Deaconu Mircea-Dan 1242B

Two-Tone Signal Project

Digital Signal Processing

Contents

1. Introduction

Non-linearities in real playback/record chains create new frequency components that were not present in the original signal. A classic way to reveal them is to feed the system with a two-tone stimulus and look for the resulting inter-modulation products (IMD) around the original tones.

This project measures how artefact level changes at different volume levels.

2. Setup

The setup needed to record and analyze the changes mentioned in the previous chapter is as follows:

- Laptop: runs the Python script, plays the two-tone signal. Then it runs the Fast Fourier Transform (FFT) script to make time and frequency plots for each recording;
- Laptop speakers: turn the digital waveform into sound;
- Phone: records the playback in stereo WAV, 48 kHz (options selected within recording app).

Libraries used:

- NumPy: fast array math; used to build the tones, windows, and run FFTs;
- SoundDevice: streams audio to/from the laptop’s sound card for playback/record;
- SciPy: saving and loading WAV files;
- Matplotlib: draws the time and frequency plots and save them as PNGs;
- Pathlib: save the graphs in a separated folder for an organized structure;
- Time: pause between recordings;

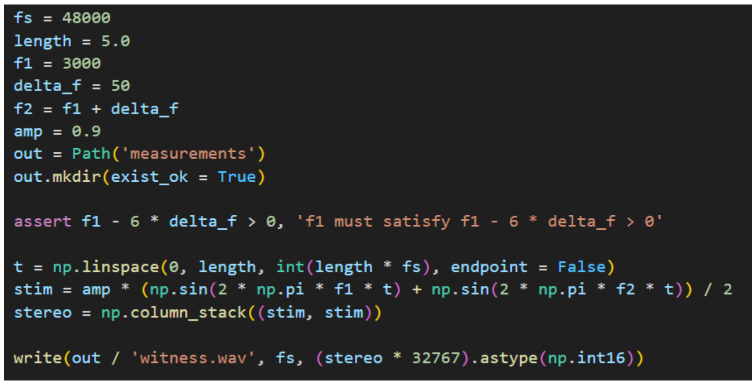
3. Code Description

3.1. Two-Tone Signal Generation

The following figure defines the test parameters and check's the rule given by the professor that

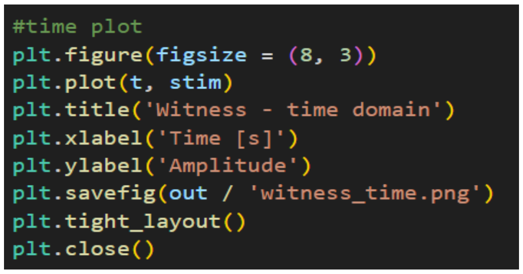
$$F_1 - 6\Delta f > 0$$

It then builds the time vector, creates the two-tone sine, copies it to left and right channels, and creates *witness.wav* file into the *measurements* folder



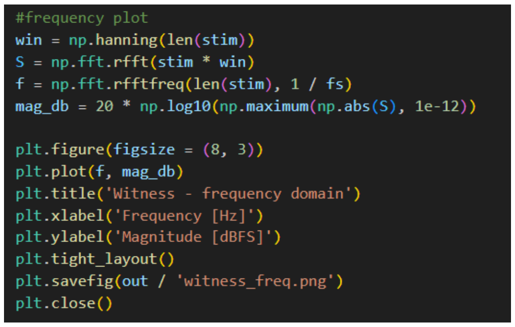
```
fs = 48000
length = 5.0
f1 = 3000
delta_f = 50
f2 = f1 + delta_f
amp = 0.9
out = Path('measurements')
out.mkdir(exist_ok = True)

assert f1 - 6 * delta_f > 0, 'f1 must satisfy f1 - 6 * delta_f > 0'

t = np.linspace(0, length, int(length * fs), endpoint = False)
stim = amp * (np.sin(2 * np.pi * f1 * t) + np.sin(2 * np.pi * f2 * t)) / 2
stereo = np.column_stack((stim, stim))

write(out / 'witness.wav', fs, (stereo * 32767).astype(np.int16))
```

Next, it creates an amplitude vs. time plot for the two-tone, labels axes and title, and saves it as a PNG file in the *measurements* folder.



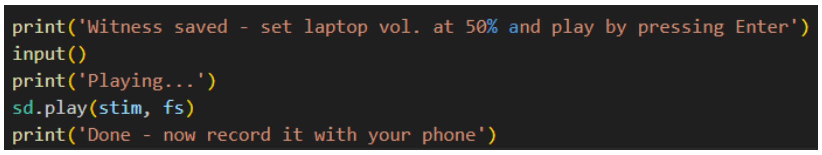
```
#time plot
plt.figure(figsize = (8, 3))
plt.plot(t, stim)
plt.title('Witness - time domain')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.savefig(out / 'witness_time.png')
plt.tight_layout()
plt.close()
```

For the frequency plot, it applies a Hann window tot the signal, performs an FFT, converts it to decibels relative to full scale (dBFS), and provides the magnitude vs. frequency plot, with the appropriate labels.



```
#frequency plot
win = np.hanning(len(stim))
S = np.fft.rfft(stim * win)
f = np.fft.rfftfreq(len(stim), 1 / fs)
mag_db = 20 * np.log10(np.maximum(np.abs(S), 1e-12))

plt.figure(figsize = (8, 3))
plt.plot(f, mag_db)
plt.title('Witness - frequency domain')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Magnitude [dBFS]')
plt.tight_layout()
plt.savefig(out / 'witness_freq.png')
plt.close()
```

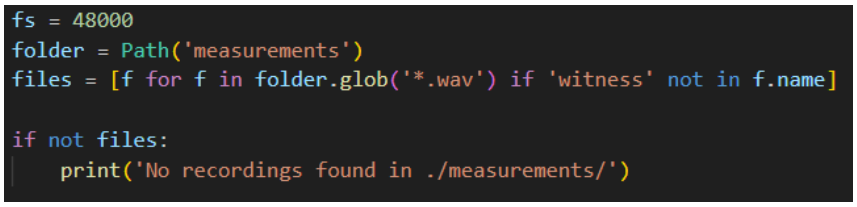
At the end of this file, the code gives simple instructions for the user to press the Enter key, then plays the signal through the laptop's speakers at the current volume using *sounddevice.play*. After the signal is done playing, the phone recording step begins.



```
print('Witness saved - set laptop vol. at 50% and play by pressing Enter')
input()
print('Playing...')
sd.play(stim, fs)
print('Done - now record it with your phone')
```

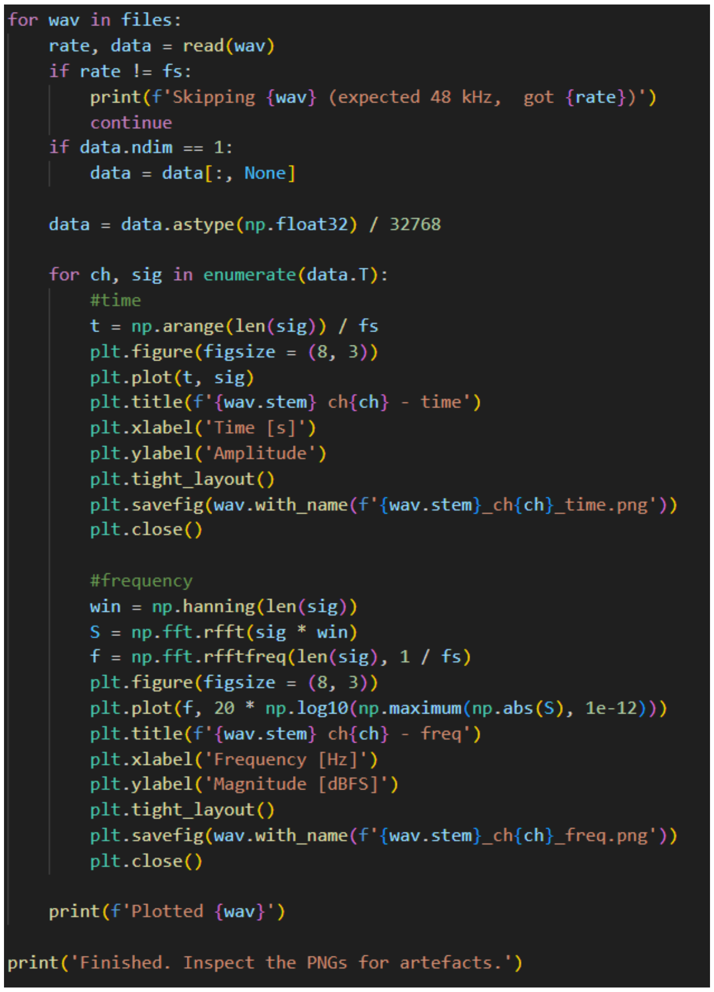
3.2. Analyzing the Recordings

In this Python script, we first fix the analysis target by gathering every *.wav* file, except for *witness.wav*, from the *measurements* folder. In case the list is empty, the user is warned promptly.



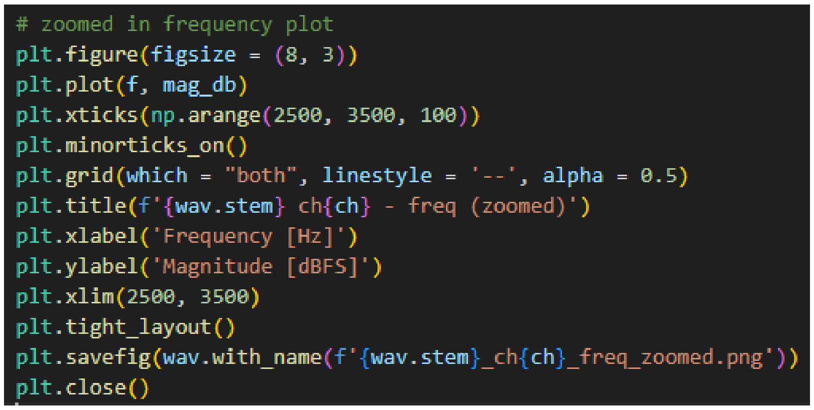
```
fs = 48000
folder = Path('measurements')
files = [f for f in folder.glob('*.wav') if 'witness' not in f.name]

if not files:
    print('No recordings found in ./measurements/')
```

With the previous check done, we loop through every recorded WAV (48 kHz only), reshapes mono to fake-stereo, converts counts to float without normalizing, then save a time plot and a Hann window FFT plot in dBFS, for each channel.



```python
for wav in files:
    rate, data = read(wav)
    if rate != fs:
        print(f'Skipping {wav} (expected 48 kHz,  got {rate})')
        continue
    if data.ndim == 1:
        data = data[:, None]

    data = data.astype(np.float32) / 32768

    for ch, sig in enumerate(data.T):
        #time
        t = np.arange(len(sig)) / fs
        plt.figure(figsize = (8, 3))
        plt.plot(t, sig)
        plt.title(f'{wav.stem} ch{ch} - time')
        plt.xlabel('Time [s]')
        plt.ylabel('Amplitude')
        plt.tight_layout()
        plt.savefig(wav.with_name(f'{wav.stem}_ch{ch}_time.png'))
        plt.close()

        #frequency
        win = np.hanning(len(sig))
        S = np.fft.rfft(sig * win)
        f = np.fft.rfftfreq(len(sig), 1 / fs)
        plt.figure(figsize = (8, 3))
        plt.plot(f, 20 * np.log10(np.maximum(np.abs(S), 1e-12)))
        plt.title(f'{wav.stem} ch{ch} - freq')
        plt.xlabel('Frequency [Hz]')
        plt.ylabel('Magnitude [dBFS]')
        plt.tight_layout()
        plt.savefig(wav.with_name(f'{wav.stem}_ch{ch}_freq.png'))
        plt.close()

    print(f'Plotted {wav}')

print('Finished. Inspect the PNGs for artefacts.')
```

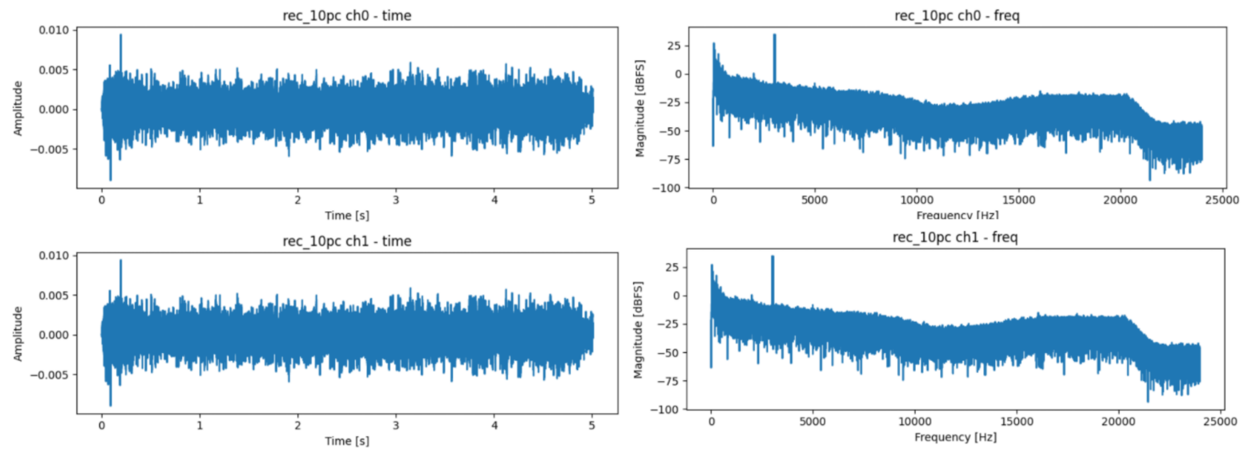
I also added new plots in the previous *for* loop so that the results can be seen in a better way in the 2500 – 3500 range. Here's the added snippet:

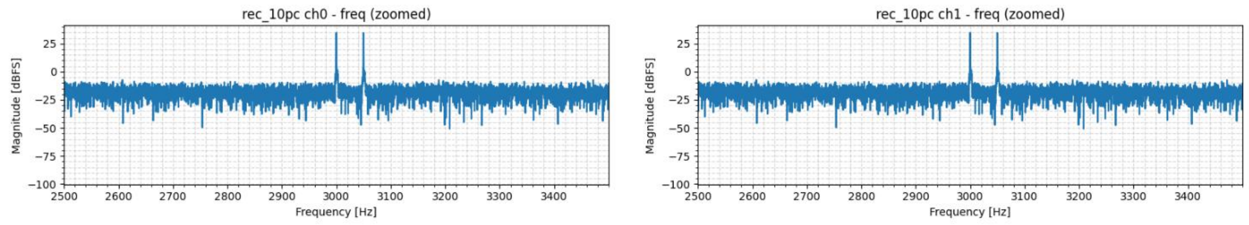


```python
# zoomed in frequency plot
plt.figure(figsize = (8, 3))
plt.plot(f, mag_db)
plt.xticks(np.arange(2500, 3500, 100))
plt.minorticks_on()
plt.grid(which = "both", linestyle = '--', alpha = 0.5)
plt.title(f'{wav.stem} ch{ch} - freq (zoomed)')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Magnitude [dBFS]')
plt.xlim(2500, 3500)
plt.tight_layout()
plt.savefig(wav.with_name(f'{wav.stem}_ch{ch}_freq_zoomed.png'))
plt.close()
```

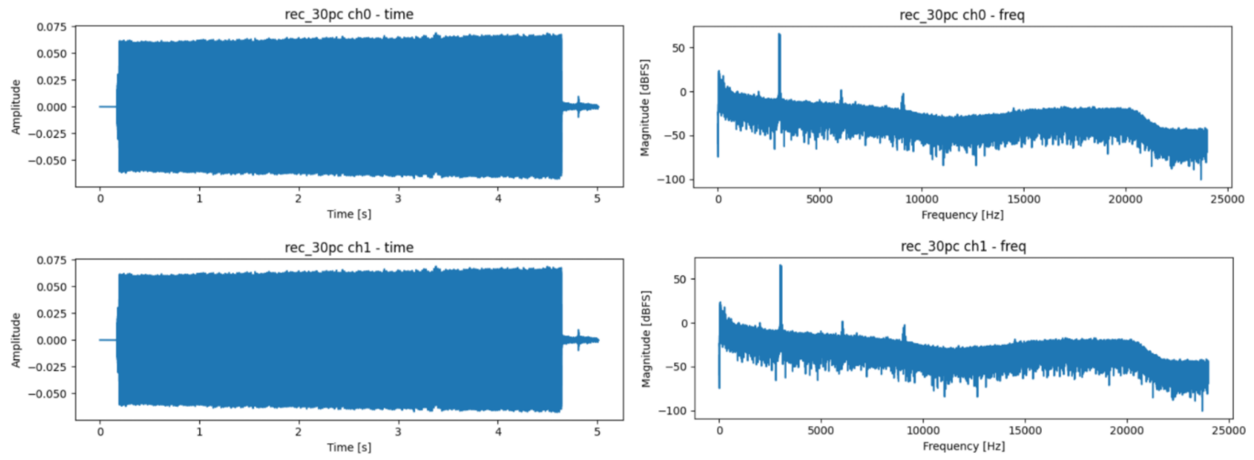
4. Results

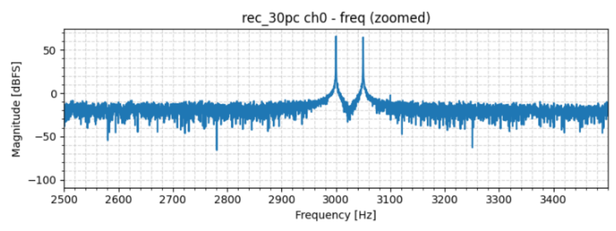
Volume at 10%

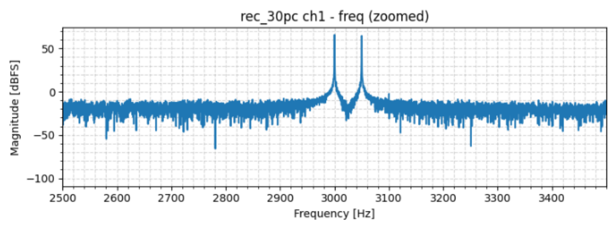




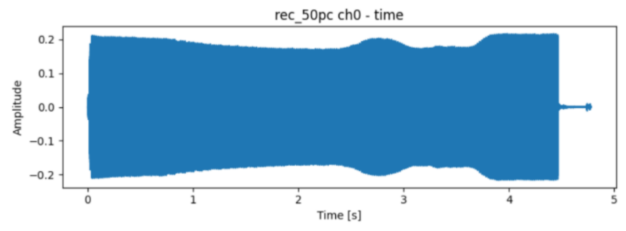
Volume at 30%

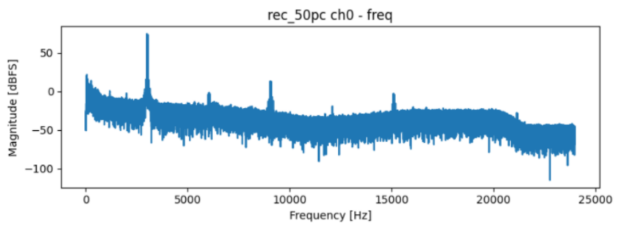


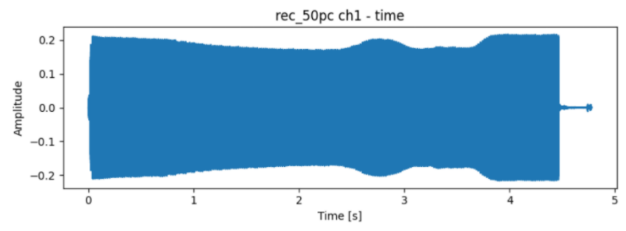


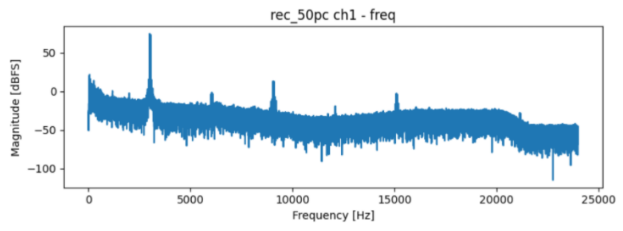


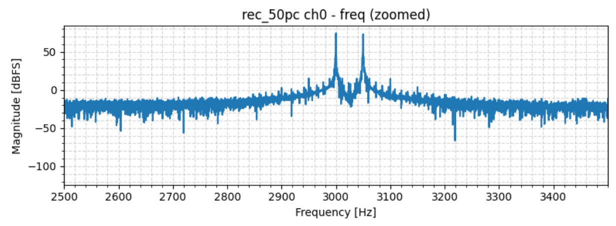
Volume at 50%

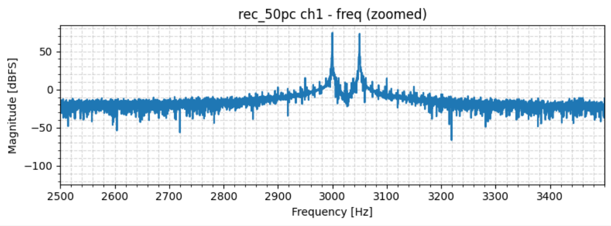




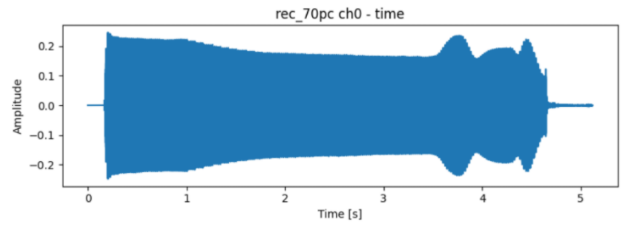


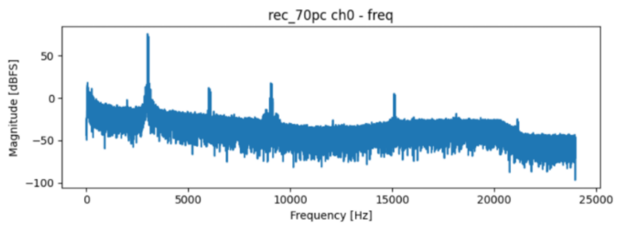


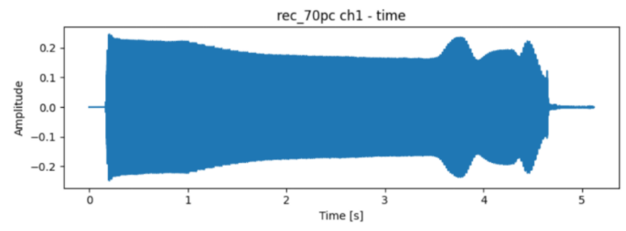


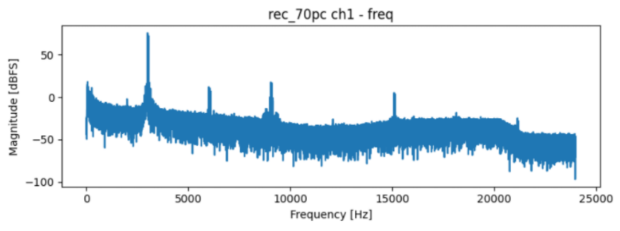


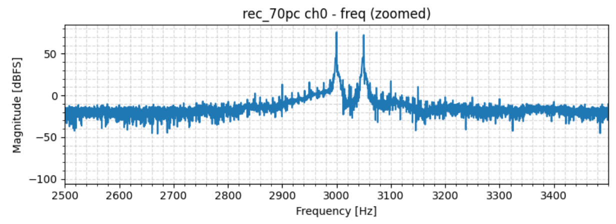
Volume at 70%

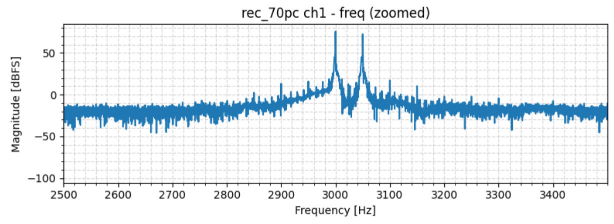




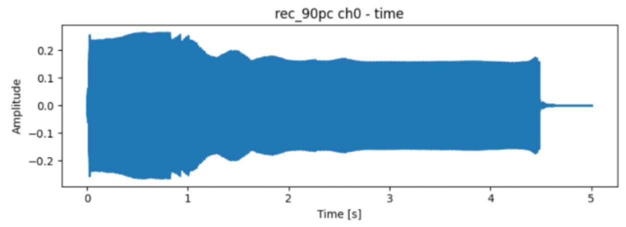


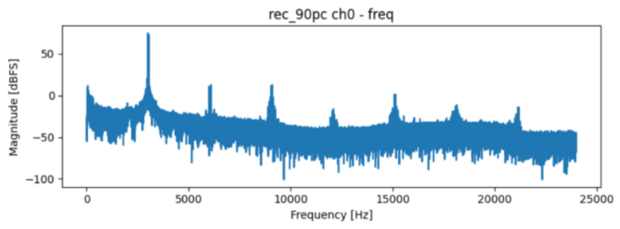


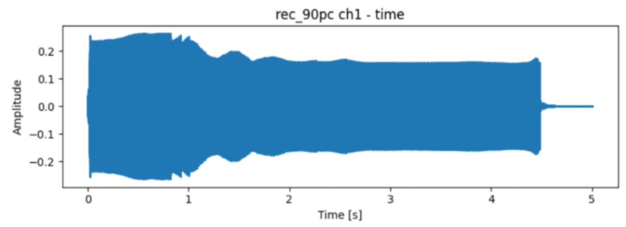


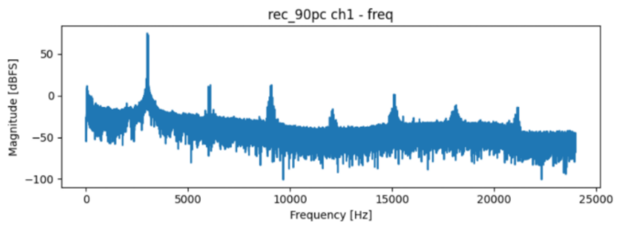


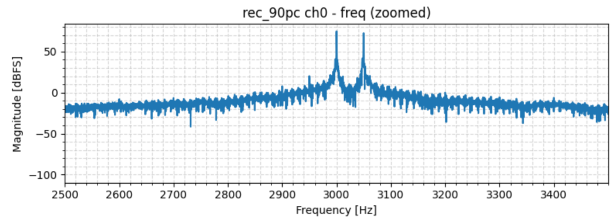
Volume at 90%

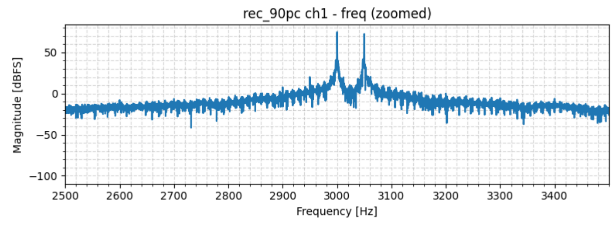












Observations:

At 10%, the two main spikes at 3000 Hz and 3050 Hz sit at about +25 dB, and there are no other clear spikes.

At 30%, fundamentals jump up to about 60 dB, with only tiny bumps at 2950 and 3100 Hz about 30 dB lower (so, around +30dB).

At 50%, the main spikes jump up by 5 dB and the side bumps by 10 dB, in comparison to the previous volume range.

At 70%, the sides are at approximately +45 dB, and some peaks can be seen at 6 kHz / 9 kHz near +35 dB.

At 90%, the two main spikes still top out at +65 dB but a dense “forest” of distortion spikes fills in from +20 to +30 dB across the whole plot.

5. Conclusion

To sum everything up, the system stays clean up to about 50% laptop volume, where a gap of approximately 30 dB can be obtained between the two-tone peaks and any distortion. Above that, distortion side-bands and harmonics rise rapidly with 70% showing obvious pumping and non-linearity, and 90% clipping. IMD measurements can be taken in the 30-50% volume range for the best accuracy.