lengthsH);
etaH = HB/Lh;
redundancyH = 1-etaH;
T_H = table(symbolsB.',pB.',string(codesH).',lengthsH.', ...
    'VariableNames',{'Symbol','Probability','Code','Length'});
disp(T_H);
fprintf('Entropy = %.6f, Lavg = %.6f bit/symbol
',HB,Lh);
fprintf('Efficiency = %.4f %%
',100*etaH);

```

Part 7: Encoding and Decoding a Random Source Sequence

1. Generate 10,000 Source A symbols according to the specified probability distribution.
2. Encode the sequence with the Shannon-Fano code.
3. Decode the bitstream using prefix matching.
4. Verify that the decoded sequence is identical to the generated source sequence.
5. Calculate the realized number of bits per symbol.
6. Compare the realized value with the theoretical average length.
7. Repeat the procedure for Source B and the Huffman code.

```

N = 10000;
cdfA = cumsum(pA);
u = rand(1,N);
seqA = arrayfun(@(v)find(v<=cdfA,1,'first'),u);
bitstreamA = [codesSF{seqA}];
decodedA = prefixDecode(bitstreamA,codesSF);
assert(isequal(seqA,decodedA),'Decoded sequence is not correct.');
```

```

realizedLengthA = length(bitstreamA)/N;
fprintf('Source A realized length = %.6f bit/symbol
',realizedLengthA);
fprintf('Source A theoretical Lavg = %.6f bit/symbol
',Lsf);

```

MATLAB Local Functions

Place the following functions at the end of the MATLAB script. MATLAB releases that support local functions in scripts are required.

```

function codes = shannonFano(p)
    [pSorted,order] = sort(p,'descend');
    codesSorted = repmat({''},1,numel(p));
    codesSorted = sfAssign(pSorted,1:numel(pSorted),codesSorted);
    codes = cell(size(codesSorted));

```

```

    codes(order) = codesSorted;
end

function codes = sfAssign(p,ids,codes)
    if numel(ids) <= 1
        return;
    end
    cs = cumsum(p(ids));
    total = cs(end);
    [~,k] = min(abs(cs(1:end-1)-total/2));
    left = ids(1:k);
    right = ids(k+1:end);
    for q = left, codes{q} = [codes{q} '0']; end
    for q = right, codes{q} = [codes{q} '1']; end
    codes = sfAssign(p,left,codes);
    codes = sfAssign(p,right,codes);
end

function codes = binaryHuffman(p)
    nodes = cell(1,numel(p));
    for k = 1:numel(p)
        nodes{k} = struct('prob',p(k),'symbol',k,'left',[],'right',[]);
    end
    while numel(nodes) > 1
        probs = cellfun(@(n)n.prob,nodes);
        [~,ord] = sort(probs,'ascend');
        a = nodes{ord(1)};
        b = nodes{ord(2)};
        keep = true(1,numel(nodes));
        keep(ord(1:2)) = false;
        parent = struct('prob',a.prob+b.prob,'symbol',[], ...
            'left',a,'right',b);
        nodes = [nodes(keep) {parent}];
    end
    codes = repmat({''},1,numel(p));
    codes = walkTree(nodes{1},' ',codes);
end

function codes = walkTree(node,prefix,codes)
    if ~isempty(node.symbol)
        codes{node.symbol} = prefix;
        return;
    end
    codes = walkTree(node.left,[prefix '0'],codes);
    codes = walkTree(node.right,[prefix '1'],codes);
end

function decoded = prefixDecode(bitstream,codes)
    decoded = [];
    buffer = '';
    for k = 1:length(bitstream)
        buffer = [buffer bitstream(k)];
        idx = find(strcmp(buffer,codes),1);
        if ~isempty(idx)
            decoded(end+1) = idx; %#ok<AGROW>
            buffer = '';
        end
    end
    if ~isempty(buffer)
        error('Incomplete codeword at the end of the bitstream.');
```

6. Evaluation of Results

Complete the following tables using the PreLab calculations and MATLAB results.

Table 10.7. Information measures of Source A.

Symbol	Probability	Theoretical I (bit)	MATLAB I (bit)	Difference
A0	0.40			
A1	0.30			
A2	0.15			
A3	0.10			
A4	0.05			

Table 10.8. Source A entropy and coding results.

Quantity	Theoretical result	MATLAB result
Entropy $H(A)$		
Information rate at 1000 symbol/s		
Fixed-length average length		
Shannon-Fano average length		
Coding efficiency		
Redundancy		

Table 10.9. Joint and mutual information results.

Quantity	PreLab result	MATLAB result
$H(X)$		
$H(Y)$		
$H(X,Y)$		
$H(Y X)$		
$H(X Y)$		
$I(X;Y)$		

Table 10.10. Channel-capacity results.

Bandwidth	SNR	Theoretical C	MATLAB C	Percentage difference
4 kHz	S/N=7			
3 kHz	S/N=15			
5 kHz	10 dB			

Table 10.11. Source-coding comparison.

Source and method	Entropy	Average length	Efficiency	Redundancy
Source A fixed length				
Source A Shannon-Fano				
Source B fixed length				
Source B Huffman				

Table 10.12. Random-sequence encoding results.

Source	Number of symbols	Encoded bits	Realized bit/symbol	Theoretical Lavg	Decode errors
Source A					
Source B					

Questions

1. Why does a less probable symbol carry more self-information than a more probable symbol?
2. Under what probability condition does a binary source have maximum entropy?
3. What is the difference between entropy and information rate?
4. Explain the meaning of $H(Y|X)$ in the joint-distribution experiment.
5. Why is mutual information equal to zero for statistically independent random variables?
6. Compare the capacity values obtained for $B=4$ kHz, $S/N=7$ and $B=3$ kHz, $S/N=15$.
7. Why must an SNR value specified in decibels be converted before it is used in the Shannon-Hartley equation?
8. Why does assigning short codewords to high-probability symbols reduce average length?
9. What is the prefix condition and why is it necessary for immediate decoding?
10. Compare the top-down Shannon-Fano procedure with the bottom-up Huffman procedure.
11. Why can the average codeword length not generally be smaller than the source entropy?
12. Explain why the realized average length of a finite random sequence differs slightly from the theoretical average length.
13. Compare fixed-length coding with Shannon-Fano and Huffman coding in terms of efficiency and decoding complexity.
14. State the principal conclusions obtained from this experiment.

Experiment 11: Linear Block, Hamming and Cyclic Codes

1. Objective of the Experiment

The objective of this experiment is to introduce the matrix and polynomial descriptions used for channel coding in digital communication systems. Students will generate systematic linear block codewords, construct generator and parity-check matrices, calculate syndromes, and examine the error-detection and error-correction capability associated with the minimum Hamming distance.

The experiment also investigates the (7,4) Hamming code and binary cyclic codes. Students will implement modulo-2 encoding, inject controlled error patterns, locate single-bit errors by syndrome lookup, and verify the cyclic-shift property. Systematic cyclic encoding and syndrome calculation will be implemented using polynomial division over GF(2).

At the end of this experiment, students are expected to:

- Explain the purpose of controlled redundancy in channel coding.
- Distinguish systematic and nonsystematic block codes.
- Construct generator and parity-check matrices for a systematic linear block code.
- Generate codewords using modulo-2 matrix multiplication.
- Calculate the syndrome of a received vector.
- Determine the minimum Hamming distance and the corresponding detection and correction capabilities.
- Encode and decode messages using a (7,4) Hamming code.
- Locate and correct single-bit errors using a syndrome table.
- Represent binary codewords as polynomials.
- Generate systematic cyclic codewords using polynomial division.
- Verify that cyclic shifts of a valid codeword remain valid codewords.
- Compare matrix-based and polynomial-based channel-coding procedures.

2. Theoretical Background

2.1. Principle of Block Coding

In a block code, the channel encoder accepts a block of k message bits and adds $n-k$ parity or check bits. The encoder output is an n -bit codeword. The ratio k/n is the code rate. The added redundancy allows the receiver to detect or correct errors introduced during transmission through a noisy channel.

$$\text{Code rate } R_c = k/n$$

A systematic block code places the k message bits directly in the codeword and appends the $n-k$ parity bits. In a nonsystematic code, message and parity bits are not separately identifiable in the transmitted word.

Block-Coding and Syndrome-Decoding Chain

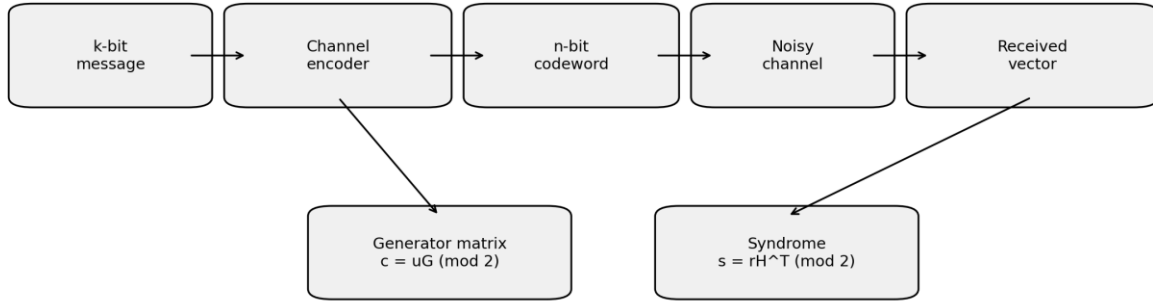


Fig. 11.1. Block-coding and syndrome-decoding chain.

2.2. Linear Block Codes

A block code is linear when the modulo-2 sum of any two codewords is also a valid codeword. For a binary systematic (n,k) code, a message vector u contains k bits and the corresponding codeword c contains n bits. The generator matrix G maps the message vector to the codeword:

$$c = uG \pmod{2}$$

For a systematic code, the generator matrix can be written as

$$G = [I_k \ P]$$

where I_k is the k by k identity matrix and P is the parity submatrix. The first k digits of the generated codeword reproduce the message bits, while the last $n-k$ digits are parity-check bits.

Matrix Description of a Systematic Linear Block Code

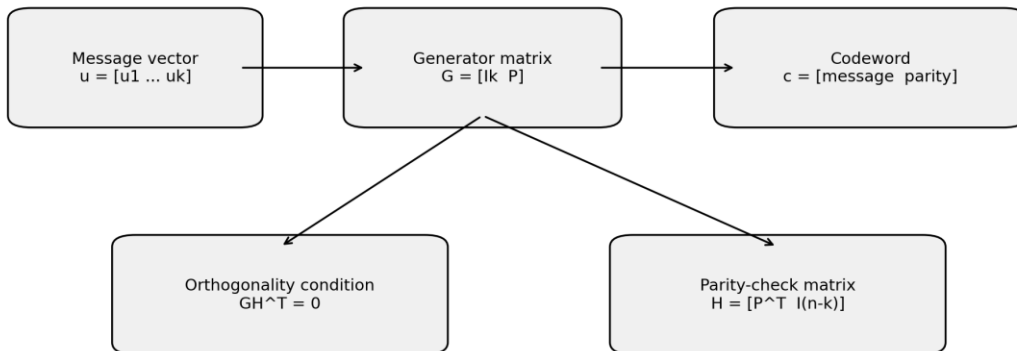


Fig. 11.2. Generator and parity-check matrix relationships for a systematic code.

2.3. Parity-Check Matrix and Syndrome

For $G = [I_k \ P]$, the parity-check matrix may be constructed as

$$H = [P^T \ I_{(n-k)}]$$

All operations are performed modulo two. A valid codeword satisfies the orthogonality condition

$$GH^T = 0$$

$$cH^T = 0$$

If the transmitted codeword c is altered by an error vector e , the received vector is $r = c + e$. The receiver calculates the syndrome

$$s = rH^T = eH^T \pmod{2}$$

The syndrome is therefore determined by the error pattern rather than by the transmitted codeword. A zero syndrome indicates that the received vector is a codeword. A nonzero syndrome indicates that the received vector is not a valid codeword, except for undetectable error patterns that transform one valid codeword into another.

2.4. Hamming Distance and Error Capability

The Hamming distance between two binary vectors is the number of bit positions in which they differ. For a linear code, the minimum distance $d_{\min}$ equals the minimum nonzero Hamming weight among all codewords. The lecture notes state that a code with minimum distance $d_{\min}$ can detect all error patterns containing at most $d_{\min} - 1$ errors. The number of errors that can be corrected is

$$t = \text{floor}((d_{\min} - 1)/2)$$

A code with $d_{\min} = 3$ can therefore correct one error and detect up to two errors. A single-error-correcting decoder must not be used to claim reliable correction of double errors, because a double-error syndrome may coincide with a syndrome assigned to a single-bit error.

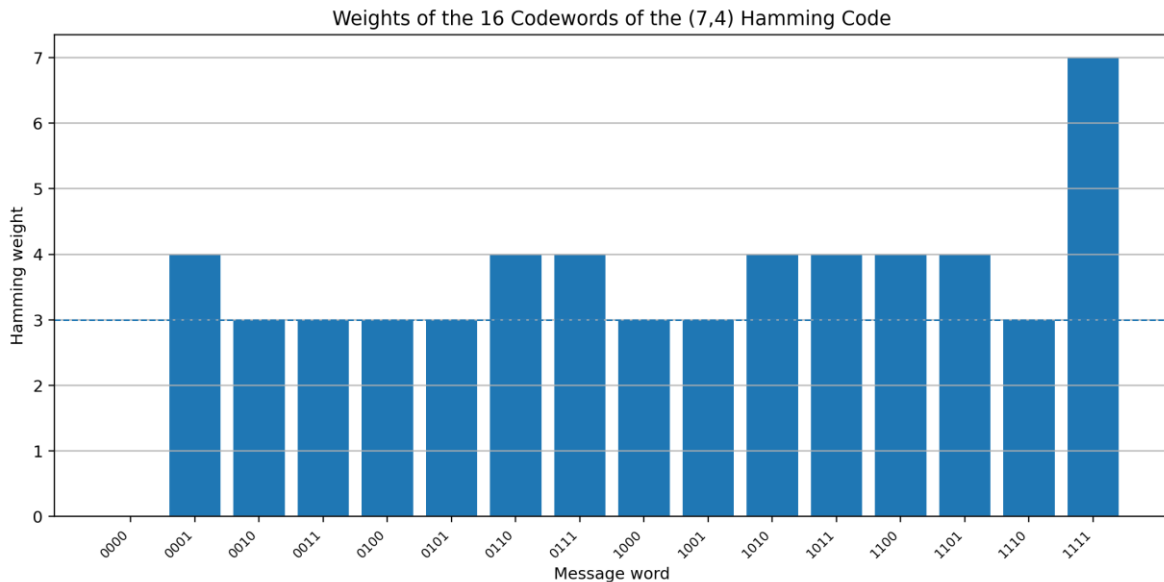


Fig. 11.3. Hamming weights of all codewords generated by the selected (7,4) code.

2.5. The (7,4) Hamming Code

For $m = 3$ parity-check bits, a Hamming code has $n = 2^m - 1 = 7$ bits, $k = 2^m - m - 1 = 4$ information bits, and $d_{\min} = 3$. The parity-check matrix contains all nonzero three-bit columns. Because each column is unique, the syndrome produced by a single-bit error identifies the error position.

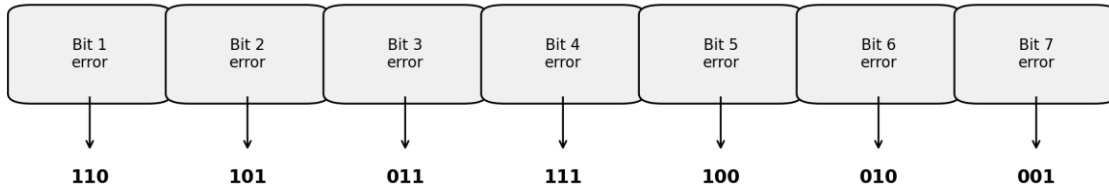
$$n = 7, \quad k = 4, \quad n - k = 3, \quad d_{\min} = 3, \quad t = 1$$

The systematic matrices used in this experiment are

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

$$G = [I_4 \ P], \quad H = [P^T \ I_3]$$

Single-Bit Error Location by Syndrome



Each nonzero column of H is a unique syndrome for one error position.

Fig. 11.4. Syndrome associated with each single-bit error position.

2.6. Binary Cyclic Codes

A cyclic code is a subclass of linear block codes. In addition to linearity, it has the cyclic property: every cyclic shift of a valid codeword is another valid codeword. An n -bit codeword can be represented by a polynomial of degree at most $n-1$. If the bit vector is $[x_{n-1} \dots x_1 x_0]$, the polynomial representation is

$$X(p) = x_{n-1}p^{n-1} + \dots + x_1 p + x_0$$

For an (n,k) cyclic code, the generator polynomial $G(p)$ has degree $q = n - k$ and divides $p^n + 1$ over $GF(2)$. In nonsystematic encoding, the codeword polynomial is obtained by multiplying the message polynomial by the generator polynomial:

$$X(p) = M(p)G(p)$$

2.7. Systematic Cyclic Encoding

For systematic encoding, q zeros are appended to the message, corresponding to multiplication by p^q . The shifted message polynomial is divided by $G(p)$, and the remainder $C(p)$ is appended to the message bits:

$$C(p) = \text{remainder}[p^q M(p) / G(p)]$$

$$X(p) = p^q M(p) + C(p)$$

Since addition and subtraction are identical in modulo-2 arithmetic, the generated codeword is exactly divisible by $G(p)$. In this experiment, the generator polynomial is

$$G(p) = p^3 + p + 1 \rightarrow [1\ 0\ 1\ 1]$$

Systematic Encoding of an (n,k) Cyclic Code

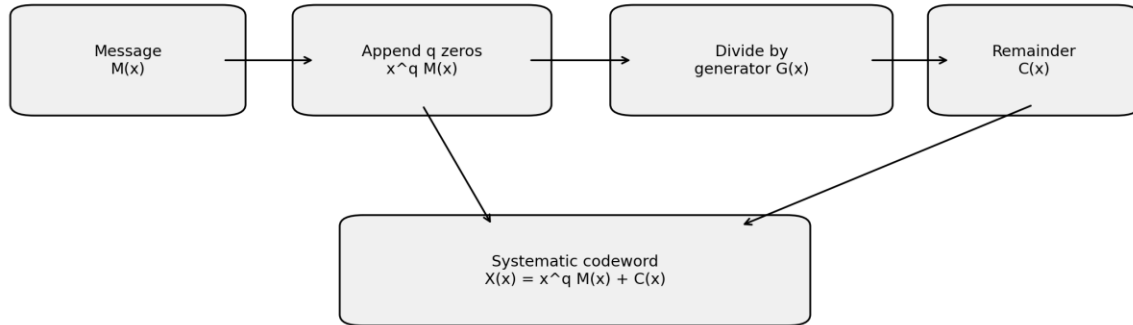


Fig. 11.5. Systematic cyclic encoding by modulo-2 polynomial division.

2.8. Syndrome Calculation for Cyclic Codes

If the received polynomial is $Y(p)$, the cyclic-code syndrome is the remainder obtained when $Y(p)$ is divided by $G(p)$:

$$S(p) = \text{remainder}[Y(p) / G(p)]$$

A zero remainder indicates that the received vector is a valid codeword. A nonzero remainder indicates an error, unless the error polynomial itself is a nonzero codeword. For single-error correction, a syndrome table can be generated by calculating the remainder associated with each one-bit error pattern.

Cyclic-Shift Property of a Valid Codeword

Shift 0:	1	1	0	1	0	0	1	
								↓
Shift 1:	1	0	1	0	0	1	1	
								↓
Shift 2:	0	1	0	0	1	1	1	
								↓
Shift 3:	1	0	0	1	1	1	0	
								↓
Shift 4:	0	0	1	1	1	0	1	
								↓
Shift 5:	0	1	1	1	0	1	0	
								↓
Shift 6:	1	1	1	0	1	0	0	

For a cyclic code, every cyclic shift of a valid codeword is also a valid codeword.

Fig. 11.6. Cyclic shifts of the valid codeword used in the experiment.

3. Materials Used

- Computer
- MATLAB software
- MATLAB Script Editor
- Digital Communications lecture notes
- Laboratory worksheet
- Calculator

The principal MATLAB commands and operations used in this experiment are:

- mod
- eye
- zeros
- ones
- transpose
- dec2bin
- bin2dec
- xor
- circshift
- sum
- min
- find
- table
- disp
- assert

4. Preliminary (PreLab)

Complete the following calculations before attending the laboratory.

4.1. Generator and Parity-Check Matrices

Use the parity submatrix

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

1. Construct the systematic generator matrix $G = [I_4\ P]$.
2. Construct the parity-check matrix $H = [P^T\ I_3]$.
3. Calculate GH^T modulo two and verify that the result is the zero matrix.
4. List the seven columns of H and confirm that each column is nonzero and unique.

Table 11.1. PreLab matrix verification.

Quantity	Expected size	Calculated result
G	4×7	
H	3×7	
GH^T	4×3	
Number of unique nonzero H columns	7	

4.2. Hamming Encoding and Syndrome

For the message

$$u = [1\ 1\ 0\ 1]$$

1. Calculate $c = uG$ modulo two.
2. Assume that an error occurs in bit position 6 and form the received vector r .
3. Calculate $s = rH^T$ modulo two.
4. Use the columns of H to identify the error position.
5. Correct the received vector and recover the original four message bits.

Table 11.2. PreLab Hamming-code calculations.

Quantity	Binary result
Message u	1 1 0 1
Codeword c	
Error vector e_6	
Received vector r	
Syndrome s	
Corrected codeword	

4.3. Minimum Distance

1. Generate all 16 possible four-bit messages.
2. Encode all messages using G .
3. Calculate the Hamming weight of every codeword.
4. Determine the minimum nonzero weight and hence $d_{\min}$.
5. Determine the guaranteed error-detection and error-correction capabilities.

Table 11.3. Minimum-distance result.

Parameter	Calculated value
Number of codewords	

Parameter	Calculated value
Minimum nonzero weight	
d_min	
Guaranteed detectable errors	
Guaranteed correctable errors	

4.4. Systematic Cyclic Encoding

Use the message and generator polynomial

$$M = [1 \ 1 \ 0 \ 1], \quad G(p) = p^3 + p + 1 \rightarrow [1 \ 0 \ 1 \ 1]$$

1. Append $q = 3$ zeros to the message.
2. Perform modulo-2 polynomial division by the generator polynomial.
3. Determine the three-bit remainder.
4. Form the systematic seven-bit codeword.
5. Divide the completed codeword by the generator and verify that the remainder is zero.
6. Generate one left cyclic shift and verify that the shifted word is also divisible by $G(p)$.

Table 11.4. PreLab cyclic-code calculations.

Quantity	Binary polynomial/vector
Message M	1 1 0 1
Generator G	1 0 1 1
Shifted message	
Remainder	
Systematic codeword	
First cyclic shift	
Remainder of shifted word	

5. Procedure

Part 1: Constructing G and H

1. Open MATLAB and create a new script file.
2. Define the parity submatrix P.
3. Construct $G = [I_4 \ P]$ and $H = [P^T \ I_3]$.
4. Calculate $\text{mod}(G*H, 2)$.
5. Use an assertion to verify the orthogonality condition.
6. Display G, H, and GH^T .

```
clear; close all; clc;
P = [1 1 0;
     1 0 1;
     0 1 1;
     1 1 1];
G = [eye(4) P];
H = [P.' eye(3)];
check = mod(G*H.', 2);
disp('Generator matrix G ='); disp(G);
disp('Parity-check matrix H ='); disp(H);
disp('G*H^T mod 2 ='); disp(check);
assert(all(check(:)==0), 'G and H do not satisfy GH^T = 0.');
```

Part 2: Encoding All Messages and Determining dmin

1. Generate the integers 0 through 15.
2. Convert each integer to a four-bit message.
3. Encode every message using modulo-2 matrix multiplication.
4. Calculate the Hamming weight of each codeword.
5. Determine the minimum nonzero codeword weight.
6. Calculate the guaranteed detection and correction capabilities.
7. Display the complete message-codeword table.

```

messages = dec2bin(0:15,4) - '0';
codewords = mod(messages*G,2);
weights = sum(codewords,2);
dmin = min(weights(weights>0));
tDetect = dmin-1;
tCorrect = floor((dmin-1)/2);
Tcode = table(string(dec2bin(0:15,4)), ...
    string(char(codewords+'0')),weights, ...
    'VariableNames',{'Message','Codeword','Weight'});
disp(Tcode);
fprintf('d_min = %d, detectable errors = %d, correctable errors = %d\n', ...
    dmin,tDetect,tCorrect);

```

Part 3: Single-Bit Syndrome Table and Hamming Decoding

1. Encode the message $u = [1 \ 1 \ 0 \ 1]$.
2. Generate all seven single-bit error vectors.
3. Calculate the syndrome produced by each error vector.
4. Construct a syndrome lookup table.
5. Inject an error successively at every codeword position.
6. Locate the error position from the syndrome.
7. Correct the received vector and verify that the original message is recovered.

```

u = [1 1 0 1];
c = mod(u*G,2);
errorPatterns = eye(7);
syndromeTable = mod(errorPatterns*H.',2);
Tsyntax = table((1:7).',string(char(syndromeTable+'0'))), ...
    'VariableNames',{'ErrorPosition','Syndrome'});
disp(Tsyntax);

for errorPosition = 1:7
    e = zeros(1,7); e(errorPosition)=1;
    r = xor(c,e);
    s = mod(r*H.',2);
    located = find(all(syndromeTable==s,2),1);
    corrected = r;
    if ~isempty(located)
        corrected(located) = xor(corrected(located),1);
    end
    fprintf('Actual=%d, located=%d, corrected=%s\n', ...
        errorPosition,located,num2str(corrected));
    assert(isequal(corrected,c),'Single-error correction failed.');
```

Part 4: Testing Double Errors

1. Inject errors into two selected bit positions.

2. Calculate the syndrome.
3. Observe whether the syndrome is nonzero.
4. Apply the single-error lookup decoder and observe the resulting codeword.
5. Repeat for at least five double-error patterns.
6. Explain why $d_{\min} = 3$ guarantees detection of two errors but not correction of two errors.

```
doubleErrors = [1 2; 1 4; 2 6; 3 7; 5 7];
for k = 1:size(doubleErrors,1)
    e = zeros(1,7); e(doubleErrors(k,:))=1;
    r = xor(c,e);
    s = mod(r*H.',2);
    located = find(all(syndroneTable==s,2),1);
    trial = r;
    if ~isempty(located)
        trial(located)=xor(trial(located),1);
    end
    fprintf('Errors at %d,%d | syndrome=%s | single-error-decoder output=%s\n', ...
        doubleErrors(k,1),doubleErrors(k,2),num2str(s),num2str(trial));
end
```

Part 5: Systematic Cyclic Encoding

1. Define the four-bit message and generator polynomial.
2. Append three zeros to the message.
3. Call the gf2Divide function to calculate the remainder.
4. Append the remainder to the original message.
5. Verify that the complete codeword produces a zero remainder.
6. Display the quotient, remainder, and systematic codeword.

```
message = [1 1 0 1];
g = [1 0 1 1]; % x^3 + x + 1
q = length(g)-1;
shifted = [message zeros(1,q)];
[quotient,remainder] = gf2Divide(shifted,g);
cyclicCodeword = [message remainder];
[~,checkRemainder] = gf2Divide(cyclicCodeword,g);

disp(['Shifted message: ' num2str(shifted)]);
disp(['Quotient: ' num2str(quotient)]);
disp(['Remainder: ' num2str(remainder)]);
disp(['Systematic word: ' num2str(cyclicCodeword)]);
disp(['Check remainder: ' num2str(checkRemainder)]);
assert(all(checkRemainder==0),'The generated word is not a cyclic codeword.');
```

Part 6: Verifying the Cyclic-Shift Property

1. Generate all seven cyclic shifts of the valid cyclic codeword.
2. Divide each shifted word by the generator polynomial.
3. Record the remainder for each shift.
4. Confirm that every remainder is zero.
5. Display the shifted words in a MATLAB table.

```
shiftedWords = zeros(7,7);
validShift = false(7,1);
for k = 0:6
    shiftedWords(k+1,:) = circshift(cyclicCodeword,[0 -k]);
    [~,remShift] = gf2Divide(shiftedWords(k+1,:),g);
    validShift(k+1) = all(remShift==0);
end
```

```
Tshift = table((0:6).',string(char(shiftedWords+'0'))),validShift, ...
    'VariableNames',{'ShiftNumber','ShiftedWord','ValidCodeword'});
disp(Tshift);
assert(all(validShift),'The cyclic-shift property was not verified.');
```

Part 7: Cyclic Syndrome Table and Single-Error Correction

1. Generate all seven single-bit error patterns.
2. Calculate the cyclic-code syndrome of each error pattern using polynomial division.
3. Create a syndrome table.
4. Inject one error into the cyclic codeword.
5. Calculate the received-word syndrome.
6. Locate the matching error pattern, correct the word, and verify zero remainder.

```
cyclicSyndromes = zeros(7,length(g)-1);
for pos = 1:7
    e = zeros(1,7); e(pos)=1;
    [~,cyclicSyndromes(pos,:)] = gf2Divide(e,g);
end

errorPosition = 5;
e = zeros(1,7); e(errorPosition)=1;
r = xor(cyclicCodeword,e);
[~,s] = gf2Divide(r,g);
located = find(all(cyclicSyndromes==s,2),1);
corrected = r;
if ~isempty(located)
    corrected(located)=xor(corrected(located),1);
end
[~,finalRemainder] = gf2Divide(corrected,g);

fprintf('Actual error position = %d\n',errorPosition);
fprintf('Located error position = %d\n',located);
fprintf('Syndrome = %s\n',num2str(s));
fprintf('Corrected word = %s\n',num2str(corrected));
assert(all(finalRemainder==0),'Cyclic-code correction failed.');
```

Part 8: Required Local Function for GF(2) Division

Place the following local function at the end of the MATLAB script. Vectors are written from the coefficient of the highest polynomial power to the constant term.

```
function [quotient,remainder] = gf2Divide(dividend,divisor)
    dividend = mod(dividend,2);
    divisor = mod(divisor,2);
    if divisor(1)~=1
        error('The highest-order coefficient of the divisor must be one.');
```

```
    end
    quotient = zeros(1,length(dividend)-length(divisor)+1);
    working = dividend;
    for k = 1:length(quotient)
        if working(k)==1
            quotient(k)=1;
            working(k:k+length(divisor)-1) = ...
                xor(working(k:k+length(divisor)-1),divisor);
        end
    end
    remainder = working(end-length(divisor)+2:end);
end
```

6. Evaluation of Results

Complete the following tables using the theoretical calculations and MATLAB results.

Table 11.5. Generator and parity-check matrix verification.

Quantity	Theoretical result	MATLAB result	Agreement
Dimensions of G	4×7		
Dimensions of H	3×7		
$GH^T \bmod 2$	Zero matrix		
Unique nonzero columns of H	7		

Table 11.6. Hamming-code encoding and decoding.

Message	Codeword	Error position	Received vector	Syndrome	Located position	Corrected word
1101		1				
1101		3				
1101		5				
1101		7				

Table 11.7. Minimum-distance and error capability.

Parameter	Theoretical value	MATLAB value
Number of valid codewords	16	
Minimum nonzero codeword weight	3	
$d_{\min}$	3	
Guaranteed detectable errors	2	
Guaranteed correctable errors	1	

Table 11.8. Double-error observations.

Error positions	Syndrome	Nonzero syndrome?	Output of single-error decoder	Original codeword recovered?
1,2				
1,4				
2,6				
3,7				
5,7				

Table 11.9. Systematic cyclic encoding.

Message	Generator	Remainder	Codeword	Division-check remainder
1101	1011			
1001	1011			
1110	1011			

Table 11.10. Cyclic-shift verification.

Shift number	Shifted codeword	Remainder	Valid codeword?
0			
1			
2			
3			
4			
5			

Shift number	Shifted codeword	Remainder	Valid codeword?
6			

Table 11.11. Matrix and cyclic-code comparison.

Feature	Linear/Hamming implementation	Cyclic implementation
Representation	Matrices G and H	Polynomials and generator G(p)
Encoding operation	$uG \bmod 2$	Polynomial multiplication/division
Error test	rH^T	Remainder of $Y(p)/G(p)$
Single-error location	Column of H	Syndrome lookup
Structural property	Linearity	Linearity and cyclic shifts

Questions

1. Why is redundancy intentionally added by a channel encoder?
2. What is the difference between a systematic and a nonsystematic block code?
3. Why must a valid codeword satisfy $cH^T = 0$?
4. Why does the syndrome depend only on the error pattern?
5. How does the minimum distance determine the error-detection and error-correction capability?
6. Why does the selected (7,4) Hamming code correct one error?
7. Why are the columns of H required to be nonzero and distinct for single-error correction?
8. What happens when the single-error decoder is applied to a double-error pattern?
9. How is a binary vector represented as a polynomial?
10. Why is the remainder appended to the message in systematic cyclic encoding?
11. How does the cyclic-shift property distinguish a cyclic code from a general linear block code?
12. Why does a zero cyclic syndrome not always prove that no transmission error occurred?
13. Compare matrix-based syndrome calculation with polynomial-remainder syndrome calculation.
14. Comment on the code rate and redundancy of the (7,4) code.
15. Summarize the main differences between Hamming and cyclic decoding procedures used in this experiment.

Submit the following files:

- MATLAB script file with the .m extension, including the gf2Divide local function.
- Completed PreLab calculations.
- Completed result tables.
- Message-codeword table for all 16 Hamming codewords.
- Single-bit syndrome tables for the matrix and cyclic implementations.
- Brief discussion of single-error correction and double-error limitations.

Experiment 12: Convolutional Encoding, Trellis Representation and Viterbi Decoding**1. Objective of the Experiment**

The objective of this experiment is to investigate convolutional channel coding and maximum-likelihood sequence decoding. Students will implement the rate-1/2 convolutional encoder presented in the lecture notes, generate encoded sequences using modulo-2 shift-register operations, and verify the same output through time-domain convolution and transform-domain polynomial multiplication.

The experiment also introduces the state, code-tree, and trellis descriptions of a convolutional encoder. Hard-decision Viterbi decoding will be applied to received sequences containing controlled bit errors. Branch metrics, accumulated path metrics, survivor paths, and traceback will be examined step by step.

At the end of this experiment, students are expected to:

- Explain how a convolutional encoder combines the present input bit with stored previous bits.
- Determine the code rate, encoder memory, number of states, and constraint length.
- Generate output sequences using shift-register and modulo-2 operations.
- Express convolutional encoding in the time domain and transform domain.
- Construct the state-transition table of a convolutional encoder.
- Interpret state, code-tree, and trellis diagrams.
- Calculate branch metrics using Hamming distance.
- Calculate accumulated path metrics and select surviving paths.
- Decode a received sequence using the hard-decision Viterbi algorithm.
- Explain the purpose of terminating the encoder in a known state.
- Compare uncoded transmission with convolutionally coded transmission in a binary-error channel.

2. Theoretical Background**2.1. Definition of Convolutional Coding**

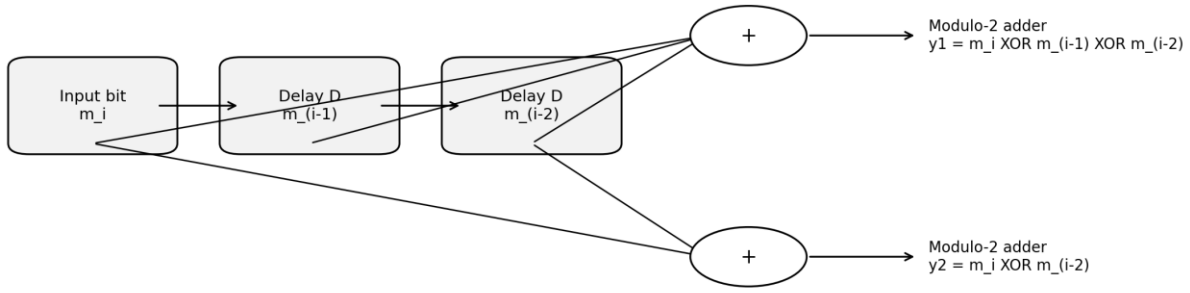
A convolutional encoder continuously combines a fixed number of present and previous message bits. The input bits are stored in a shift register, and selected register contents are connected to modulo-2 adders. Unlike a block code, the output associated with one information bit depends not only on that bit but also on the encoder memory.

The encoder used in this experiment follows the structure presented in Unit V of the lecture notes. One input bit produces two output bits. The two output equations are

$$x1(i) = m(i) \text{ XOR } m(i-1) \text{ XOR } m(i-2)$$

$$x2(i) = m(i) \text{ XOR } m(i-2)$$

Rate-1/2 Convolutional Encoder Used in the Experiment



Generator sequences: $g1 = [1\ 1\ 1]$ and $g2 = [1\ 0\ 1]$ (octal 7 and 5).

Fig. 12.1. Rate-1/2 convolutional encoder used in the experiment.

2.2. Code Rate, Memory and Constraint Length

If k input bits are accepted and n coded bits are produced during each encoding interval, the code rate is

$$R_c = k/n$$

For the selected encoder, $k = 1$ and $n = 2$; therefore $R_c = 1/2$. The encoder stores two previous input bits, so the memory order is $M = 2$. The constraint length is the number of consecutive input bits that can influence the output and is therefore $K = M + 1 = 3$. The encoder has $2^M = 4$ possible states.

$$(n, k, K) = (2, 1, 3), \quad \text{number of states} = 2^{(K-1)} = 4$$

2.3. Time-Domain Approach

In the time-domain approach, the encoder outputs are obtained by modulo-2 convolution between the message sequence and the generator sequences. For the present encoder,

$$g1 = [1\ 1\ 1], \quad g2 = [1\ 0\ 1]$$

$$x1(i) = \text{sum}[g1(j)m(i-j)] \text{ mod } 2$$

$$x2(i) = \text{sum}[g2(j)m(i-j)] \text{ mod } 2$$

The two output sequences are multiplexed pair by pair to form the serial coded stream. The lecture notes emphasize that each new input bit remains in the register for three consecutive shifts and therefore affects several successive output pairs.

Time-Domain Encoding Example for m = 1101

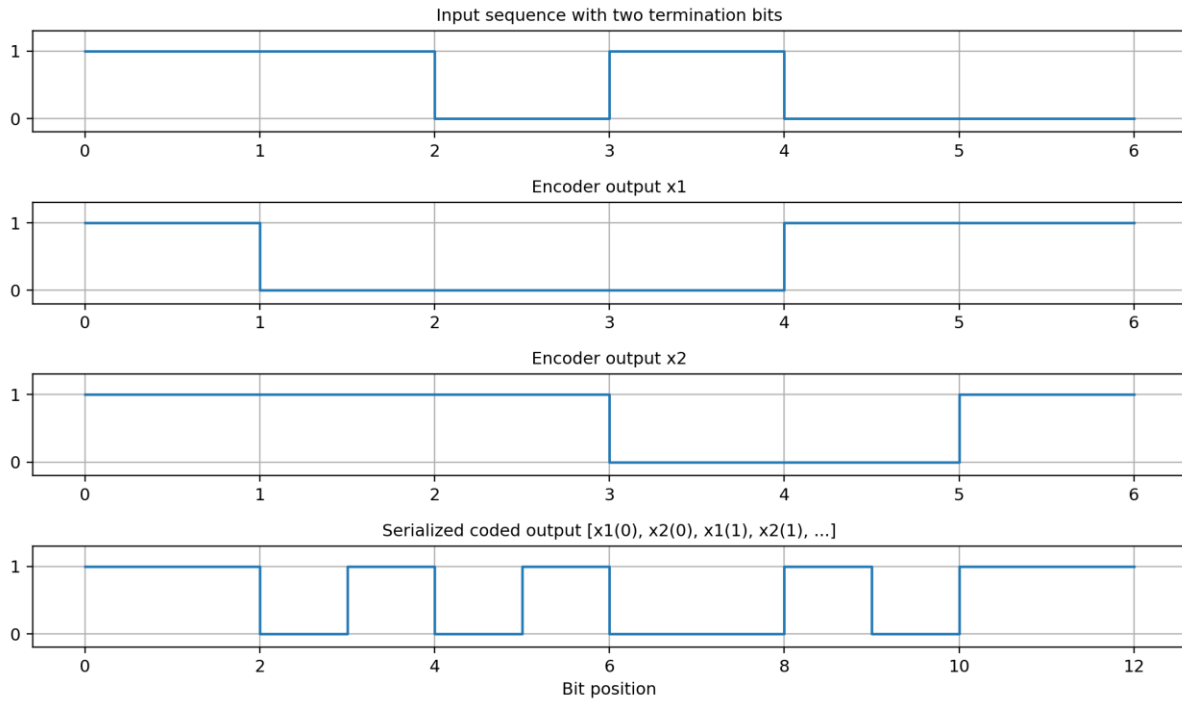


Fig. 12.2. Time-domain encoding example with termination bits.

2.4. Transform-Domain Approach

The transform-domain approach replaces binary convolution by polynomial multiplication over $GF(2)$. Let D denote one unit delay. The generator polynomials of the encoder are

$$G1(D) = 1 + D + D^2$$

$$G2(D) = 1 + D^2$$

For message polynomial $M(D)$, the output polynomials are

$$X1(D) = M(D)G1(D) \mod 2$$

$$X2(D) = M(D)G2(D) \mod 2$$

The time-domain and transform-domain approaches must produce identical output sequences when the same bit-order convention and initial state are used.

2.5. Encoder State and State-Transition Table

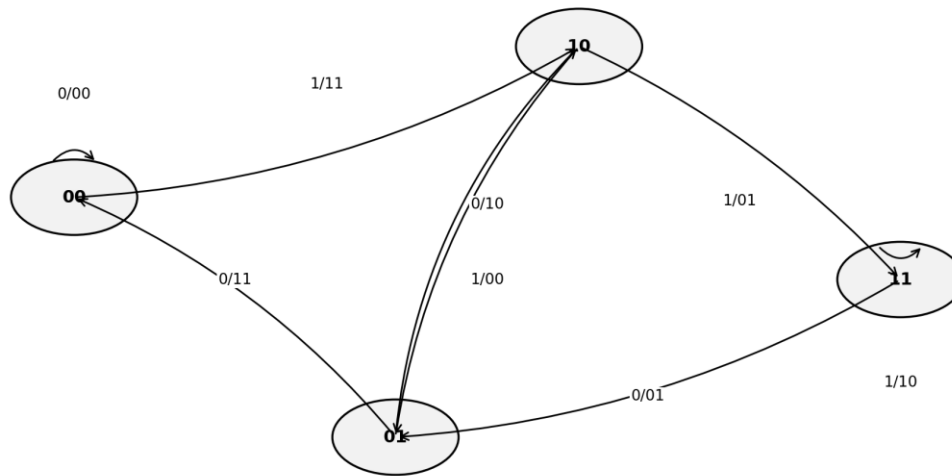
The state of the selected encoder is defined by the two stored bits $[m(i-1), m(i-2)]$. Each new input bit determines both the two output bits and the next state. The complete transition table is shown below.

Table 12.1. State-transition table of the selected encoder.

Current state	Input	Output x1x2	Next state
00	0	00	00
00	1	11	10
01	0	11	00
01	1	00	10
10	0	10	01

Current state	Input	Output x1x2	Next state
10	1	01	11
11	0	01	01
11	1	10	11

State Diagram for the (2,1,3) Convolutional Encoder



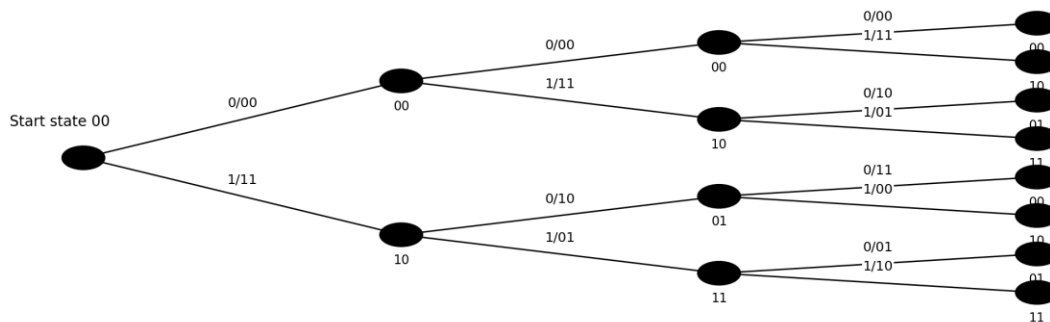
Branch label = input bit / two output bits

Fig. 12.3. State diagram with input/output branch labels.

2.6. Code Tree and Trellis Diagram

A code tree displays every possible encoded path from the initial state. Each branch corresponds to an input bit and is labeled by the resulting output pair. The number of branches doubles at every level, so a code tree becomes large for long sequences.

Code Tree from the All-Zero Initial State



Each path represents one possible input sequence; branches never merge in the code tree.

Fig. 12.4. First three levels of the code tree.

A trellis merges tree nodes that represent the same encoder state at the same time. This repeated state structure makes sequence decoding practical because competing paths can be compared and inferior paths can be discarded.

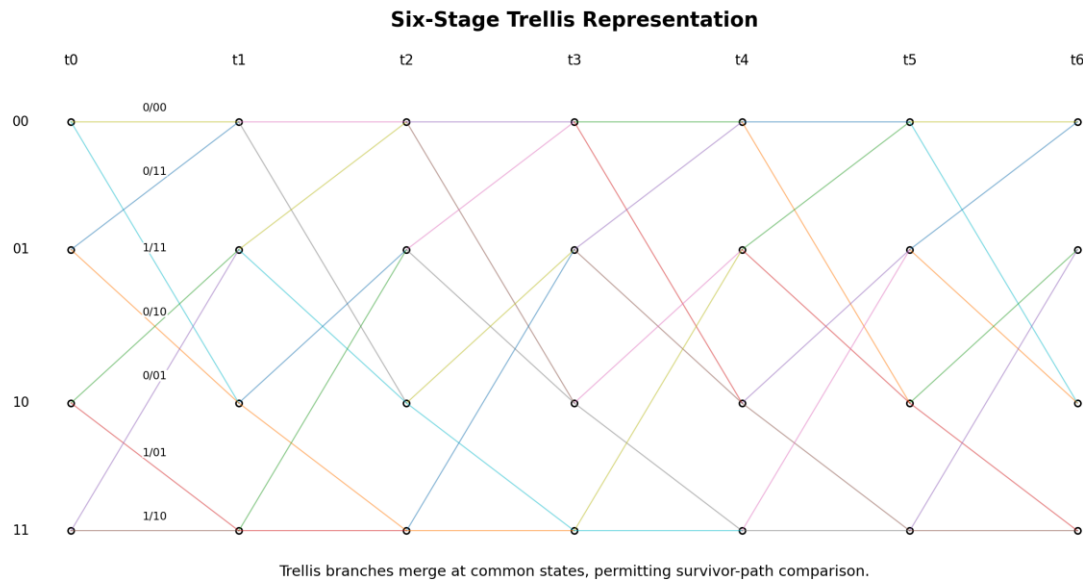


Fig. 12.5. Six-stage trellis diagram of the encoder.

2.7. Viterbi Decoding

The lecture notes define the Viterbi algorithm as a maximum-likelihood decoding method for convolutional codes. The received sequence is compared with the output sequence associated with each trellis branch. For hard-decision decoding, the branch metric is the Hamming distance between the received pair and the expected pair.

$$\text{Branch metric} = \text{Hamming distance}(\text{received pair}, \text{expected pair})$$

A path metric is obtained by adding the branch metrics along a path. When two paths enter the same state, the path with the smaller metric is retained as the surviving path. After the complete received sequence is processed, traceback through the surviving path produces the decoded information sequence.

The essential Viterbi steps are:

- Initialize the metric of the known starting state to zero and all other state metrics to infinity.
- Calculate branch metrics for all permitted state transitions.
- Add each branch metric to the corresponding previous path metric.
- Compare paths entering the same state and retain the path with the smallest metric.
- Repeat for every received output pair.
- Select the final state and trace the survivor path backward to recover the input bits.

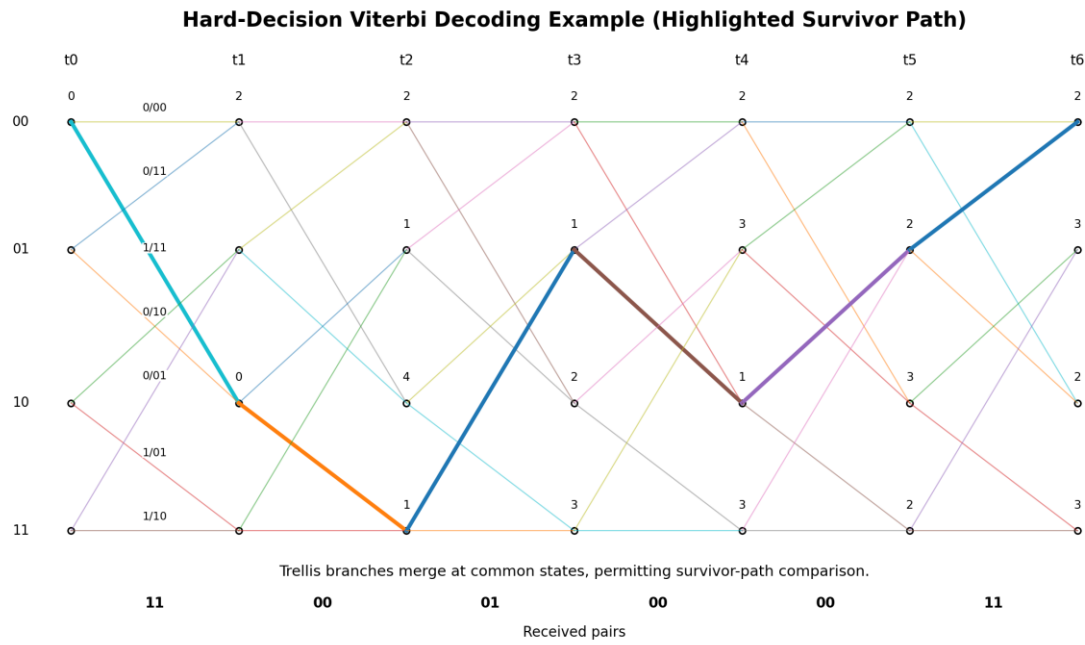


Fig. 12.6. Viterbi survivor path for a received sequence containing two bit errors.

2.8. Laboratory Termination Convention

The lecture notes describe an initially zero encoder. For the MATLAB implementation in this laboratory, two zero tail bits are appended to each information sequence. This returns the memory-2 encoder to state 00 and provides a known final state for traceback. The tail bits are removed after decoding and are not counted as information bits.

3. Materials Used

- Computer
- MATLAB software
- MATLAB Script Editor
- Digital Communications lecture notes
- Laboratory worksheet
- Calculator

The principal MATLAB commands and operations used in this experiment are:

- xor
- mod
- conv
- zeros
- ones
- reshape
- inf
- min
- find
- randi
- rand
- bitxor

- semilogy
- table
- disp
- assert

4. Preliminary (PreLab)

Complete the following calculations before attending the laboratory.

4.1. Encoder Parameters

1. For $k = 1$ and $n = 2$, calculate the code rate.
2. For two memory elements, calculate the constraint length and number of states.
3. Write the generator sequences and generator polynomials.
4. List the four encoder states.

Table 12.2. PreLab encoder parameters.

Parameter	Symbol	Calculated value
Input bits per interval	k	
Output bits per interval	n	
Code rate	R_c	
Memory order	M	
Constraint length	K	
Number of states	2^M	

4.2. Manual Shift-Register Encoding

Encode the information sequence

$$m = [1 \ 1 \ 0 \ 1]$$

Assume initial state 00 and append two zero tail bits. Complete the following table using the equations of the encoder.

Table 12.3. Manual convolutional encoding.

i	Input $m(i)$	Current state	$x1(i)$	$x2(i)$	Next state
0	1	00			
1	1	10			
2	0	11			
3	1	01			
4	0	10			
5	0	01			

1. Write the $x1$ and $x2$ output sequences.
2. Write the serialized output sequence in $x1$ - $x2$ pair order.
3. Confirm that the last state after the two tail bits is 00.

4.3. Transform-Domain Verification

For the time-order convention $m = [m0 \ m1 \ m2 \ m3]$, write

$$M(D) = 1 + D + D^3$$

1. Calculate $X1(D) = M(D)(1 + D + D^2)$ modulo two.
2. Calculate $X2(D) = M(D)(1 + D^2)$ modulo two.
3. Write the coefficient sequences of $X1(D)$ and $X2(D)$.
4. Compare the results with the manual shift-register outputs.

Table 12.4. Time-domain and transform-domain comparison.

Method	x1 sequence	x2 sequence	Identical?
Shift-register method			
Polynomial method			

4.4. State Transition and Viterbi Metric

1. Complete the state-transition table for all four current states and both input values.
2. For transmitted output 11 01 01 00 10 11, form a received sequence by reversing coded bits 4 and 9.
3. Group the received stream into two-bit symbols.
4. At the first trellis stage, calculate the Hamming metric of the two possible branches from state 00.
5. Predict whether the original information sequence can be recovered by minimum-metric decoding.

Table 12.5. Initial Viterbi calculations.

Quantity	Binary value / metric
Transmitted coded stream	11 01 01 00 10 11
Received coded stream	
Metric for 00 -> 00 branch	
Metric for 00 -> 10 branch	
Predicted decoded message	

5. Procedure

Part 1: Defining the Encoder Trellis

1. Open MATLAB and create a new script file.
2. Define the four states as [00; 01; 10; 11].
3. For every state and input bit, calculate the next state and output pair.
4. Display the state-transition table.
5. Compare the generated table with Table 12.1.

```
clear; close all; clc;
states = [0 0; 0 1; 1 0; 1 1];
nextState = zeros(4,2);
outPair = zeros(4,2,2);
for s = 1:4
    m1 = states(s,1); m2 = states(s,2);
    for b = 0:1
        y1 = xor(xor(b,m1),m2);
        y2 = xor(b,m2);
        ns = [b m1];
        nextState(s,b+1) = find(ismember(states,ns,'rows'));
        outPair(s,b+1,:) = [y1 y2];
    end
end
```

```
disp(nextState);
disp(outPair);
```

Part 2: Shift-Register Encoding

1. Define the information sequence $m = [1 \ 1 \ 0 \ 1]$.
2. Append two zero tail bits.
3. Initialize the encoder state to 00.
4. Calculate x_1 and x_2 at every interval.
5. Serialize the output pairs.
6. Plot the input, x_1 , x_2 , and serialized coded sequence.

```
m = [1 1 0 1];
inputWithTail = [m 0 0];
state = [0 0];
pairs = zeros(length(inputWithTail),2);
stateHistory = zeros(length(inputWithTail)+1,2);
stateHistory(1,:) = state;
for i = 1:length(inputWithTail)
    b = inputWithTail(i);
    pairs(i,1) = xor(xor(b,state(1)),state(2));
    pairs(i,2) = xor(b,state(2));
    state = [b state(1)];
    stateHistory(i+1,:) = state;
end
coded = reshape(pairs.',1,[]);
disp('Output pairs ='); disp(pairs);
disp('Serialized code ='); disp(coded);
assert(all(state==0),'Encoder did not return to state 00.');
```

Part 3: Time-Domain and Transform-Domain Verification

1. Use the generator sequences $g_1 = [1 \ 1 \ 1]$ and $g_2 = [1 \ 0 \ 1]$.
2. Calculate the modulo-2 convolution outputs.
3. Compare the polynomial outputs with the shift-register outputs.
4. Use assertions to verify equality.

```
g1 = [1 1 1];
g2 = [1 0 1];
x1Poly = mod(conv(m,g1),2);
x2Poly = mod(conv(m,g2),2);
assert(isequal(x1Poly,pairs(:,1).'), ...
    'The x1 sequences are not identical.');
```

```
assert(isequal(x2Poly,pairs(:,2).'), ...
    'The x2 sequences are not identical.');
```

```
disp('x1 polynomial output ='); disp(x1Poly);
disp('x2 polynomial output ='); disp(x2Poly);
```

Part 4: Controlled-Error Viterbi Decoding

1. Use the encoded sequence generated in Part 2.
2. Reverse coded bits 4 and 9 to create a received sequence containing two bit errors.
3. Group the received sequence into two-bit symbols.
4. Initialize the path metric of state 00 to zero and the remaining metrics to infinity.
5. Calculate Hamming branch metrics for both possible inputs from every reachable state.
6. Retain the lower-metric predecessor for every next state.

7. Trace back from final state 00.
8. Remove the two zero tail bits and compare the decoded sequence with the original message.

```
received = coded;
received([4 9]) = xor(received([4 9]),1);
rxPairs = reshape(received,2,[]).';
[decodedWithTail,metricHistory,statePath] = ...
    viterbiHardR12(rxPairs,nextState,outPair,true);
decoded = decodedWithTail(1:end-2);
disp('Received pairs ='); disp(rxPairs);
disp('Path metrics ='); disp(metricHistory);
disp('Decoded message ='); disp(decoded);
assert(isequal(decoded,m),'Viterbi decoder failed.');
```

Part 5: Viterbi Decoder Function

Place the following local function at the end of the MATLAB script.

```
function [decoded,metricHistory,statePath] = ...
    viterbiHardR12(rxPairs,nextState,outPair,forceZeroFinal)
T = size(rxPairs,1);
INF = 1e9;
metricHistory = INF*ones(T+1,4);
metricHistory(1,1) = 0; % initial state 00
predecessor = zeros(T+1,4);
inputMemory = zeros(T+1,4);
for t = 1:T
    for s = 1:4
        if metricHistory(t,s) >= INF, continue; end
        for b = 0:1
            ns = nextState(s,b+1);
            expected = squeeze(outPair(s,b+1,:)).';
            branchMetric = sum(expected ~= rxPairs(t,:));
            candidate = metricHistory(t,s) + branchMetric;
            if candidate < metricHistory(t+1,ns)
                metricHistory(t+1,ns) = candidate;
                predecessor(t+1,ns) = s;
                inputMemory(t+1,ns) = b;
            end
        end
    end
end
if forceZeroFinal
    currentState = 1; % state 00
else
    [~,currentState] = min(metricHistory(end,:));
end
decoded = zeros(1,T); statePath = zeros(1,T+1);
statePath(T+1) = currentState;
for t = T:-1:1
    decoded(t) = inputMemory(t+1,currentState);
    currentState = predecessor(t+1,currentState);
    statePath(t) = currentState;
end
end
```

MATLAB Helper Function: Convolutional Encoder

Place this local function after the Viterbi decoder function when completing Part 6.

```
function [coded,pairs,stateHistory] = convEncoder12(data,terminate)
if terminate, inputSequence = [data 0 0];
else, inputSequence = data; end
state = [0 0];
pairs = zeros(length(inputSequence),2);
stateHistory = zeros(length(inputSequence)+1,2);
stateHistory(1,:) = state;
for i = 1:length(inputSequence)
    b = inputSequence(i);
```

```

pairs(i,1) = xor(xor(b,state(1)),state(2));
pairs(i,2) = xor(b,state(2));
state = [b state(1)];
stateHistory(i+1,:) = state;
end
coded = reshape(pairs.',1,[]);
end

```

Part 6: Random-Error Performance Extension

This part is a laboratory extension of the hard-decision Hamming-metric method. It compares an uncoded binary sequence with the rate-1/2 convolutional code over a binary symmetric channel.

1. Generate random 100-bit frames.
2. For the uncoded case, reverse each bit independently with probability p .
3. For the coded case, encode and terminate each frame, then apply the same crossover probability to each coded bit.
4. Decode with the Viterbi function.
5. Calculate the uncoded and coded bit-error rates for several p values.
6. Plot the results on a logarithmic vertical scale.

```

rng(14);
pList = [0.005 0.01 0.02 0.03 0.05 0.08 0.10 0.12];
frameLength = 100; numberOfFrames = 100;
BERuncoded = zeros(size(pList)); BERcoded = zeros(size(pList));
for ip = 1:length(pList)
    errU = 0; errC = 0; total = 0;
    for frame = 1:numberOfFrames
        data = randi([0 1],1,frameLength);
        rxU = xor(data,rand(size(data)) < pList(ip));
        errU = errU + sum(rxU ~= data);
        [txC,~,~] = convEncodeR12(data,true);
        rxC = xor(txC,rand(size(txC)) < pList(ip));
        rxCPairs = reshape(rxC,2,[]).';
        decTail = viterbiHardR12(rxCPairs,nextState,outPair,true);
        dec = decTail(1:end-2);
        errC = errC + sum(dec ~= data);
        total = total + frameLength;
    end
    BERuncoded(ip) = errU/total;
    BERcoded(ip) = errC/total;
end
semilogy(pList,BERuncoded,'o-',pList,BERcoded,'s-');
grid on; xlabel('Crossover probability p'); ylabel('BER');
legend('Uncoded','Convolutional code + Viterbi','Location','best');

```

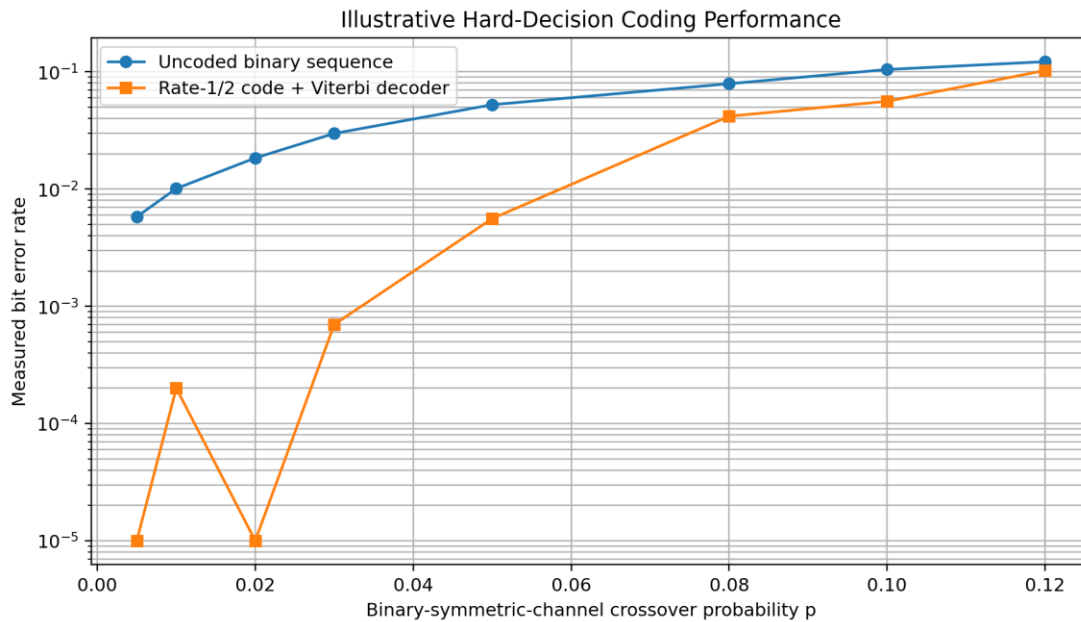


Fig. 12.7. Illustrative hard-decision performance over a binary symmetric channel.

6. Evaluation of Results

Complete the following tables and answer the evaluation questions.

Table 12.6. Encoder parameters and generated sequences.

Quantity	Theoretical result	MATLAB result
Code rate		
Constraint length		
Number of states		
x1 sequence		
x2 sequence		
Serialized coded stream		

Table 12.7. State transition verification.

Current state	Input 0: output/next state	Input 1: output/next state
00		
01		
10		
11		

Table 12.8. Viterbi decoding results.

Quantity	Result
Original information bits	
Transmitted coded pairs	
Error positions	
Received coded pairs	
Final survivor metric	
Decoded bits including tail	
Recovered information bits	
Number of corrected coded-bit errors	

Table 12.9. Random-error performance.

Crossover probability p	Uncoded BER	Coded BER	Improvement observed?
0.005			
0.010			
0.020			
0.030			
0.050			
0.080			
0.100			
0.120			

Evaluation Questions

1. Why does one input bit affect more than one output interval in a convolutional encoder?
2. What are the code rate, memory order, constraint length, and number of states of the selected encoder?
3. How are the generator sequences $[1\ 1\ 1]$ and $[1\ 0\ 1]$ related to the shift-register connections?
4. Why do the time-domain and transform-domain encoding methods produce the same result?
5. What information is represented by the encoder state?
6. What is the main difference between a code tree and a trellis diagram?
7. Why can competing paths entering the same state be compared directly?
8. How is a hard-decision branch metric calculated?
9. What is a surviving path in the terminology of the Viterbi algorithm?
10. Why are path metrics accumulated rather than evaluated one branch at a time?
11. What is the purpose of the two zero tail bits used in this laboratory?
12. What would change if the decoder did not know the final state?
13. Under what conditions may two paths have equal metrics, and how should a decoder handle this tie?
14. Why does Viterbi decoding not guarantee correction of every arbitrary error pattern?
15. Compare the uncoded and coded BER results. What is gained, and what cost is introduced by the rate-1/2 code?
16. Explain how the encoder, trellis, and Viterbi decoder together provide error-control capability.