Sharif University of Technology
Department of Electrical Engineering

Communication Systems
Computer Assignment 1
Instructor: Dr. Behroozi
TA Contact: @YourTelegramID

Introduction

This assignment will guide you through the fundamental concepts of signals, systems, and noise. Using MATLAB , you will investigate how signals are represented in time and frequency, how cross-correlation can be used to identify unknown systems, and how filters can be designed to remove noise and even reverse channel distortion.

Instructions:

- For all plots, you must clearly label your axes (e.g., Time (s), Frequency (Hz), Amplitude, Power/Frequency (dB/Hz)).
- Provide your commented code and all resulting plots in your final submission.
- Provide a final pdf report and upload that in cw as well.
- Answer all explanatory questions clearly and concisely.

---

1 Question 1: System Identification using Cross-Correlation

A powerful use of cross-correlation is "system identification." By sending a known "probe" signal $x[n]$ into an unknown linear time-invariant (LTI) system $h[n]$, we can determine the system's impulse response $h[n]$ by analyzing the output $y[n]$. This is used in everything from measuring room acoustics to network channel estimation.

We will use a special probe signal called a "linear chirp," which is a sine wave whose frequency increases over time. A chirp's auto-correlation is a very sharp spike, making it excellent for this purpose.

(a) **Create the Probe Signal:** Create a 1-second linear chirp signal $x[n]$ sampled at $f_s = 8000$ Hz. The frequency should sweep from $f_{start} = 100$ Hz to $f_{end} = 4000$ Hz over its duration.

(b) **Analyze the Probe:** Compute and plot the **auto-correlation** $R_{xx}[k]$ of the chirp signal $x[n]$. Plot it versus lag k in samples. What property do you observe at $k = 0$? (This property is why chirps are useful).

(c) **Simulate the "Room":** We will send this chirp into an "unknown" system $h[n]$ (a simulated room with echoes). The received signal $y[n]$ is the convolution of the chirp $x[n]$ and the room's impulse response $h[n]$, plus some noise:

$$y[n] = (x * h)[n] + w[n]$$

Use the (unknown to you, for now) impulse response $h[n]$ and noise $w[n]$ provided in the file `room_data.mat` (or `room_data.npz`). Load this file to get $y[n]$ and $x[n]$ (the $x[n]$ is provided to ensure you have the exact same one).

(d) **Time-Domain Plot:** Plot the received signal $y[n]$ in the time domain. Does it look like the original chirp?

(e) **Find the Impulse Response:** Compute the **cross-correlation** $R_{yx}[k] = \sum_m y[m]x[m-k]$. (Note: $x[m-k]$ is $x[-n]$ shifted by k, so this is $y[k] * x[-k]$).

(f) **Analysis:** Plot the computed cross-correlation $R_{yx}[k]$ versus lag k (in samples). You should see several distinct peaks.

(g) **Conclusion:** Recall that $R_{yx}[k] = R_{hx}[k] * R_{xx}[k]$. Given your observation from part (b) (where $R_{xx}[k] \approx \delta[k]$), what does the plot of $R_{yx}[k]$ you just made actually represent?

(h) Based on your plot of $R_{yx}[k]$, state the sample delays k and relative amplitudes of the three strongest "echoes" (i.e., the impulse response $h[n]$) you have found in the "room".

---

2 Question 2: Forensic Audio Cleanup: Spectral Analysis and Filtering

You are given a "corrupted" audio file, `corrupted_speech.wav`. This file contains a speech signal, but it is contaminated with two types of noise:

1. Wideband "hiss" (Additive White Gaussian Noise, AWGN).
2. A persistent, high-frequency "whine" (a sinusoidal interference).

Your goal is to use spectral analysis and filtering to remove both types of noise.

(a) **Load and Plot:** Load `corrupted_speech.wav`. Let the signal be $y[n]$ and note the sampling frequency f_s. Plot the signal $y[n]$ in the time domain.

(b) **Spectral Analysis:** Compute and plot the **Power Spectral Density (PSD)** of $y[n]$. Plot it in dB/Hz versus frequency in Hz (from 0 Hz to $f_s/2$).

(c) **Identify Interference:** Look at your PSD plot.

- What is the approximate frequency f_{whine} of the sharp "whine" (the large spike)?

- How does the AWGN "hiss" manifest itself in this plot?
- In what approximate frequency range (e.g., f_{min} to f_{max}) does most of the speech signal's energy appear to be?

(d) **Filter 1: Notch Filter:** Design a **Notch Filter** (a band-stop filter) to remove the f_{whine} you identified. (In MATLAB, see `designfilt` with 'bandstop'). Apply this filter to $y[n]$ to get a new signal, $y_{notched}[n]$.

(e) **Filter 2: Low-Pass Filter:** Now, design a **Low-Pass FIR Filter** to remove the high-frequency "hiss". Choose a cutoff frequency f_{cut} based on your answer in (c) that preserves the speech energy while cutting off as much noise as possible. Apply this LPF to $y_{notched}[n]$ to get the final, clean signal $z[n]$.

(f) **Verification (PSD):** Plot the PSD of the final signal $z[n]$. Compare it to the PSD from part (b). Have the whine and the high-frequency hiss been successfully removed?

(g) **Verification (Spectrogram):** Plot the **spectrogram** of the original noisy signal $y[n]$ and the final clean signal $z[n]$ side-by-side. Describe the visual differences.

(h) **Listen:** If you have audio enabled, play the original $y[n]$ and the cleaned $z[n]$. Describe the audible improvement.

---

3 Question 3: Channel Equalization (Inverse Filtering)

Sometimes, a signal is not just corrupted by noise, but is *distorted* by the channel it passes through. For example, a cheap microphone or a long cable might act as a low-pass filter, "muffling" the sound by attenuating the high frequencies. Our goal is to "equalize" the signal by designing an *inverse filter* $g[n]$ that attempts to undo the channel's effect $h[n]$.

(a) **Load Original Signal:** Load your *own* voice recording (e.g., "Hello world") from the file `my_voice.wav`. Let this original, clean signal be $x[n]$. Note its sampling frequency f_s.

(b) **The Muffling Channel:** We are told the signal was distorted by a simple FIR channel:

$$h[n] = 0.6\delta[n] + 0.3\delta[n-1] + 0.1\delta[n-2]$$

Compute and plot the frequency response $|H(f)|$ of this channel. (Hint: use `scipy.signal.freqz` or `freqz` in MATLAB). What kind of filter is this?

(c) **Create Distorted Signal:** Create the muffled signal $y[n] = x[n] * h[n]$.

(d) **Compare Spectrograms:** Plot the spectrogram of the original $x[n]$ and the muffled $y[n]$ side-by-side. What key difference do you observe, especially at high frequencies?

(e) **The "Equalizer" Goal:** We want to find an equalizer filter $g[n]$ such that $x_{rec}[n] = y[n] * g[n] \approx x[n]$. In the frequency domain, this means $Y(f) \cdot G(f) \approx X(f)$, or $(X(f) \cdot H(f)) \cdot G(f) \approx X(f)$.

This implies the ideal equalizer $G(f)$ should be the *inverse* of the channel:

$$G_{ideal}(f) = \frac{1}{H(f)}$$

(f) **Equalizer Design:** We cannot implement the ideal inverse perfectly. Instead, we will design an FIR filter $g[n]$ that *approximates* this inverse response.

- Define a vector of frequencies f from 0 to $f_s/2$.
- Compute $H(f)$ at these frequencies.
- Create the desired inverse response $G_{desired}(f) = 1.0/H(f)$.
- **Important:** The high-frequency gain of $G_{desired}$ will be huge (since $H(f)$ is small). This amplifies noise. We must "clip" or "regularize" it. Use: `G_desired = 1.0 / (H(f) + 0.1)`. This prevents division by small numbers.
- Use an FIR filter design function (like `scipy.signal.firls` or `firls` in MATLAB) to find the taps $g[n]$ for a 51-tap filter that best matches the magnitude of $G_{desired}(f)$.

(g) **Apply Equalizer:** Apply your designed equalizer $g[n]$ to the muffled signal: $x_{rec}[n] = y[n] * g[n]$. (Use `mode='same'` for convolution).

(h) **Final Comparison:** Plot the spectrograms of the original $x[n]$, the muffled $y[n]$, and the "recovered" $x_{rec}[n]$ side-by-side. Did your equalizer succeed in restoring the high-frequency content? What side-effects, if any, do you notice? (e.g., in terms of ringing or noise).

---

4 Question 4: Sampling Theorem, Aliasing, and DTMF Decoding

The Nyquist-Shannon sampling theorem is the foundation of all digital communications. It states that to perfectly reconstruct a signal, the sampling frequency f_s must be at least twice the maximum frequency in the signal ($f_s > 2f_{max}$). If this rule is broken, "aliasing" occurs.

(a) **The Signal:** Create a 1-second continuous-time signal $x(t)$ (represented by a very high $f_s = 50000$ Hz) containing a single tone at $f_0 = 300$ Hz:

$$x(t) = \cos(2\pi \cdot 300t)$$

Plot its magnitude spectrum $|X(f)|$ from -5000 Hz to 5000 Hz. You should see two sharp peaks at ± 300 Hz.

(b) **Case 1: "Good" Sampling (Nyquist is Met):** Sample $x(t)$ with a sampling frequency $f_s = 1000$ Hz. (Here, $f_s > 2f_0$). Let this be $x_1[n]$. (Hint: You can "sample" by taking every 50th point of $x(t)$).

(c) **Spectrum of Good Sampling:** Compute and plot the magnitude spectrum $|X_1(f)|$ of the *sampled* signal $x_1[n]$. The x-axis should go from $-f_s/2$ to $f_s/2$ (i.e., -500 Hz to 500 Hz). Where are the peaks? Do they match the original signal's frequency?

(d) **Case 2: "Bad" Sampling (Aliasing):** Now, sample the original $x(t)$ with a sampling frequency $f_s = 400$ Hz. (Here, $f_s < 2f_0$, so the Nyquist theorem is violated). Let this be $x_2[n]$. (Hint: Take every 125th point of $x(t)$).

(e) **Spectrum of Bad Sampling:** Compute and plot the magnitude spectrum $|X_2(f)|$ of the sampled signal $x_2[n]$. The x-axis should go from $-f_s/2$ to $f_s/2$ (i.e., -200 Hz to 200 Hz). Where are the peaks now?

(f) **Analysis:** The peaks in (e) are at the "aliased" frequency, f_{alias}. The alias appears at a frequency of $f_{alias} = |f_0 - f_s| = |300 - 400| = 100$ Hz. Does your plot from (e) show peaks at ± 100 Hz?

(g) **Application: DTMF Decoding:** This principle is used in telephone "touch-tone" dialing (DTMF). The system was cleverly designed to be robust against noise. Instead of using one tone per button (which could be accidentally triggered by voice), each button press generates a *unique signal by adding two pure sine waves*:

- One from a "low frequency" group (corresponding to the button's **row**).
- One from a "high frequency" group (corresponding to the button's **column**).

This creates a 4x3 grid where each button is uniquely identified by its frequency pair, as shown in the table below. You are given the file `dtmf_sequence.wav`, which contains a sequence of these button presses sampled correctly at $f_s = 8000$ Hz. Your task is to act as a decoder and find the hidden sequence.

(h) **Decode the Tones:** Plot the **spectrogram** of the DTMF signal. A spectrogram is the perfect tool for this job because it shows you exactly which frequencies are active (i.e., "on") at any given point in time.

You will see a series of vertical blocks corresponding to each button press. Within each block, you should see **two** distinct bright horizontal lines—one for the row frequency and one for the column frequency.

By reading the (approximate) frequencies of these two lines from the spectrogram's y-axis, you can decode the button. Use the standard DTMF keypad chart below for reference.

Frequencies	**1209 Hz**	**1336 Hz**	**1477 Hz**
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

(e.g., "The first tone has peaks at 697 Hz and 1209 Hz, which corresponds to the '1' key."). Look at your spectrogram, read the frequency pairs for each tone, and write down the full sequence that was pressed.