Digital Signal Processing Project: Two-Tone Signal Analysis

Author: Raul-Anton JAC | **Group:** 1231EA | **Date:** 22.05.2025

1. Introduction

This report presents the analysis of a two-tone signal recorded and played back at different volume levels (10, 30, 50, 70, 90). The study explores signal behavior and intercept point extraction based on varying amplitude levels.

2. Signal Generation, playback and recording

I generated a two-tone signal using Python, composed of frequencies **800 Hz** and **1000 Hz**, sampled at **44.1 kHz** for a duration of **5 seconds**. The signal was played back on a laptop at **50% volume** and recorded using a mobile device. I also recorded the signal for another 4 distinct volume levels: 10%, 30%, 70%, 90%. So, we have 5 recorded files which will be used for intercept points and for analyze of the generated signal.

4. Intercept Point Extraction

This part of the project examines a set of audio recordings made at different volume levels to find where the waveform crosses the zero amplitude line. For each recording, it loads the audio data, converts it to a normalized format, and identifies the points where the signal changes sign, which correspond to zero crossings. The process is repeated for each recording, and the first three odd intercepts are printed for comparison.

The first three intercept points for each

- recorded_volume_10.wav: [2236 2264 2276]
- recorded_volume_30.wav: [2274 2285 2298]
- recorded_volume_50.wav: [2225 2232 2270]
- recorded_volume_70.wav: [2306 2318 2355]
- recorded_volume_90.wav: [2215 2271 2277]

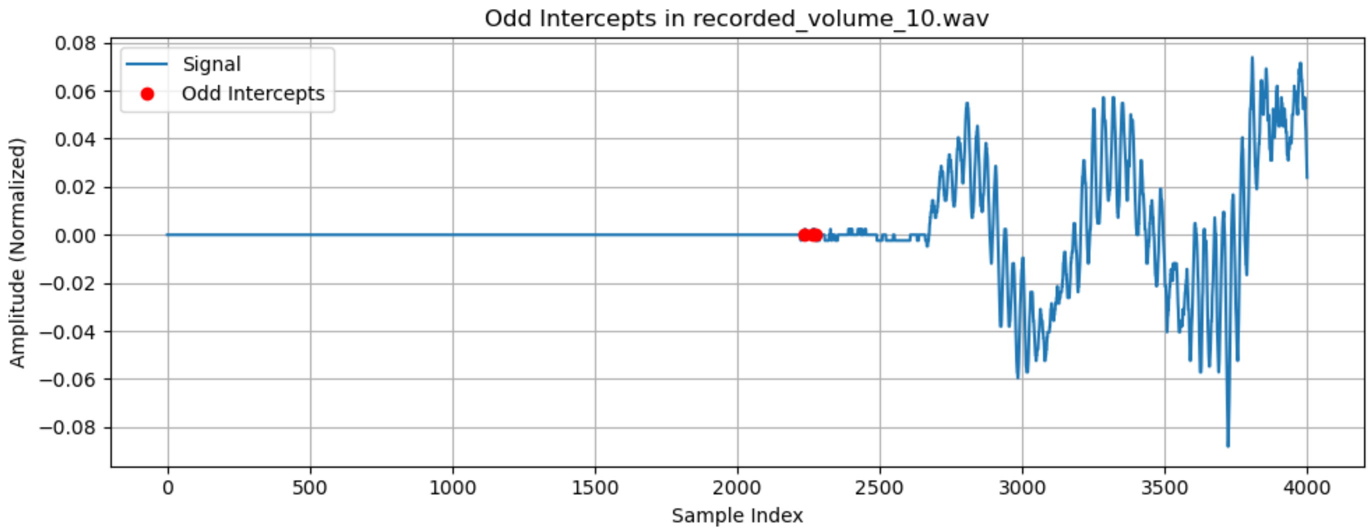
5. Signal analyze

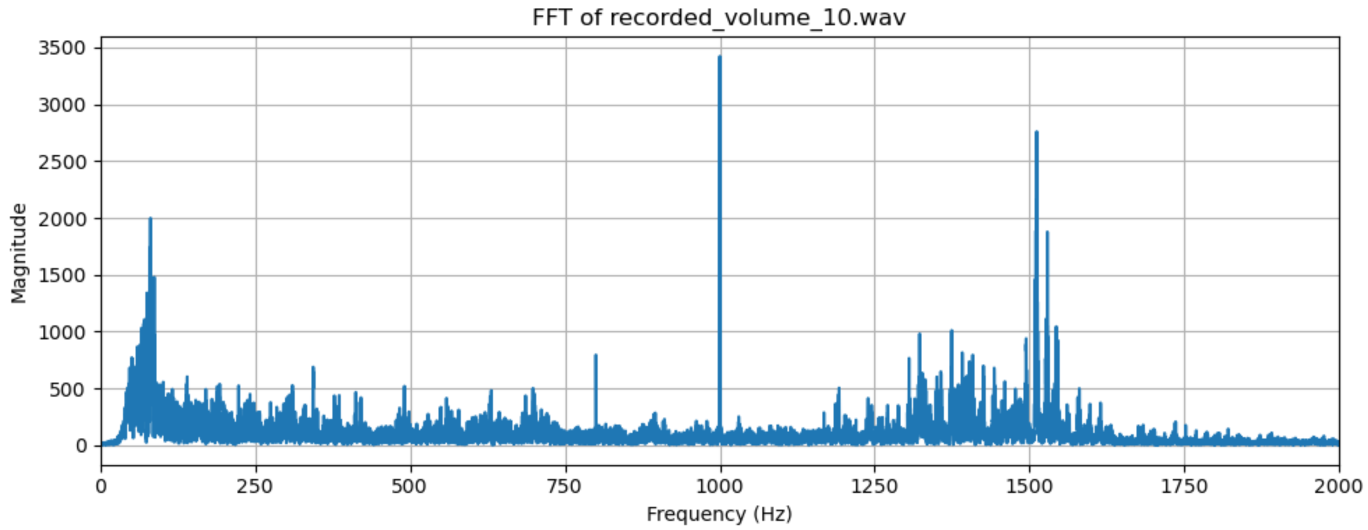
This part of the project analyzes audio files recorded at different volume levels to identify their dominant frequencies. For each audio file, it reads the waveform data, normalizes it, and computes its frequency spectrum using a Fast Fourier Transform. It then visualizes the frequency content in a graph, focusing on frequencies up to 2000 Hz, and prints out the most prominent frequency along with its magnitude. This process is repeated for several recordings, allowing comparison of their frequency characteristics.

7. Plots of intercept points and signal analyze

For a good image of each recorded signal, I decided to take each one and explain it.

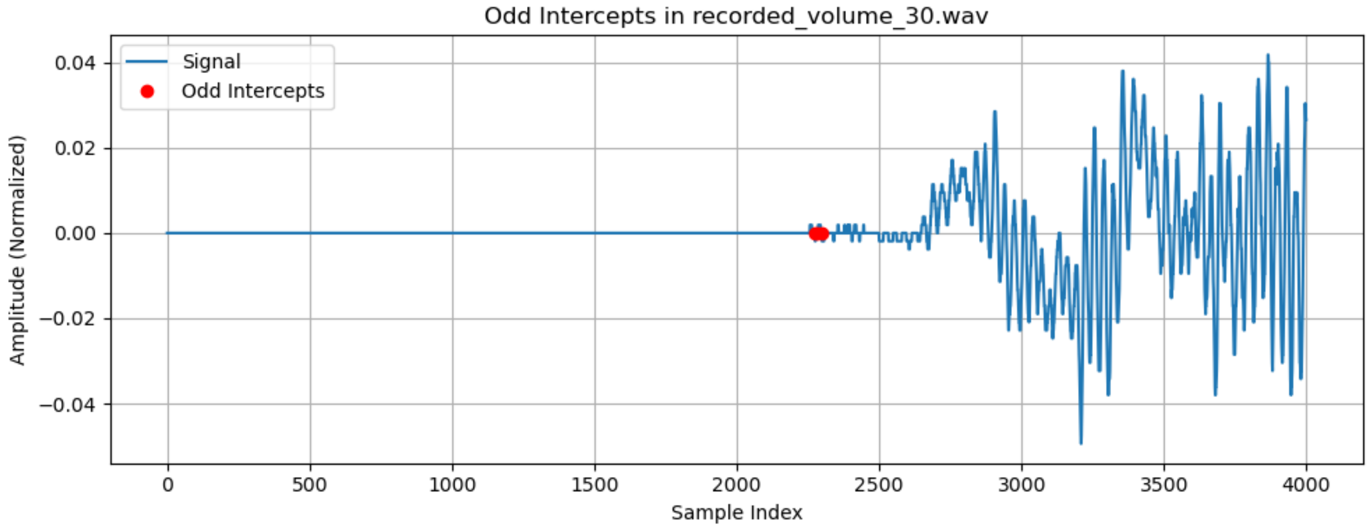
a) 10% volume

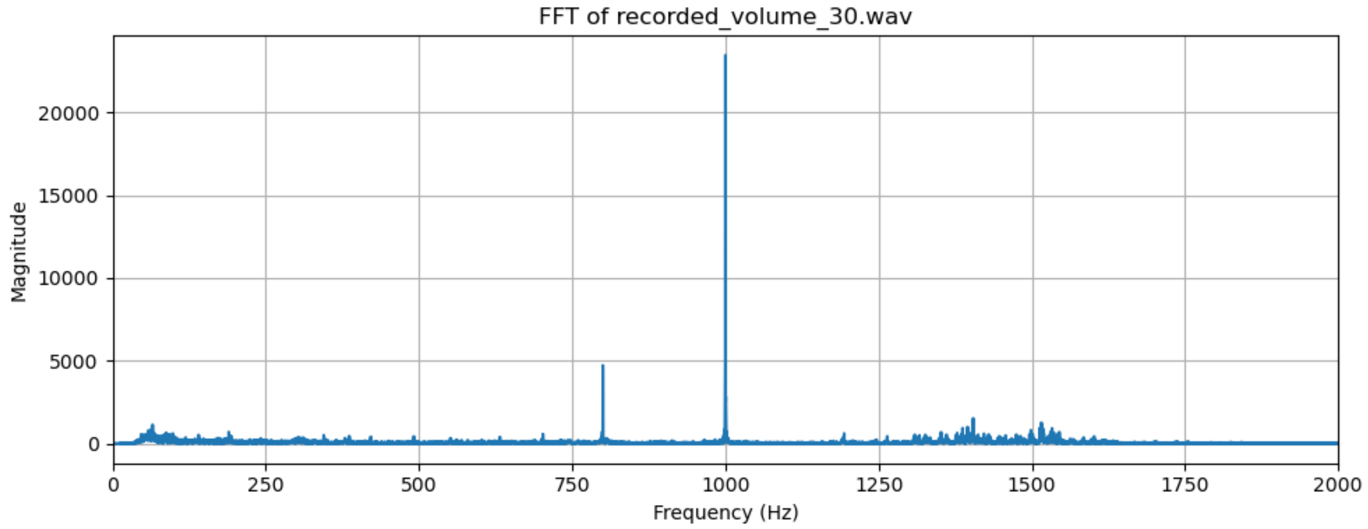




Intercept: The first odd zero-crossing occurs at sample index **2236**.
Top Frequency: Dominant tone at **1000.1 Hz**.
Magnitude: Relatively low at **3422.67**, as expected for a quiet signal.
Visual: The waveform is low in amplitude with noticeable noise in the FFT.

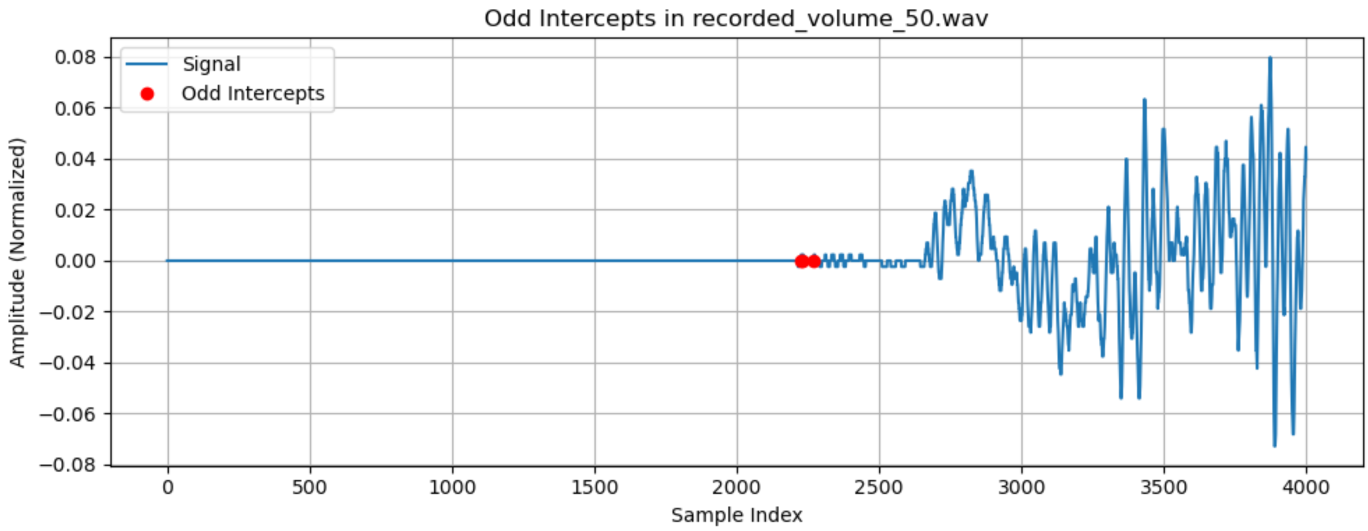
b) 30% volume

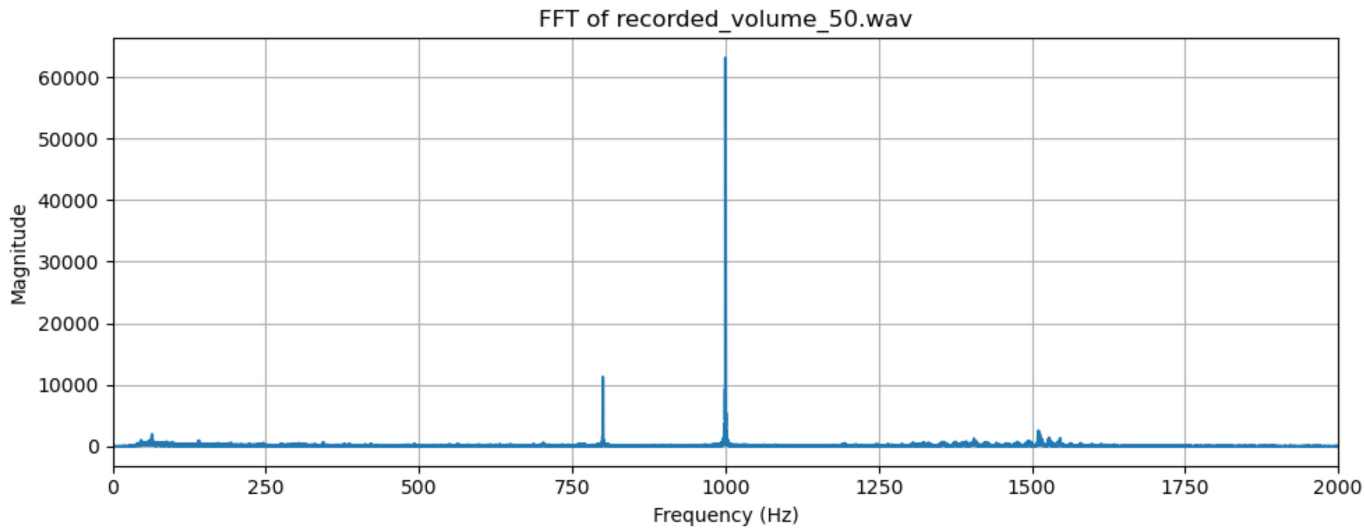




Intercept: The first odd zero-crossing occurs at sample index **2274**.
Top Frequency: Dominant tone at **1000.0 Hz**.
Magnitude: Moderate at **23435.50**, indicating a louder signal than 10%.
Visual: The waveform shows increased amplitude, and the FFT displays a clearer peak at the dominant frequency.

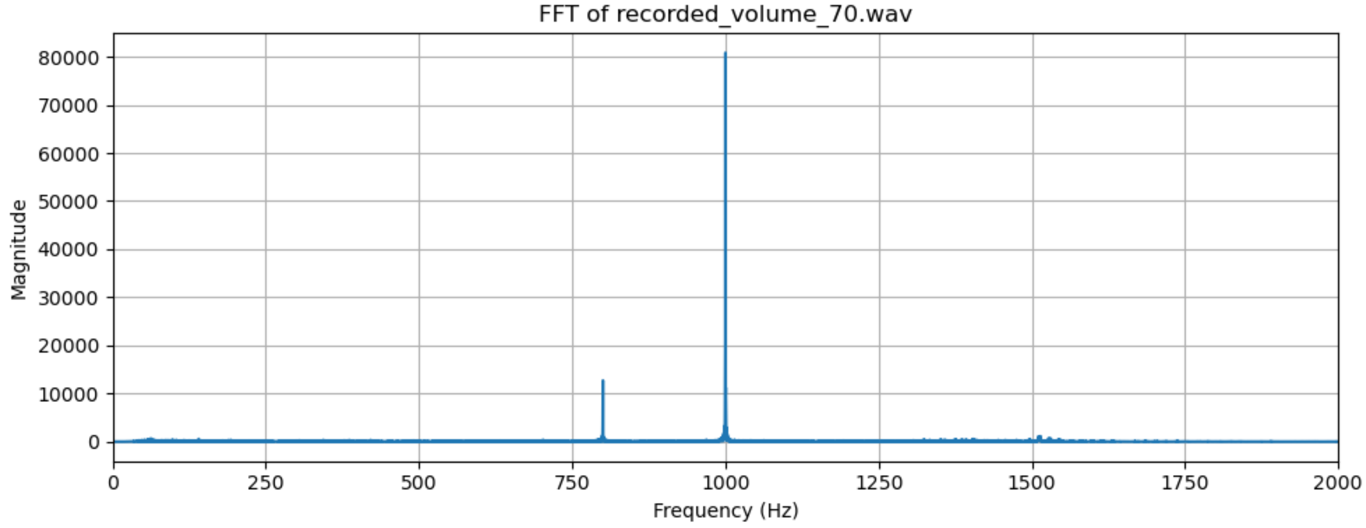
c) 50% volume

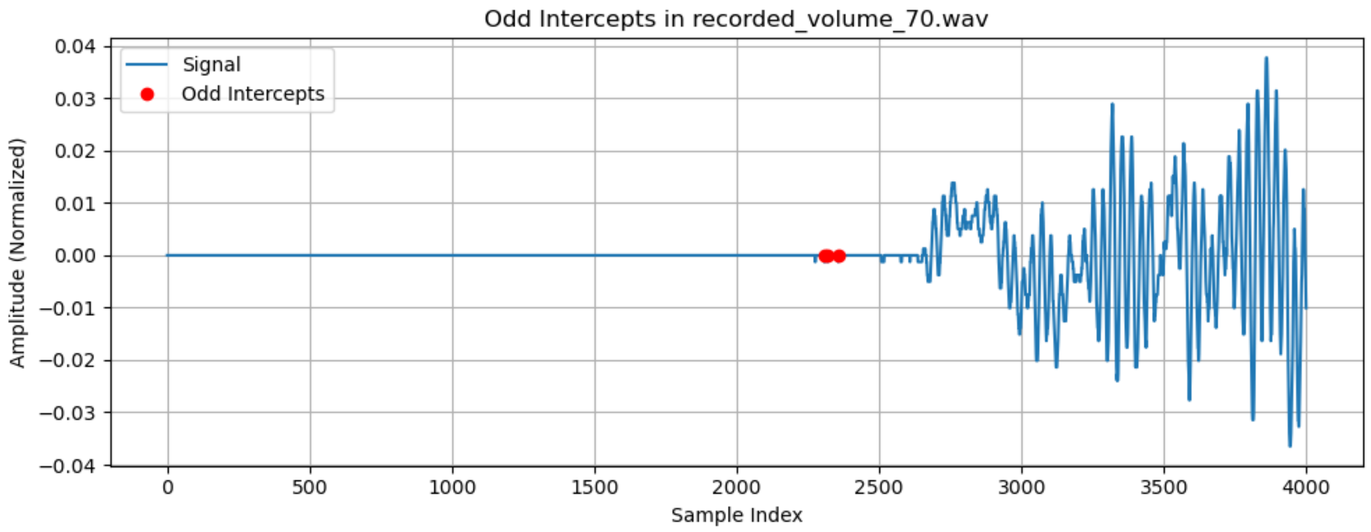




Intercept: The first odd zero-crossing occurs at sample index **2225**.
Top Frequency: Dominant tone at **999.9 Hz**.
Magnitude: Higher at **63151.09**, indicating a significantly louder signal.
Visual: The waveform has a higher amplitude, and the FFT shows a prominent peak at the dominant frequency.

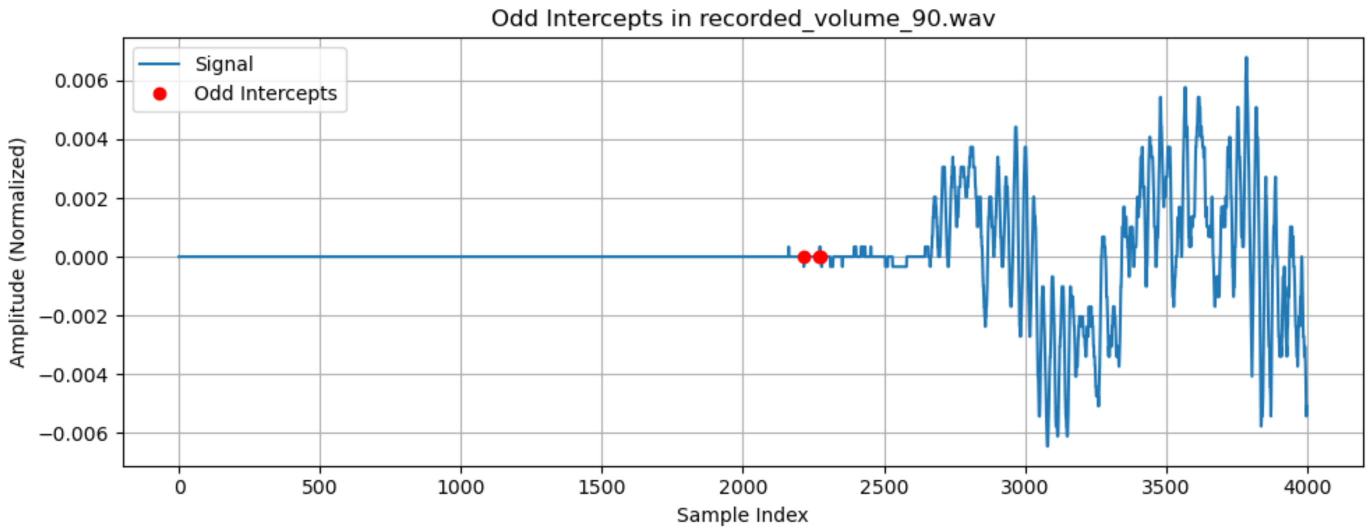
d) 70% volume

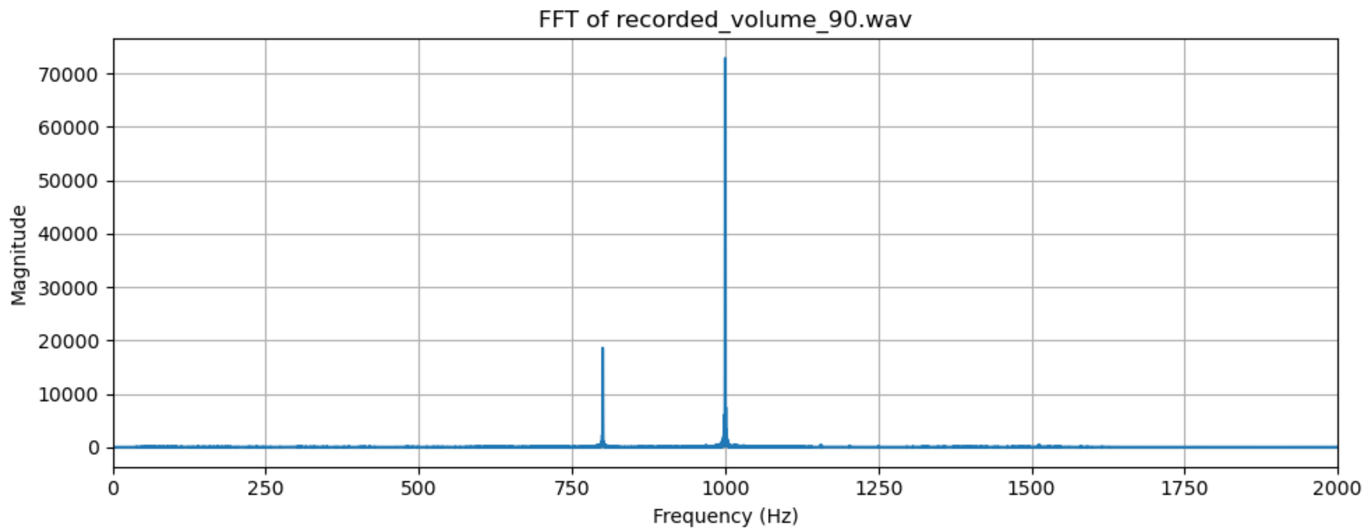




Intercept: The first odd zero-crossing occurs at sample index **2306**.
Top Frequency: Dominant tone at **1000.0 Hz**.
Magnitude: High at **80815.75**, indicating a extremely loud signal.
Visual: The waveform has a very high amplitude, and the FFT shows a strong peak at the dominant frequency.

e) 90% volume





Intercept: The first odd zero-crossing occurs at sample index **2215**.
Top Frequency: Dominant tone at **1000.1 Hz**.
Magnitude: Very high at **72815.96**, indicating an very loud signal, but under magnitude of 70% volume.
Visual: The waveform has the highest amplitude, and the FFT shows a very strong peak at the dominant frequency.

8. Conclusion

This project demonstrated how signal characteristics change under different playback conditions. The extracted intercept points provide insight into amplitude response and signal integrity.

9. Code explanation – Addendum

Folder structure:
dsp_project
----- signal_generator.py
----- intercepts.py
----- analyze.py
----- 2tone.wav
----- recorded_volume_10.wav
----- recorded_volume_30.wav
----- recorded_volume_50.wav
----- recorded_volume_70.wav
----- recorded_volume_90.wav

signal_generator.py

Code	Explanations
import numpy as np from scipy.io.wavfile import write import sounddevice as sd	
# Parameters	
fs = 44100	sample frequency
duration = 5	*duration of the sound*
f1, f2 = 800, 1000	frequency of the two tones

# Generate time axis 	
t = np.linspace(0, duration, int(fs * duration), endpoint=False)	

# Generate signal	
signal = 0.5 * (np.sin(2 * np.pi * f1 * t) + np.sin(2 * np.pi * f2 * t))	each tone is created using the sine function sin(2πft)

# Save to file 	
write("2tone.wav", fs, (signal * 32767).astype(np.int16))
	converts the signal into 16 bit format for wav file compatibility, 32767 is the max value of int16

sd.play(signal, fs) sd.wait()	

intercepts.py	
Code	Explanations
import numpy as np	
from scipy.io.wavfile import read	
import matplotlib.pyplot as plt	
def extract_intercepts(filename):	
fs, data = read(filename)	read the wav file
data = data.astype(np.float32)	*convert the signal into float for more precise output and overflowing*
if data.ndim > 1:	checks if the audio is stereo
data = data[:, 0]	*extract only the 1st channel*
data = data / np.max(np.abs(data))	normalize the signal amplitude to range [-1;1]
intercepts = np.where(np.diff(np.sign(data)))[0]	find the indices where the signal crosses 0
odd_intercepts = intercepts[2::2]	*extract every odd numbered zero crossing*
plt.figure(figsize=(10, 4))	set the figure size
plt.plot(data[:4000], label="Signal")	*plot the first 4000 samples of the signal*
plt.plot(odd_intercepts[:3], data[odd_intercepts[:3]], 'ro', label="Odd Intercepts")	mark the first three intercepts with red circles
plt.title(f"Odd Intercepts in {filename}")	
plt.xlabel("Sample Index")	
plt.ylabel("Amplitude (Normalized)")	
plt.legend()	
plt.grid()	
plt.tight_layout()	
plt.show()	
return odd_intercepts	return the array of odd intercept indices
# Loop through all volume levels	
for vol in [10, 30, 50, 70, 90]:	
filename = f"recorded_volume_{vol}.wav"	
print(f"Intercepts for volume {vol}%:")	
intercepts = extract_intercepts(filename)	
print("First 3 odd intercept indices:", intercepts[:3])	

analyze.py	
Code	Explanations
import numpy as np from scipy.io.wavfile import read import matplotlib.pyplot as plt	
def analyze_fft(filename):	
fs, data = read(filename)	read the wav file
data = data.astype(np.float32)	*convert the signal into float for more precise output and overflowing*
if data.ndim > 1:	checks if the audio is stereo
data = data[:, 0]	*extract only the 1st channel*
data = data / np.max(np.abs(data))	normalize the signal amplitude to range [-1;1]
N = len(data)	determine the number of samples in the signal
fft_result = np.fft.fft(data)	*compute the Fast Fourier Transform of the signal*
fft_magnitude = np.abs(fft_result)[:N // 2]	compute the magnitude and keep only the positive frequency
freqs = np.fft.fftfreq(N, 1 / fs)[:N // 2]	generate the corresponding frequency values for fft results
plt.figure(figsize=(10, 4))	set the figure size
plt.plot(freqs, fft_magnitude)	*plot fft frequency vs magnitude*

plt.title(f"FFT of {filename}")
   plt.xlabel("Frequency (Hz)")
   plt.ylabel("Magnitude")
   plt.grid()	

plt.xlim(0, 2000) # Adjust for your range of interest	limit the frequency range for better visibility
plt.tight_layout() plt.show()	
top_indices = np.argsort(fft_magnitude)[-5:][::-1]	get the indices of the 5 strongest frequency components
top_index = np.argmax(fft_magnitude)	find the index of the highest frequency

print(f"Top Frequency in {filename}:")
   print(f" Frequency: {freqs[top_index]:.1f} Hz, Magnitude: {fft_magnitude[top_index]:.2f}")
   print()	

# Loop through all volume levels for vol in [10, 30, 50, 70, 90]: analyze_fft(f"recorded_volume_{vol}.wav")	

9. Conclusion

- Intercept Analysis: The first odd zero-crossing indices fluctuate slightly across different volume levels but remain within a relatively consistent range. This suggests that while loudness increases, the signal structure does not drastically change in terms of timing characteristics.

- Frequency Stability: The dominant frequency remains around 1000 Hz across all volume levels, indicating that the primary tone is stable and unaffected by amplitude scaling.

- **Magnitude Growth**: The magnitude of the dominant frequency increases significantly with volume, showing a direct correlation between amplitude and perceived loudness. The jump from 3422.67 (10%) to 72815.96 (90%) confirms that higher volume levels amplify the signal strength effectively.

10. References

- DSP Course Book & DSP Laboratory materials
- Python libraries: **numpy, scipy.io.wavfile, matplotlib.pyplot, sounddevice**