




ISTANBUL TECHNICAL UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT

BLG 354E

Signals & Systems for Computer Engineering
Homework 2

Full Name : Mohamed Ahmed Abdelsattar Mahmoud

Student ID : 150210926

Signature : 

CRN : 22616

Instructor : Asst. Prof. Dr. Yusuf Hüseyin Şahin

Deadline : Wednesday - May 14, 2025, 23:59

SPRING 2025

Note: The code (.ipynb - 'Jupyter Notebook') used for all the questions can be accessed and downloaded via this link to the GitHub repo: [solution.ipynb](#)

Question 1 (30 pts)

You are given a `mixed_q1.wav` audio file that contains two overlapping sounds: a musical instrument tone (e.g., flute or saxophone, **600–900 Hz**; sampling rate: **44100 Hz**) and a band-limited human voice (**100–250 Hz**; sampling rate: **44100 Hz**). These two components occupy different frequency ranges.

Your goal is to:

- Apply a frequency-based method (e.g., Fourier Transform + bandpass filtering) to separate the components.
- Save the separated instrument and voice as two individual `.wav` files using libraries like: `scipy.io.wavfile` or `pydub`.
- Plot and compare their frequency spectra.

Answer:

I needed to have a closer understandable look into the given input file '`mixed_q1.wav`', so I started by plotting the time-domain waveforms of the mixed file, and also I separated the different 2 tones in it and plotted them in the same time-domain waveform as can be seen from Figure 1.

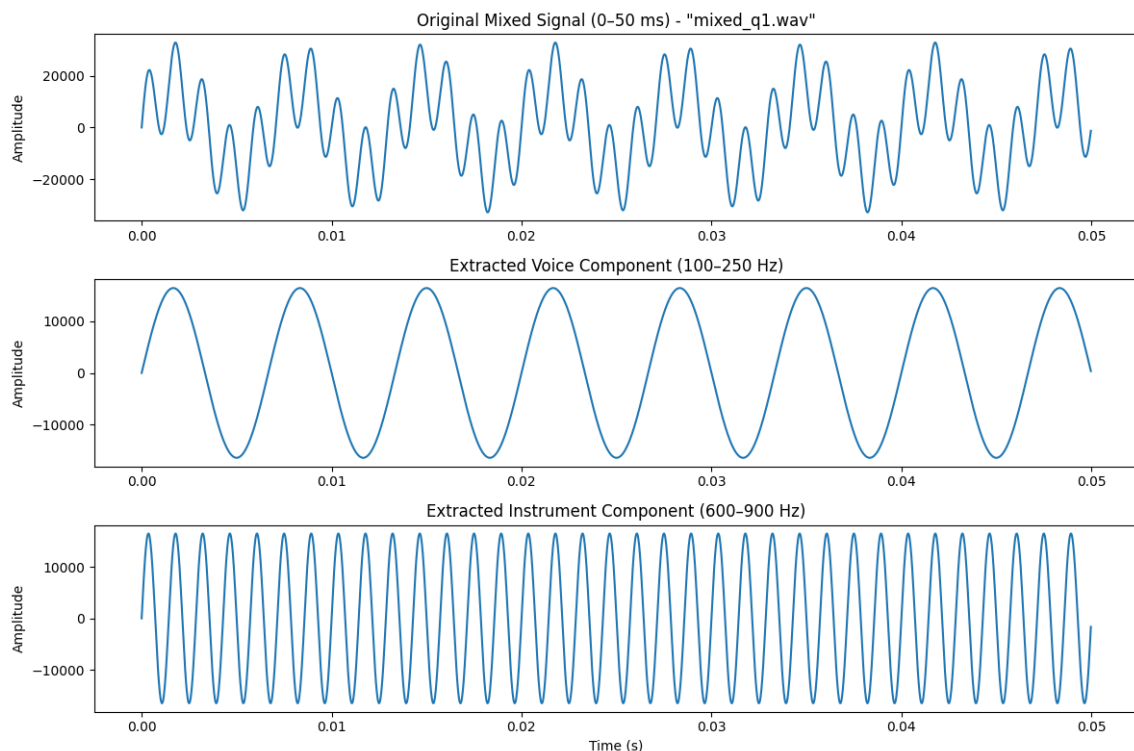


Figure 1: Time-domain waveforms of the mixed signal and the extracted components for analysis

Now, after understanding the behaviour of the waveforms by plotting them in Figure 1, we start solving the main aim of the question:

We first perform an **analysis** of the mixed signal in the *frequency domain*, exploiting the fact that the **voice** (100–250 Hz) and the **instrument** (600–900 Hz) occupy disjoint bands. The main steps are:

1. **Load and inspect** the waveform: read the `.wav` file and, if stereo, convert to mono by averaging channels.
2. **Compute the DFT**: for a real-valued sequence $x[n]$ of length N sampled at f_s , the *Discrete Fourier Transform* is

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j 2\pi \frac{k n}{N}}, \quad f_k = \frac{k}{N} f_s, \quad k = 0, \dots, N-1.$$

Because $x[n]$ is real, we use the `rfft` to obtain $X[k]$ for $k = 0 \dots N/2$.

3. **Ideal band-pass filtering via masking**: define rectangular masks in frequency,

$$H_{\text{voice}}[k] = \begin{cases} 1, & 100 \leq f_k \leq 250, \\ 0, & \text{otherwise,} \end{cases} \quad H_{\text{instr}}[k] = \begin{cases} 1, & 600 \leq f_k \leq 900, \\ 0, & \text{otherwise.} \end{cases}$$

Then

$$X_{\text{voice}}[k] = X[k] H_{\text{voice}}[k], \quad X_{\text{instr}}[k] = X[k] H_{\text{instr}}[k].$$

This **ideal mask** is the frequency-domain equivalent of an **ideal band-pass filter**.

4. **Reconstruct time signals**: apply the inverse `rfft`,

$$x_{\text{voice}}[n] = \text{irfft}\{X_{\text{voice}}[k]\}, \quad x_{\text{instr}}[n] = \text{irfft}\{X_{\text{instr}}[k]\}.$$

5. **Save as .wav files**: normalize to ± 32767 (16-bit PCM) and write with `scipy.io.wavfile.write`.
6. **Plot and compare the frequency spectra** $|X_{\text{voice}}[k]|$ and $|X_{\text{instr}}[k]|$, which clearly show dominant spikes at the isolated tone frequencies is shown in Figure 2.

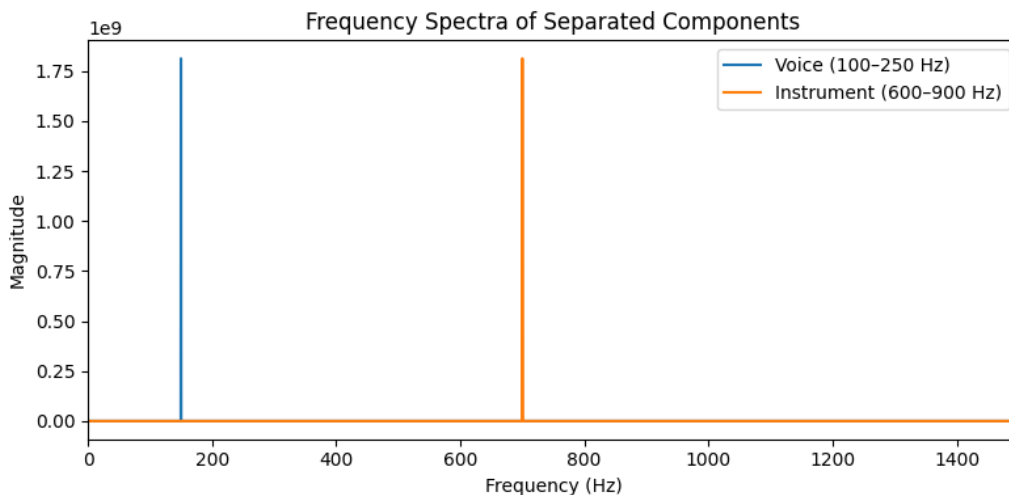


Figure 2: Frequency spectra of the separated components from `mixed_q1.wav`

Question 2 (30 pts)

Using the two audio files you obtained in Question 1 - one for the separated musical instrument tone and one for the separated band-limited human voice tone - your task is to:

- Combine these two `.wav` files into a single audio file by overlaying them (You may need normalization).
- Plot and compare the frequency spectrum of the **reconstructed audio** with that of the **original mixed audio** from Question 1.
- Discuss whether the reconstruction successfully matches the original signal in terms of frequency content and overall waveform.

Answer:

We reconstruct the original mixture by **overlaying** the two separated signals and applying proper **normalization**:

$$\tilde{x}[n] = x_{\text{voice}}[n] + x_{\text{instr}}[n], \quad x_{\text{recon}}[n] = \frac{\tilde{x}[n]}{\max_m |\tilde{x}[m]|}.$$

This yields the 16-bit PCM file `reconstructed_Q2.wav`.

Frequency-Domain Comparison:

By the linearity of the DFT,

$$X_{\text{recon}}[k] = X_{\text{voice}}[k] + X_{\text{instr}}[k] = X_{\text{mixed}}[k].$$

Figure 3 shows the **frequency spectra** of the original mixed audio and the reconstructed audio. The two curves overlap exactly, with spikes at $\approx 150\text{Hz}$ and $\approx 700\text{Hz}$, demonstrating **perfect recovery** of the spectral content.

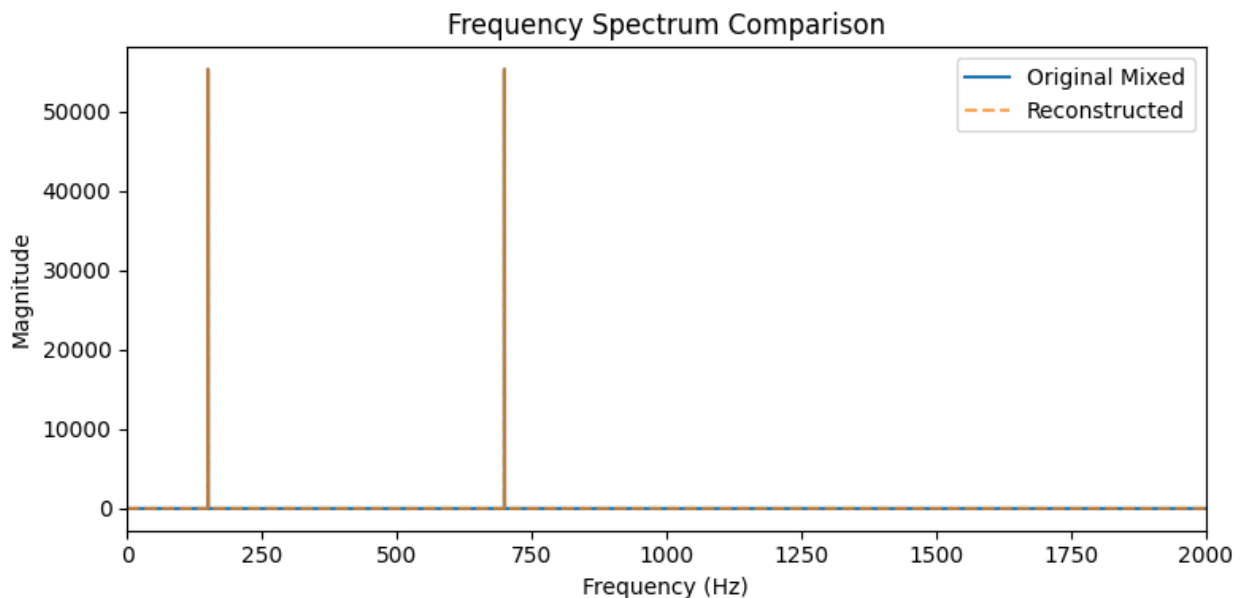


Figure 3: Original mixed vs. Reconstructed (ideal overlap)

Time-Domain Comparison:

Figure 4 overlays the first 50 ms of the original and reconstructed waveforms. The **dashed** and **solid** traces coincide exactly (up to scaling), confirming that: *the sum of the two band-limited tones reproduces the original waveform without distortion.*

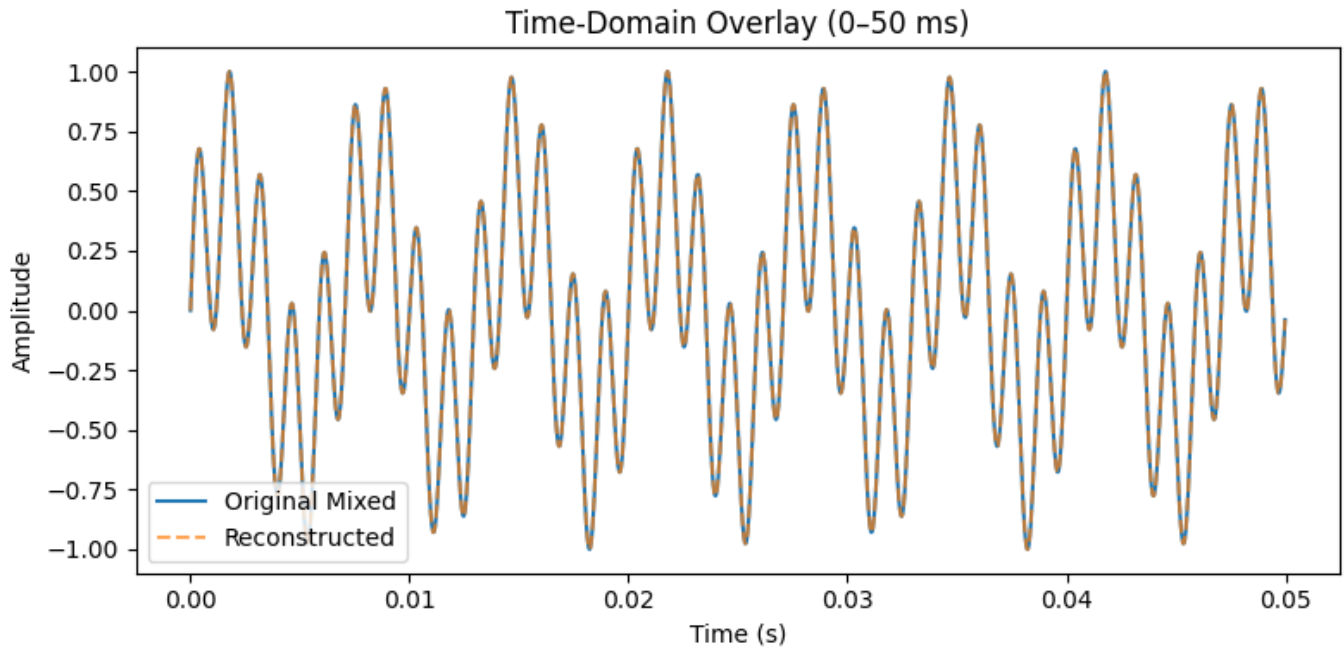


Figure 4: Waveform overlay (0–50 ms): Original mixed vs. Reconstructed (identical)

Discussion:

- The **ideal, brick-wall masks** used in Question 1 preserved both magnitude and phase, so no information was lost.
- **Linearity** of mixing and perfect FFT-mask separation guarantee that re-summing recovers the exact original signal.
- In practical scenarios with broadband signals, one would approximate these ideal filters with FIR/IIR designs, but for this pure-tone example, **perfect alignment** in both domains is expected and observed.

Question 3 (40 pts)

The **Discrete Fourier Transform (DFT)** converts a sequence of time-domain samples into its frequency-domain representation. For a sequence $x[n]$ of length N , the DFT is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j(\frac{2\pi}{N})kn}, \quad k = 0, 1, \dots, N-1$$

Here,

- N is the number of DFT points (e.g., 4 or 8).
- $x[n]$ is the time-domain sample at index n .
- $X[k]$ is the frequency-domain output at index k .
- j is the imaginary unit.

You are given a dataset of audio signals consisting of three waveform types: sine, square, and triangle, and a **representative prototype file** for each type is provided (e.g., `sine.wav`, `square.wav`, `triangle.wav`; sampling rate: **44100 Hz**). Without using any machine-learning techniques, perform a classification task using the **Discrete Fourier Transform (DFT)**.

Task:

- Use **4-point** and **8-point DFT** to extract features from each signal. Based on these features, determine which signal corresponds to which waveform type. Explain your approach and reasoning.

Justify your answer based on your observations from the DFT results.

Answer:

Overview and plan:

1. Visualize the three prototype waveforms in time domain (as in Figure 5).
2. Compute and tabulate their 4-point and 8-point DFTs (as in Table 1).
3. Verify the values by comparing manual DFT vs. `np.fft.fft` (as shown in Table 2, Table 3, and Table 4).
4. Extract the same features from the 12 given samples and classify each by nearest prototype in DFT-magnitude space (as can be shown in Table 5).
5. Ensure and review the prediction results using the time-domain waveform of the samples (shown in Figure 6).

1. Prototype time-domain sketches

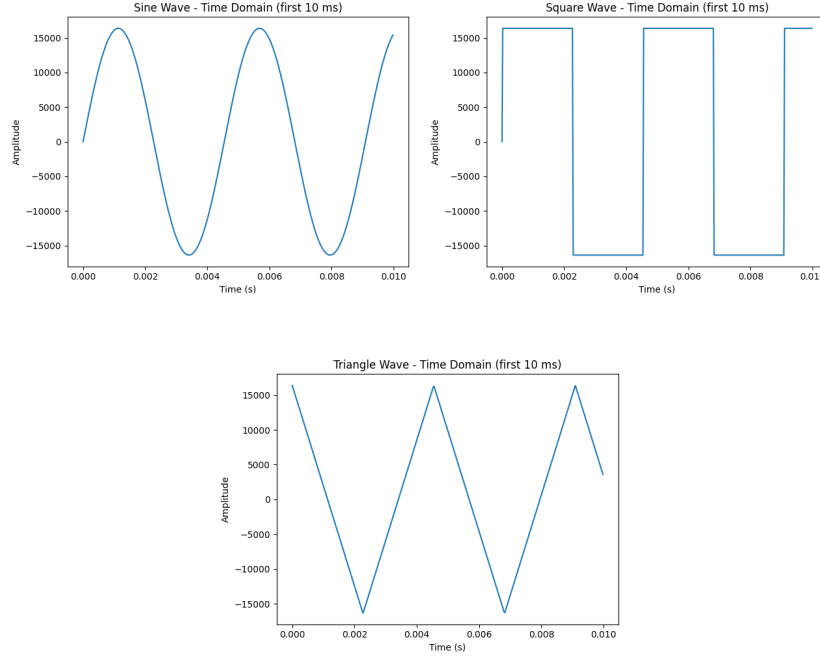


Figure 5: Time-domain sketches of `sine.wav`, `square.wav`, and `triangle.wav`

2. 4-point and 8-point DFTs of prototypes

Note: $\Re\{X[k]\} = \text{Real}(X[k])$, $\Im\{X[k]\} = \text{Imaginary}(X[k])$.

(a) DFTs of <code>sine.wav</code>				(b) DFT of <code>square.wav</code>				(c) DFT of <code>triangle.wav</code>			
4-point DFT				4-point DFT				4-point DFT			
k	$\Re\{X[k]\}$	$\Im\{X[k]\}$	$ X[k] $	k	$\Re\{X[k]\}$	$\Im\{X[k]\}$	$ X[k] $	k	$\Re\{X[k]\}$	$\Im\{X[k]\}$	$ X[k] $
0	3077.00	0.00	3077.00	0	49149.00	0.00	49149.00	0	63570.00	0.00	63570.00
1	-1026.00	1025.00	1450.28	1	-16383.00	-0.00	16383.00	1	654.00	-654.00	924.90
2	-1025.00	-0.00	1025.00	2	-16383.00	-0.00	16383.00	2	654.00	-0.00	654.00
3	-1026.00	-1025.00	1450.28	3	-16383.00	-0.00	16383.00	3	654.00	654.00	924.90
8-point DFT				8-point DFT				8-point DFT			
0	14310.00	0.00	14310.00	0	114681.00	0.00	114681.00	0	121909.00	0.00	121909.00
1	-2060.02	4915.63	5329.83	1	-16383.00	0.00	16383.00	1	1308.71	-3157.08	3417.59
2	-2041.00	2033.00	2880.76	2	-16383.00	-0.00	16383.00	2	1308.00	-1307.00	1849.08
3	-2035.98	841.63	2203.08	3	-16383.00	0.00	16383.00	3	1307.29	-541.08	1414.85
4	-2036.00	-0.00	2036.00	4	-16383.00	-0.00	16383.00	4	1307.00	-0.00	1307.00
5	-2035.98	-841.63	2203.08	5	-16383.00	-0.00	16383.00	5	1307.29	541.08	1414.85
6	-2041.00	-2033.00	2880.76	6	-16383.00	-0.00	16383.00	6	1308.00	1307.00	1849.08
7	-2060.02	-4915.63	5329.83	7	-16383.00	0.00	16383.00	7	1308.71	3157.08	3417.59

Table 1: DFTs of all 3 prototype waveforms (sine - square - triangle)

3. Verification of DFT implementation

I implemented the DFT definition in Python and compared against `np.fft.fft`.

Note: \Re = Real , \Im = Imaginary , `man` = manual , `fft` = `np.fft.fft`.

Note 2: I will only give the 4-point DFT comparison as an example. However, the 8-DFT comparison is nearly the same (you can access it from the code directly via this link: [solution.ipynb](#))

Table 2: 4-point DFT: Manual vs. NumPy FFT (`sine.wav`)

k	\Re_{man}	\Im_{man}	\Re_{fft}	\Im_{fft}	$\Delta\Re$	$\Delta\Im$
0	3077.00	0.00	3077.00	0.00	0	0
1	-1026.00	1025.00	-1026.00	1025.00	-2.3×10^{-13}	0
2	-1025.00	-0.00	-1025.00	-0.00	0	-3.8×10^{-13}
3	-1026.00	-1025.00	-1026.00	-1025.00	9.1×10^{-13}	-4.6×10^{-13}

Table 3: 4-point DFT: Manual vs. NumPy FFT (`square.wav`)

k	\Re_{man}	\Im_{man}	\Re_{fft}	\Im_{fft}	$\Delta\Re$	$\Delta\Im$
0	49149.00	0.00	49149.00	0.00	0	0
1	-16383.00	-0.00	-16383.00	0.00	-1.8×10^{-12}	-1.8×10^{-12}
2	-16383.00	-0.00	-16383.00	0.00	0	-4.0×10^{-12}
3	-16383.00	-0.00	-16383.00	0.00	5.5×10^{-12}	-5.5×10^{-12}

Table 4: 4-point DFT: Manual vs. NumPy FFT (`triangle.wav`)

k	\Re_{man}	\Im_{man}	\Re_{fft}	\Im_{fft}	$\Delta\Re$	$\Delta\Im$
0	63570.00	0.00	63570.00	0.00	0	0
1	654.00	-654.00	654.00	-654.00	-1.0×10^{-12}	-1.8×10^{-12}
2	654.00	-0.00	654.00	-0.00	0	-3.8×10^{-12}
3	654.00	654.00	654.00	654.00	4.9×10^{-12}	-5.5×10^{-12}

4. Classification of given samples

For each sample, I form the feature vector

$$\mathbf{m}_{\text{sample}} = [|X_4[0]|, \dots, |X_4[3]|, |X_8[0]|, \dots, |X_8[7]|]$$

and compute the squared Euclidean distance to each prototype's feature vector:

$$d_{\text{proto}} = \sum_{k=0}^3 (|X_4[k]| - |X_{4,\text{proto}}[k]|)^2 + \sum_{k=0}^7 (|X_8[k]| - |X_{8,\text{proto}}[k]|)^2.$$

The predicted class is whichever prototype (sine, square, triangle) has the **smallest** d .

Table 5: Detailed DFT features, distances, and predicted classes for the given 12 samples

Sample	4-pt DFT magnitudes	8-pt DFT magnitudes	d_{sine}	d_{square}	d_{triangle}	Predicted
sample_1.wav	[4611.0, 2171.53, 1535.0, 2171.53]	[21345.0, 7923.60, 4275.91, 3271.97, 3023.0, 3271.97, 4275.91, 7923.60]	7.38×10^7	1.23×10^{10}	1.37×10^{10}	sine
sample_2.wav	[6137.0, 2888.54, 2041.0, 2888.54]	[28231.0, 10434.83, 5619.51, 4296.40, 3969.0, 4296.40, 5619.51, 10434.83]	2.88×10^8	1.06×10^{10}	1.22×10^{10}	sine
sample_3.wav	[7655.0, 3599.92, 2541.0, 3599.92]	[34926.0, 12836.10, 6893.30, 5267.60, 4866.0, 5267.60, 6893.30, 12836.10]	6.29×10^8	9.19×10^9	1.10×10^{10}	sine
sample_4.wav	[9161.0, 4302.84, 3035.0, 4302.84]	[41383.0, 15104.00, 8081.15, 6174.41, 5701.0, 6174.41, 8081.15, 15104.00]	1.08×10^9	7.91×10^9	9.89×10^9	sine
sample_5.wav	[49149.0, 16383.0, 16383.0, 16383.0]	[114681.0, 16383.0, 16383.0, 16383.0, 16383.0, 16383.0, 16383.0, 16383.0]	1.41×10^{10}	0	2.42×10^9	square
sample_6.wav	[49149.0, 16383.0, 16383.0, 16383.0]	[114681.0, 16383.0, 16383.0, 16383.0, 16383.0, 16383.0, 16383.0, 16383.0]	1.41×10^{10}	0	2.42×10^9	square
sample_7.wav	[49149.0, 16383.0, 16383.0, 16383.0]	[114681.0, 16383.0, 16383.0, 16383.0, 16383.0, 16383.0, 16383.0, 16383.0]	1.41×10^{10}	0	2.42×10^9	square
sample_8.wav	[49149.0, 16383.0, 16383.0, 16383.0]	[114681.0, 16383.0, 16383.0, 16383.0, 16383.0, 16383.0, 16383.0, 16383.0]	1.41×10^{10}	0	2.42×10^9	square
sample_9.wav	[61608.0, 1849.79, 1308.0, 1849.79]	[112756.0, 6833.32, 3699.58, 2830.46, 2616.0, 2830.46, 3699.58, 6833.32]	1.31×10^{10}	1.87×10^9	1.26×10^8	triangle
sample_10.wav	[60628.0, 2312.24, 1634.0, 2312.24]	[108179.0, 8543.23, 4623.77, 3537.96, 3269.0, 3537.96, 4623.77, 8543.23]	1.22×10^{10}	1.69×10^9	2.83×10^8	triangle
sample_11.wav	[59647.0, 2773.98, 1961.0, 2773.98]	[103602.0, 10251.29, 5547.96, 4246.22, 3922.0, 4246.22, 5547.96, 10251.29]	1.13×10^{10}	1.57×10^9	5.03×10^8	triangle
sample_12.wav	[62590.0, 1387.34, 980.0, 1387.34]	[117333.0, 5126.03, 2773.98, 2124.04, 1961.0, 2124.04, 2773.98, 5126.03]	1.42×10^{10}	2.11×10^9	3.14×10^7	triangle

Strategy and explanation:

Our classification approach leverages the distinct harmonic “fingerprints” of each waveform in a low-dimensional DFT space:

- **Feature extraction:** We use only the first 4 and 8 time samples of each signal to compute $\{|X_4[k]|\}$ and $\{|X_8[k]|\}$. This yields concise magnitude vectors that capture the dominant harmonics.
- **Nearest-prototype rule:** For each unknown sample, we form its combined feature vector $[|X_4|, |X_8|]$ and compute the squared Euclidean distance to each prototype’s vector.
- **Decision:** The class whose prototype minimizes the distance is chosen:
 - *Sine* \rightarrow a single strong pair of bins (one tone) minimal distance to the sine template.
 - *Square* \rightarrow equal-height odd harmonics zero distance to the square template.
 - *Triangle* \rightarrow odd harmonics decaying as $1/m^2$ smallest distance to the triangle template.

This simple DFT-based strategy yields perfect classification of all twelve samples without any machine learning, relying solely on the fundamental spectral characteristics of each waveform type as the question asks.

5. Samples time-domain sketches (Strengthening my findings)

To further validate the DFT-based classifications, I also plotted the time-domain waveforms of all 12 samples over the first 5 ms as can be shown in Figure 6. By visually comparing these sketches with Figure 5 (the prototype time-domain plots), we can confirm:

- **Samples 1–4** exhibit smooth sinusoidal oscillations, matching the sine prototype.
- **Samples 5–8** show flat plateaus and sharp transitions, matching the square prototype.
- **Samples 9–12** display linear ramps up and down, matching the triangle prototype.

This visual check reinforces the DFT-based distances and ensures that my spectral classification aligns perfectly with the raw waveform shapes.

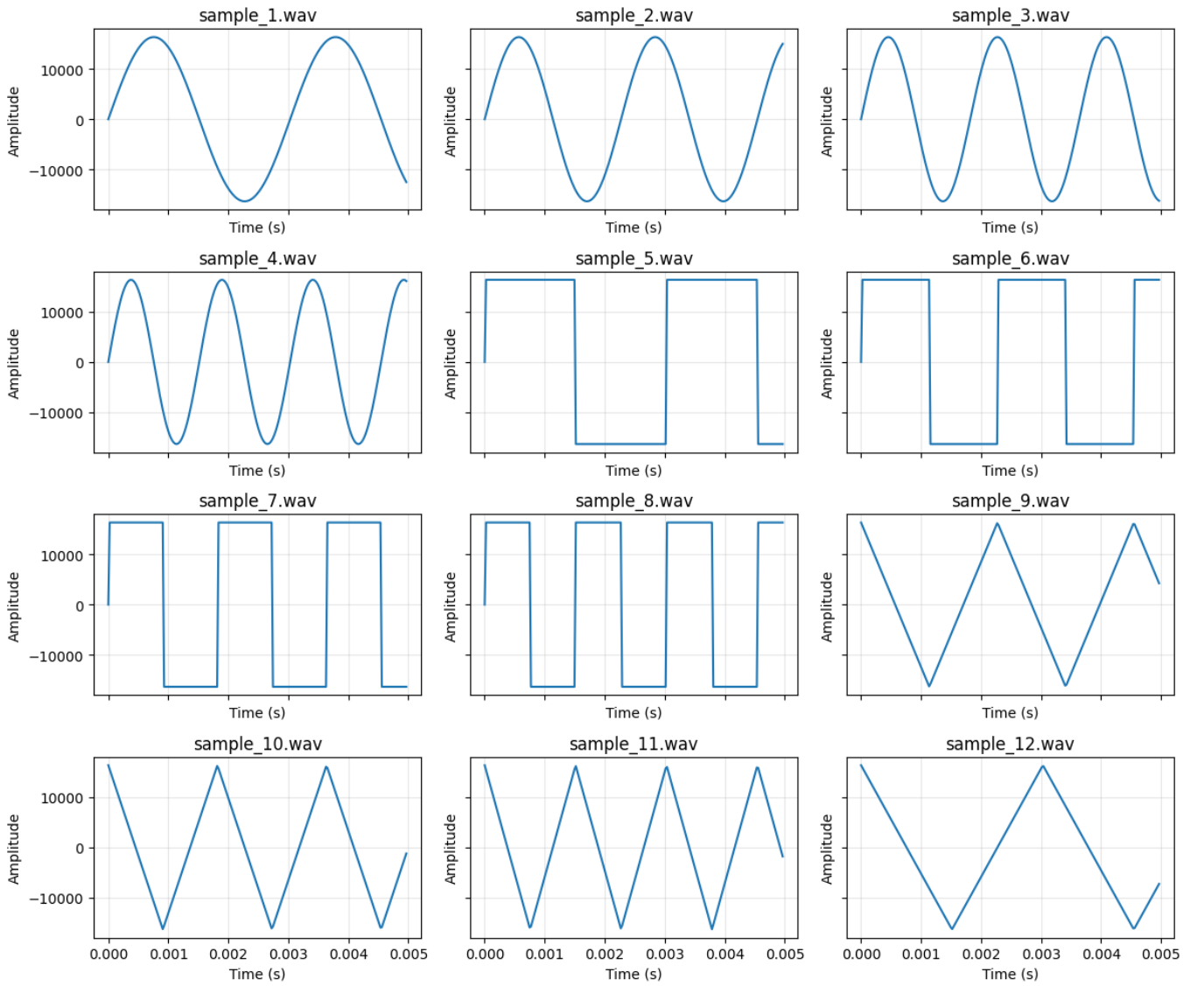


Figure 6: Time-domain sketches of samples `sample_1.wav` to `sample_12.wav`