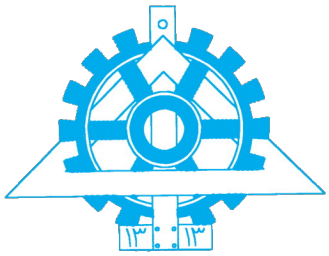

Signals and Systems

Computer Assignment 5 Report

University of Tehran

School of Electrical and Computer Engineering

Student Name: Parsa Bukani
Student Number: 810102501
Course: Signals and Systems
Instructor: Dr. Akhavan
Semester: Fall 1403

December 14, 2025

Contents

Introduction

This report presents the complete solution, MATLAB codes, figures, and analysis for Computer Assignment 5 of the Signals and Systems course. The assignment consists of two major parts. Each subsection includes explanations, figures, and source code when required.

Part 1: Fourier Analysis and Frequency Resolution

Exercise 1.1: Time and Frequency Analysis of a Cosine Signal

In this exercise, the signal

$$x(t) = \cos(10\pi t)$$

is analyzed in both the time domain and the frequency domain. The signal is sampled with a sampling frequency of $f_s = 50\,\text{Hz}$ over the time interval $-1 \leq t \leq 1$ seconds.

Time-Domain Representation

The signal is uniformly sampled and plotted in the time domain. Since the angular frequency of the cosine is 10π, the corresponding frequency is $f = 5\,\text{Hz}$. The sampled signal clearly shows a periodic cosine waveform with the expected frequency and amplitude.

```
% Exercise 1.1(a): Time-domain representation

fs = 50;
Ts = 1/fs;
t = -1:Ts:1;

x = cos(10*pi*t);

figure;
plot(t, x, 'LineWidth', 1.5);
grid on;
xlabel('Time␣(s)');
ylabel('x(t)');
```

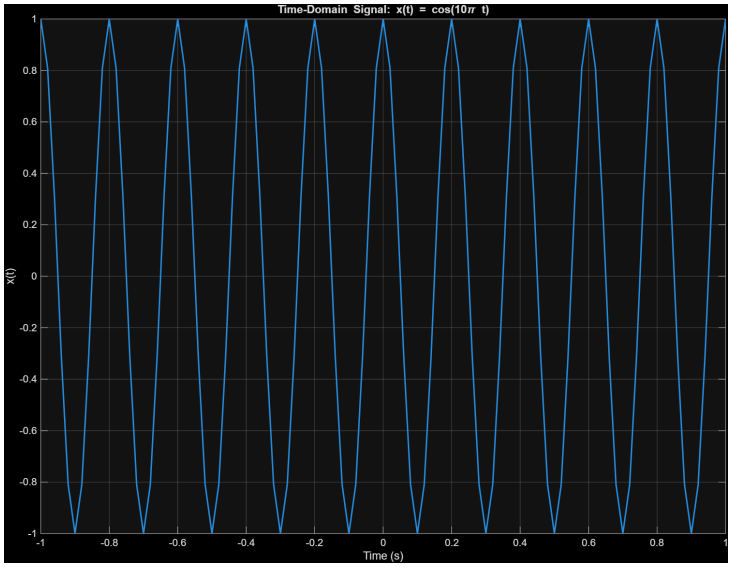

Figure 1: Time-domain representation of $x(t) = \cos(10\pi t)$ sampled at $f_s = 50$ Hz.

Magnitude of the Fourier Transform

To analyze the frequency content of the signal, the discrete Fourier transform is computed using the `fft` function and shifted using `fftshift` to obtain a symmetric frequency axis around zero. The magnitude of the Fourier transform is normalized so that its maximum value is equal to one.

```
% Exercise 1.1(b): Magnitude spectrum

N = length(x);
X = fftshift(fft(x));
f = (-N/2:N/2-1) * (fs/N);

X_mag = abs(X);
X_mag = X_mag / max(X_mag);

figure;
plot(f, X_mag, 'LineWidth', 1.5);
grid on;
xlabel('Frequency␣(Hz)');
ylabel('|X(f)|␣(normalized)');
```

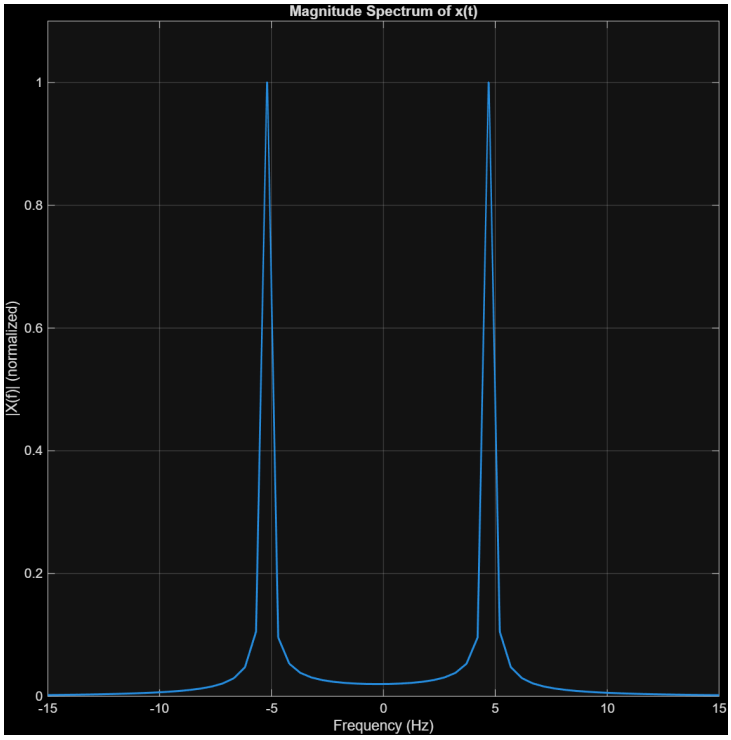

Figure 2: Normalized magnitude of the Fourier transform of $x(t) = \cos(10\pi t)$.

The spectrum contains two dominant peaks at frequencies $\pm 5\,\text{Hz}$, which is exactly consistent with the theoretical Fourier transform of a cosine signal. This confirms that the discrete-time simulation correctly captures the frequency content of the original continuous-time signal.

Exercise 1.2: Analysis of a Phase-Shifted Cosine Signal

In this exercise, the signal

$$x(t) = \cos(30\pi t + \pi/4)$$

is analyzed in the time domain and frequency domain. The signal is sampled with a sampling frequency of $f_s = 100\,\text{Hz}$ over the interval $0 \leq t \leq 1$ second.

Time-Domain Representation

The signal is uniformly sampled and plotted in the time domain. The waveform corresponds to a cosine signal with frequency 15 Hz and a phase shift of $\pi/4$.

```
% Exercise 1.2(a): Time-domain representation

fs = 100;
Ts = 1/fs;
t = 0:Ts:1;
```

```
x = cos(30*pi*t + pi/4);

figure;
plot(t, x, 'LineWidth', 1.5);
grid on;
xlabel('Time (s)');
ylabel('x(t)');
```

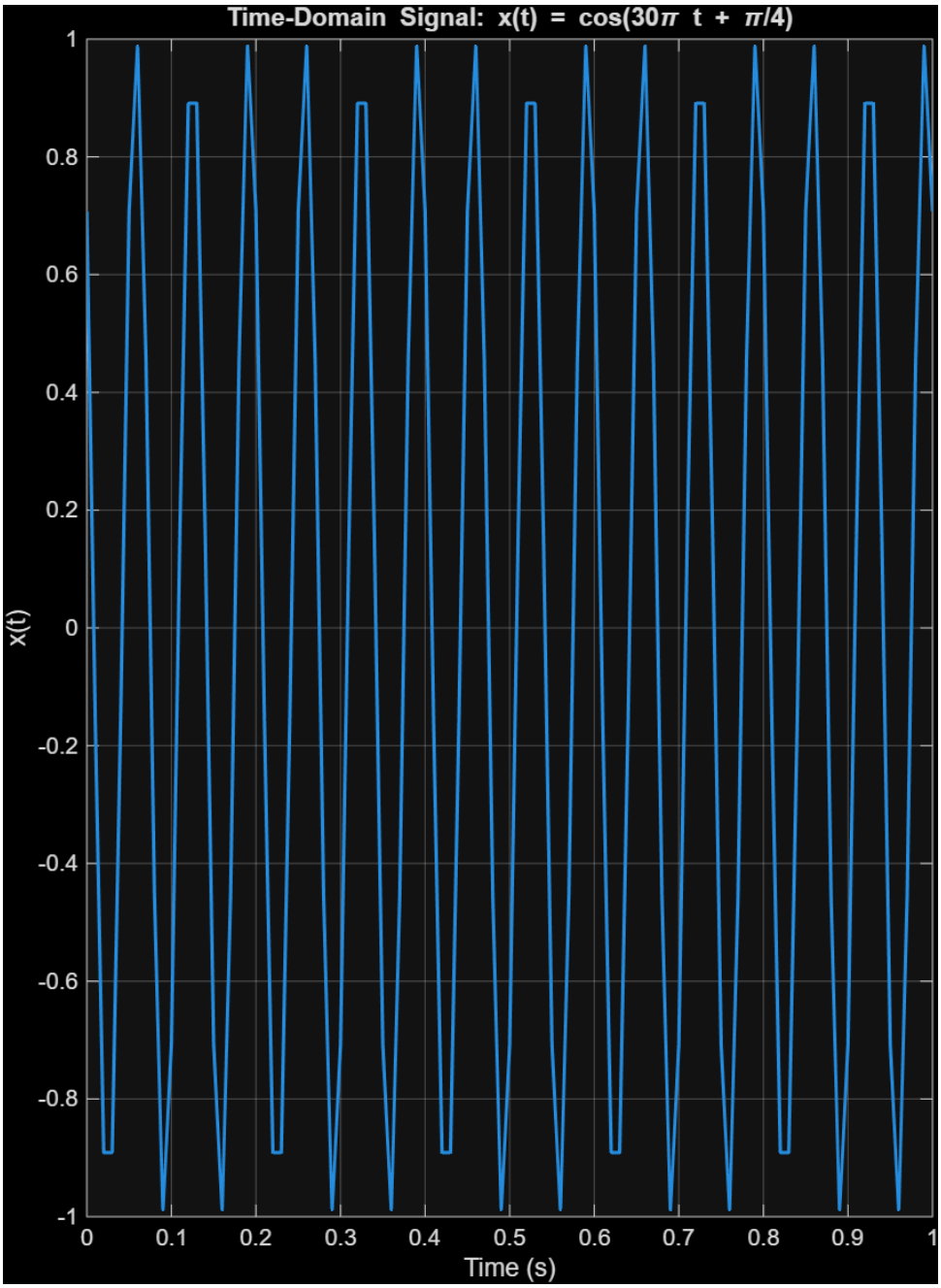

Figure 3: Time-domain representation of $x(t) = \cos(30\pi t + \pi/4)$.

Magnitude of the Fourier Transform

The discrete Fourier transform of the signal is computed using the FFT. The magnitude spectrum is normalized so that its maximum value equals one. As expected, two dominant peaks appear at $\pm 15\,\text{Hz}$, and the phase shift does not affect the magnitude spectrum.

```
% Exercise 1.2(b): Magnitude spectrum

N = length(x);
X = fftshift(fft(x));
f = (-N/2:N/2-1) * (fs/N);

X_mag = abs(X);
X_mag = X_mag / max(X_mag);

figure;
plot(f, X_mag, 'LineWidth', 1.5);
grid on;
xlabel('Frequency (Hz)');
ylabel('|X(f)| (normalized)');
```

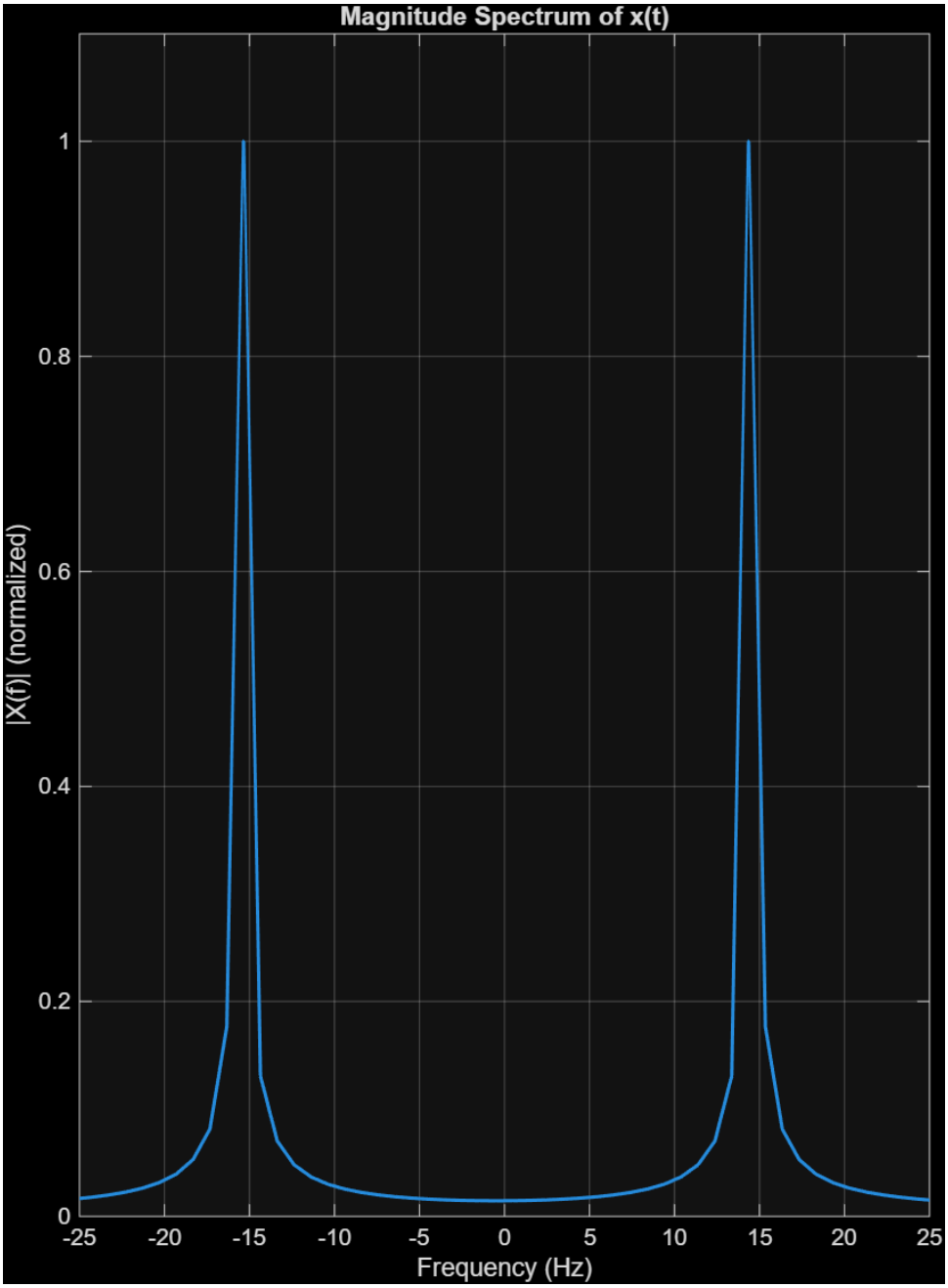

Figure 4: Normalized magnitude spectrum of $x(t)$.

Phase of the Fourier Transform

To obtain a meaningful phase plot, phase values corresponding to negligible magnitudes are first set to zero. The phase is then plotted as a multiple of π.

```
% Exercise 1.2(c): Phase spectrum

tol = 1e-6;
X(abs(X) < tol) = 0;
theta = angle(X);
```

```
figure;
plot(f, theta/pi, 'LineWidth', 1.5);
grid on;
xlabel('Frequency (Hz)');
ylabel('Phase / \pi');
```

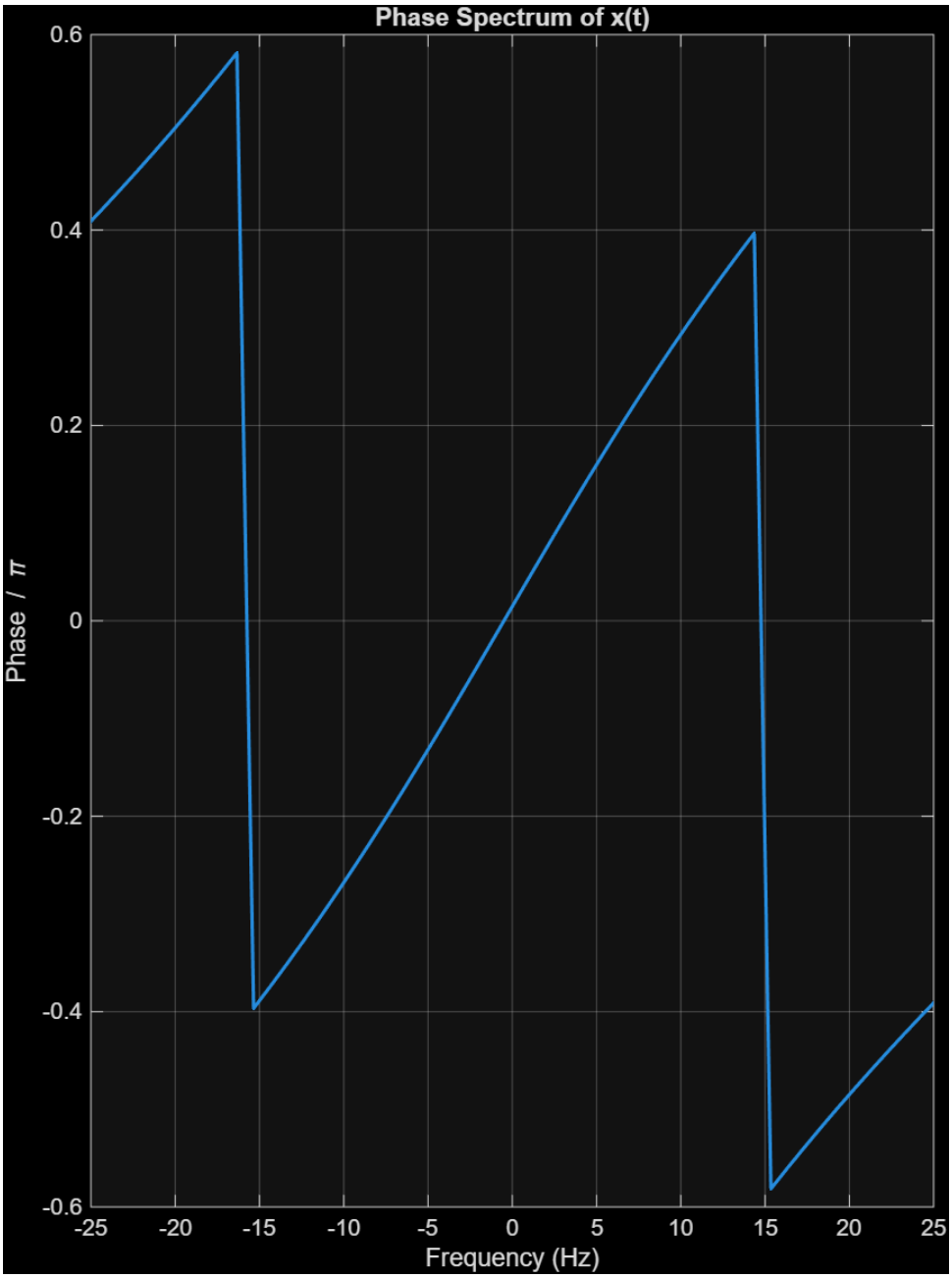

Figure 5: Phase spectrum of $x(t)$ shown as a multiple of π.

The phase values at $\pm 15\,\text{Hz}$ are approximately $\pm\pi/4$, which is fully consistent with the theoretical Fourier transform of a cosine signal with a phase shift of $\pi/4$.

Part 2: Frequency-Based Information Transmission

Exercise 2.1: Character-to-Bit Mapping (Mapset)

In this exercise, a fixed mapping between characters and binary sequences is defined. The allowed characters include 26 lowercase English letters, space, and the symbols . , ! ; " which results in a total of 32 characters. Each character is represented using a unique 5-bit binary code.

The mapping is stored in a 32×2 cell array named `Mapset`, where the first column contains the characters and the second column contains their corresponding 5-bit binary representations.

```
% Exercise 2.1: Define Mapset (32 characters -> 5-bit codes)

chars = ['a':'z', '␣', '.', ',', '!', ';', '"'];
codes = dec2bin(0:31, 5);

Mapset = cell(32, 2);
for i = 1:32
    Mapset{i,1} = chars(i);
    Mapset{i,2} = codes(i,:);
end

disp(Mapset);
```

This mapping is used in subsequent exercises for converting text messages into binary bitstreams prior to frequency-based encoding.

Exercise 2.2: Frequency-Based Encoding Function

In this exercise, a function named `freq_coding` is implemented to encode a textual message using frequency-based modulation. The function takes the input message and the desired bit rate (in bits per second) and generates the corresponding time-domain signal.

For a bit rate of R bits/sec, each symbol occupies one second and represents R bits. Therefore, 2^R distinct frequencies are required. Each group of R bits is mapped to one of these frequencies, and a sinusoidal signal with the selected frequency is transmitted for one second.

```
function tx_signal = freq_coding(message, bit_rate)
% Exercise 2.2: Frequency-based encoder

fs = 100;
```

```
Ts = 1/fs;
t_sym = 0:Ts:1-Ts;

chars = ['a':'z', '␣', '.', ',', '!', ';', '"'];
codes = dec2bin(0:31, 5);

bitstream = '';
for k = 1:length(message)
    idx = find(chars == message(k));
    bitstream = [bitstream codes(idx,:)];
end

num_symbols = floor(length(bitstream)/bit_rate);
bitstream = bitstream(1:num_symbols*bit_rate);

num_freqs = 2^bit_rate;
freqs = round(linspace(5, 49, num_freqs));

tx_signal = [];
for i = 1:num_symbols
    bits = bitstream((i-1)*bit_rate+1:i*bit_rate);
    idx = bin2dec(bits) + 1;
    f = freqs(idx);
    tx_signal = [tx_signal sin(2*pi*f*t_sym)];
end
end
```

This function forms the core of the frequency-based communication system used in the rest of this section.

Exercise 2.3: Encoding and Visualization of the Message `signal`

The implemented encoder is now tested by transmitting the word `signal` using two different bit rates: 1 bit/sec and 5 bits/sec. The resulting time-domain signals are plotted to illustrate the effect of increasing the bit rate.

Transmission with 1 Bit/sec

At a bit rate of 1 bit/sec, each bit is transmitted over one second using one of two frequencies. Since the word `signal` contains 6 characters and each character is represented by 5 bits, the total transmission duration is 30 seconds.

```
% Exercise 2.3: Encoding "signal" at 1 bit/sec

tx_1bps = freq_coding('signal', 1);
t1 = (0:length(tx_1bps)-1)/100;

figure;
plot(t1, tx_1bps);
grid on;
xlabel('Time (s)');
ylabel('Amplitude');
title('Frequency Coding of "signal" (1 bit/sec)');
```

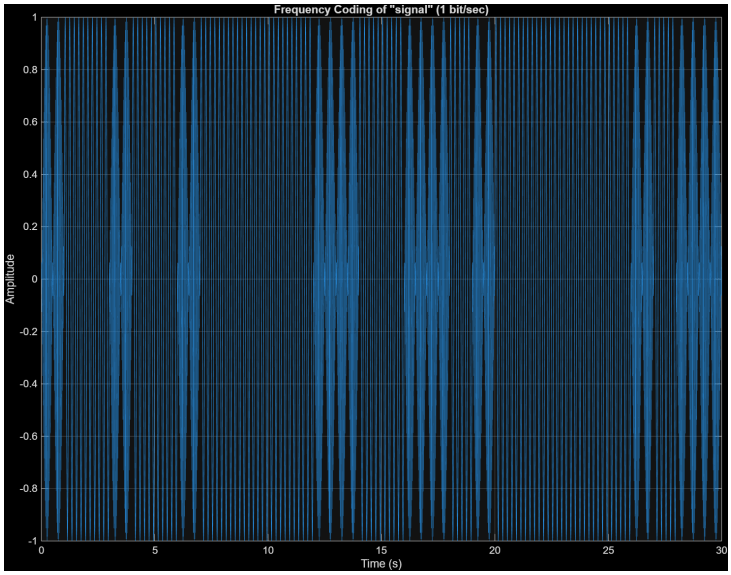

Figure 6: Frequency-coded signal for the word `signal` at 1 bit/sec.

Transmission with 5 Bits/sec

For a bit rate of 5 bits/sec, each one-second symbol represents 5 bits, requiring 32 distinct frequencies. In this case, the total transmission time is reduced to 6 seconds. The resulting waveform appears significantly denser in the time domain, making frequency-domain analysis necessary at the receiver.

```
% Exercise 2.3: Encoding "signal" at 5 bits/sec

tx_5bps = freq_coding('signal', 5);
t5 = (0:length(tx_5bps)-1)/100;

figure;
plot(t5, tx_5bps);
grid on;
```

```
xlabel('Time (s)');
ylabel('Amplitude');
title('Frequency Coding of "signal" (5 bits/sec)');
```

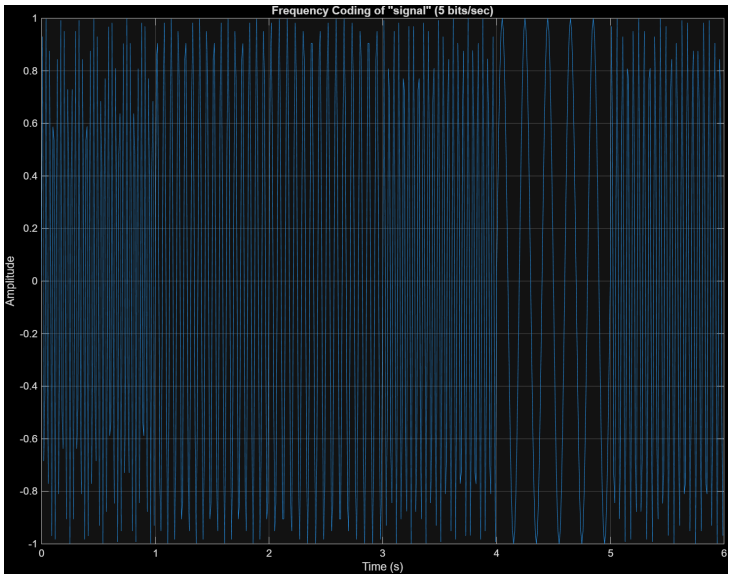

Figure 7: Frequency-coded signal for the word `signal` at 5 bits/sec.

These results demonstrate the trade-off between transmission speed and signal complexity. As the bit rate increases, the signal duration decreases, but the time-domain waveform becomes more complex, motivating the use of FFT-based decoding at the receiver.

Exercise 2.4: Frequency-Based Decoding Using FFT

In this exercise, a receiver-side decoding function is implemented to recover the transmitted message from the frequency-coded time-domain signal. The decoder uses the discrete Fourier transform to identify the dominant frequency in each one-second symbol interval and maps it back to the corresponding binary data.

Decoding Method

For a given bit rate of R bits/sec, each transmitted symbol spans one second and represents R bits of information. The decoding process consists of the following steps:

- Divide the received signal into one-second segments.
- Compute the FFT of each segment and apply `fftshift`.
- Consider only the positive frequency components.
- Identify the frequency with the maximum magnitude.

- Match the detected frequency to the closest predefined reference frequency.
- Convert the frequency index into an R-bit binary sequence.

The recovered bitstream is then grouped into 5-bit blocks and mapped back to characters using the predefined character-to-bit mapping.

MATLAB Implementation

The MATLAB function used for frequency-based decoding is given below.

```
function decoded_message = freq_decoding(rx_signal, bit_rate)
% Exercise 2.4: Frequency-based decoder

fs = 100;
samples_per_sym = fs;

chars = ['a':'z', '␣', '.', ',', '!', ';', '"'];
codes = dec2bin(0:31, 5);

num_freqs = 2^bit_rate;
freqs = round(linspace(5, 49, num_freqs));

num_symbols = floor(length(rx_signal)/samples_per_sym);
bitstream = '';

for k = 1:num_symbols
    segment = rx_signal((k-1)*samples_per_sym+1 : k*
       samples_per_sym);

    Y = fftshift(fft(segment));
    mag = abs(Y);

    f = (-samples_per_sym/2 : samples_per_sym/2-1) * (fs/
       samples_per_sym);

    pos_idx = f > 0;
    f_pos = f(pos_idx);
    mag_pos = mag(pos_idx);

    [~, idx_max] = max(mag_pos);
    detected_freq = f_pos(idx_max);
```

```
    [~, freq_idx] = min(abs(freqs - detected_freq));
    bits = dec2bin(freq_idx-1, bit_rate);

    bitstream = [bitstream bits];
end

decoded_message = '';
num_chars = floor(length(bitstream)/5);

for i = 1:num_chars
    bits = bitstream((i-1)*5+1 : i*5);
    idx = bin2dec(bits) + 1;
    decoded_message = [decoded_message chars(idx)];
end
end
```

Verification of Decoding

The decoding function was tested using the encoded version of the word `signal` generated in Exercise 2.3. The test was performed for both transmission rates of 1 bit/sec and 5 bits/sec. In both cases, the decoded output exactly matched the original transmitted message:

- Decoded message at 1 bit/sec: `signal`
- Decoded message at 5 bits/sec: `signal`

These results confirm that the frequency-based decoding approach works correctly and that the encoder and decoder are fully consistent.

Exercise 2.5: Decoding in the Presence of Additive Gaussian Noise

In this exercise, the effect of noise on the frequency-based communication system is examined. To simulate realistic channel conditions, zero-mean additive white Gaussian noise (AWGN) with variance 0.0001 is added to the transmitted signal before decoding.

The experiment is performed for both transmission rates of 1 bit/sec and 5 bits/sec using the message `signal`. After adding noise, the received signal is decoded using the FFT-based decoder implemented in Exercise 2.4.

MATLAB Implementation

The following MATLAB code was used to add noise to the transmitted signal and perform decoding at the receiver.

```
% Exercise 2.5: Decoding with additive Gaussian noise

msg = 'signal';
noise_var = 0.0001;
noise_std = sqrt(noise_var);

% ---- 1 bit/sec ----
tx_1bps = freq_coding(msg, 1);
noise_1bps = noise_std * randn(size(tx_1bps));
rx_1bps = tx_1bps + noise_1bps;

decoded_1bps = freq_decoding(rx_1bps, 1);

% ---- 5 bits/sec ----
tx_5bps = freq_coding(msg, 5);
noise_5bps = noise_std * randn(size(tx_5bps));
rx_5bps = tx_5bps + noise_5bps;

decoded_5bps = freq_decoding(rx_5bps, 5);
```

Decoding Results

For both transmission rates, the decoded output exactly matched the original transmitted message:

- Decoded message at 1 bit/sec: `signal`
- Decoded message at 5 bits/sec: `signal`

These results indicate that for a low noise variance of 0.0001, the frequency separation between the selected tones is sufficient to allow correct frequency detection using the FFT, even at the higher transmission rate.

This experiment also establishes a baseline for subsequent exercises, where the noise power will be gradually increased to study the robustness of the system and compare the performance of different transmission rates.

Exercise 2.6: Effect of Increasing Noise Power

In this exercise, the robustness of the frequency-based communication system is further investigated by gradually increasing the noise power and repeating the decoding process. Zero-mean additive white Gaussian noise (AWGN) with increasing variance is added to the transmitted signal, and the decoded message is examined for two different transmission rates: 1 bit/sec and 5 bits/sec.

MATLAB Implementation

The following MATLAB code was used to perform the noise power sweep and observe the decoding results at different noise levels.

```
% Exercise 2.6: Noise robustness comparison

msg = 'signal';
noise_vars = [0.01 0.1 0.5 1 1.5 2 3 5];

disp('--- Noise Robustness Test ---');

for nv = noise_vars
    noise_std = sqrt(nv);

    % ---- 1 bit/sec ----
    tx_1bps = freq_coding(msg, 1);
    rx_1bps = tx_1bps + noise_std * randn(size(tx_1bps));
    dec_1bps = freq_decoding(rx_1bps, 1);

    % ---- 5 bits/sec ----
    tx_5bps = freq_coding(msg, 5);
    rx_5bps = tx_5bps + noise_std * randn(size(tx_5bps));
    dec_5bps = freq_decoding(rx_5bps, 5);

    fprintf('\nNoise variance = %.4f\n', nv);
    fprintf('  1 bit/sec decoded: %s\n', dec_1bps);
    fprintf('  5 bits/sec decoded: %s\n', dec_5bps);
end
```

Observed Results

A representative outcome of the simulation is summarized as follows:

- For noise variances up to approximately 2, both transmission rates were able to correctly decode the message `signal`.
- At a noise variance of approximately 3, decoding errors began to appear for the 5 bits/sec transmission, while the 1 bit/sec transmission was still decoded correctly.
- At higher noise levels (around 5), both transmission rates failed and the decoded messages were significantly corrupted.

Discussion

From a theoretical perspective, lower bit rates are expected to be more robust against noise because they use fewer and more widely separated frequencies. The simulation results are consistent with this expectation when considering overall trends rather than individual noise realizations.

It should be noted that due to the long symbol duration (1 second) and the use of FFT-based frequency detection, both transmission rates exhibited similar robustness at low and moderate noise levels. As the noise variance increased further, the higher bit rate began to fail earlier, while at very high noise levels both systems became unreliable.

Overall, the results confirm that increasing the bit rate leads to higher sensitivity to noise, which is consistent with the discussion presented in the introduction of this assignment.

Exercise 2.7: Maximum Tolerable Noise Variance

Based on the simulations performed in Exercise 2.6, an approximate estimate of the maximum noise variance that each transmission rate can tolerate before decoding failure can be obtained.

From the observed results, both transmission rates were able to correctly decode the message `signal` for noise variances up to approximately 2. As the noise variance increased beyond this level, decoding errors began to appear.

In particular:

- For the transmission rate of 5 bits/sec, decoding errors typically appeared at noise variances around 3.
- For the transmission rate of 1 bit/sec, correct decoding was generally maintained up to slightly higher noise variances, with failures occurring at larger noise powers.

These results indicate that, although both transmission rates show strong robustness due to long symbol duration and FFT averaging, the lower bit rate is able to tolerate higher noise power before failure.

Exercise 2.8: Improving Noise Robustness

A straightforward and effective method to improve the noise robustness of frequency-based coding is to increase the separation between the selected frequencies. When the distance between frequencies is larger, noise-induced fluctuations are less likely to cause incorrect frequency detection at the receiver.

However, increasing frequency separation requires a larger occupied bandwidth. As a result, there exists a trade-off between bandwidth usage, transmission speed, and noise robustness. By consuming more bandwidth, it is possible to transmit information at higher rates while maintaining robustness against noise.

This principle is directly related to the well-known concept that increasing available bandwidth enables higher data rates in communication systems, such as in modern internet technologies.

Exercise 2.9: Effect of Sampling Rate Without Increasing Bandwidth

Increasing the sampling rate alone, without increasing the occupied bandwidth, does not significantly improve noise robustness. While a higher sampling rate provides more samples per symbol and may improve numerical accuracy, it does not increase the frequency separation between the modulation tones.

Since the primary factor affecting noise robustness in this system is the distance between the selected frequencies, increasing the sampling rate without expanding the bandwidth does not fundamentally improve the ability of the receiver to distinguish between frequencies under noisy conditions.

Therefore, improving noise robustness requires increased bandwidth usage rather than merely a higher sampling rate.