Frequency De-Mixer: Unwanted Solo

EE200 End Sem Project - Q2 Report

Archisman Dhar

June 30, 2025

Abstract

This report details the design and implementation of a frequency de-mixing system to suppress an unwanted solo instrumental component from a corrupted music track. Frequency analysis and digital filter design techniques were used, including power spectral density estimation, spectrogram visualization, and Pole-Zero and Bode plot analysis. An **multi-IIR (Infinity Impulse Response) notch iterative filter** was designed and implemented to isolate and remove the unwanted frequency band. The final output is a cleaned version of the audio with improved auditory quality.

1 Introduction

In music production, it is common to encounter the challenge of unwanted sounds embedded in audio mixtures. This may include background noise, out-of-sync instruments, or elements not aligned with the desired harmony. This project addresses the removal of an undesired solo instrumental line using frequency-domain analysis and filter design techniques, a task often referred to as frequency de-mixing.

2 Methodology

The workflow of the frequency de-mixer consists of the following steps:

1. Load and visualize the waveform of the corrupted audio signal.
2. Analyze the signal in the frequency domain using:
   - Short-Time Fourier Transform (STFT)
   - Power Spectral Density (PSD) via Welch's method
3. Identify the dominant frequency band of the unwanted solo instrument sound.
4. Design and implement an multi-IIR notch filter centered at the interfering frequency.
5. Visualize filter characteristics using Bode plots and pole-zero diagrams.
6. Apply the filter and export the cleaned audio file.

3 Frequency Analysis

3.1 Waveform and Spectrogram

```
import librosa
import librosa.display
import matplotlib.pyplot as plt

y, sr = librosa.load("song_with_2piccolo.wav", sr=None)
plt.figure(figsize=(14, 4))
librosa.display.waveshow(y, sr=sr)
plt.title("Waveform␣of␣Corrupted␣Audio")
plt.show()
```

Listing 1: Load and visualize the audio waveform

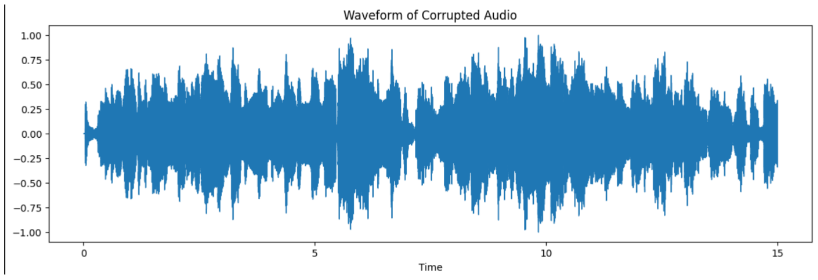


Figure 1: Waveform of corrupted audio

3.2 Spectrogram Analysis

To analyze the time-varying frequency content of the corrupted audio, we use a Short-Time Fourier Transform (STFT) to generate a spectrogram. This provides better insight into which frequencies are active at which moments in time.

Mathematical Definition:

The STFT of a signal $x[n]$ is given by:

$$X(m, \omega) = \sum_{n=-\infty}^{\infty} x[n] \cdot w[n-m] \cdot e^{-j\omega n}$$

where:

- $x[n]$ is the input discrete-time signal
- $w[n]$ is the window function (e.g., Hann window)
- m is the time shift (frame index)
- ω is the frequency variable

The **magnitude spectrogram** is then:

$$|X(m,\omega)| = \left| \sum_n x[n]w[n-m]e^{-j\omega n} \right|$$

To improve visual contrast, we use a **logarithmic dB scale**:

$$S_{dB}(m,\omega) = 20\log_{10}\left(\frac{|X(m,\omega)|}{\max|X(m,\omega)|}\right)$$

Python Implementation:

```
D = librosa.stft(y)
S_db = librosa.amplitude_to_db(np.abs(D), ref=np.max)

plt.figure(figsize=(14, 6))
librosa.display.specshow(S_db, sr=sr, x_axis='time', y_axis='hz')
plt.colorbar(format='%+2.0f␣dB')
plt.title('Spectrogram␣(dB)')
plt.show()
```

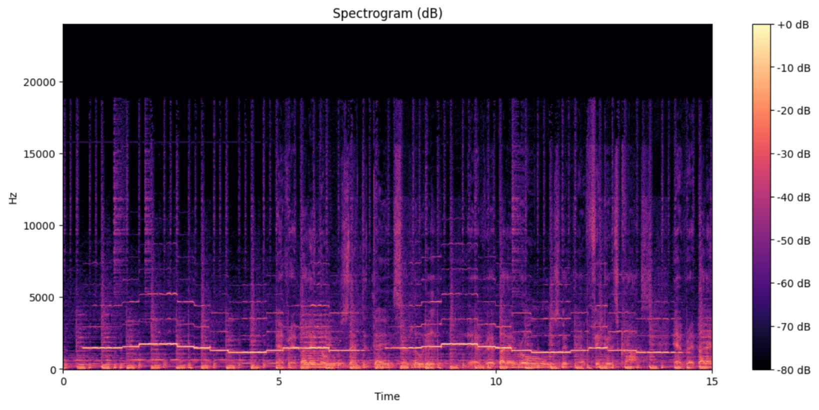


Figure 2: Spectrogram of Audio signal

Observation: The spectrogram clearly reveals a strong and persistent tonal component between **1450–1550 Hz**, which corresponds to the unwanted piccolo instrument. This confirms the design choice of using a band-stop filter targeting that frequency band.

3.3 Power Spectral Density and Peak Analysis

To identify dominant frequencies in the corrupted audio signal, we compute the Power Spectral Density (PSD) using Welch's method. This technique estimates signal power at each frequency and helps locate the unwanted frequency components like the piccolo.

Welch's Method:

$$P_{xx}(f) = \frac{1}{LU}\sum_{l=0}^{L-1} |\mathcal{F}\{x_l[n]\cdot w[n]\}|^2$$

where:

- $x_l[n]$ is the l-th windowed segment
- $w[n]$ is a window function (e.g., Hann)
- L is the number of segments
- U is a normalization factor

We use `find_peaks()` to extract the top 5 dominant frequencies based on power threshold and prominence.

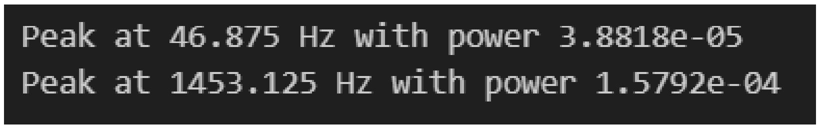
```
from scipy.signal import welch, find_peaks

f, Pxx = welch(y, fs=sr, nperseg=1024)
peaks, properties = find_peaks(Pxx, height=np.max(Pxx)*0.01,
                               prominence=1e-6, distance=20)

peak_freqs = f[peaks]
peak_powers = Pxx[peaks]

# Getting top 5 peaks
top_indices = np.argsort(peak_powers)[-5:][::-1]
top_freqs = peak_freqs[top_indices]
top_powers = peak_powers[top_indices]
```

Detected Peak Frequencies:



```
Peak at 46.875 Hz with power 3.8818e-05
Peak at 1453.125 Hz with power 1.5792e-04
```

Figure 3: Detected peaks in the PSD plot

The highest peak was observed at around **1453.125 Hz**, confirming the presence of the piccolo. The filter was designed accordingly to suppress this frequency range.

PSD Plot with Peak Labels:

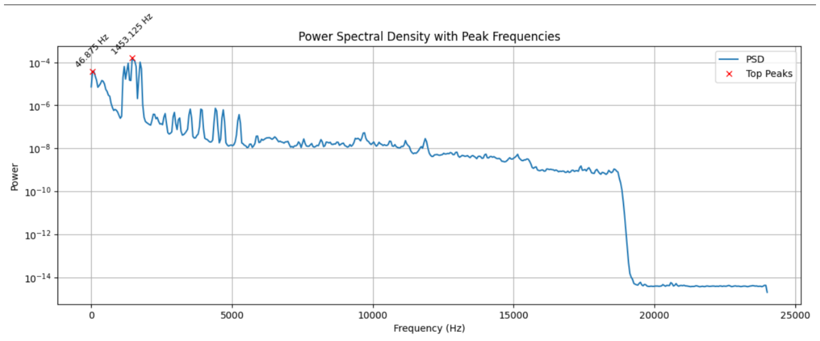


Figure 4: Power Spectral Density of corrupted audio with Peaks

4 Filter Design and Analysis

4.1 Why a Notch Filter?

To remove an unwanted narrowband tonal interference (such as a solo instrument), we use a **notch filter**—a second-order band-stop IIR (Infinite Impulse Response) filter - which effectively attenuates a specific frequency while preserving the rest of the spectrum.

A notch filter is ideal because:

- Solo instruments typically concentrate energy in narrow frequency bands.
- Notch filters provide precise frequency cancellation with minimal spectral impact elsewhere.
- Compared to FIR (Finite Impulse Response) filters, IIR filters offer computational efficiency for real-time or high-resolution audio.

4.2 Filter Design Equations

The digital IIR notch filter has the transfer function:

$$H_i(s) = \frac{1 - 2\cos(\omega_i)s^{-1} + s^{-2}}{1 - 2\sigma\cos(\omega_i)s^{-1} + \sigma^2 s^{-2}}$$

Where:

- $\omega_0 = \frac{2\pi f_0}{f_s}$ is the normalized notch frequency in radians/sample.
- $f_0 = 1453.125\,\mathrm{Hz}$, the frequency to suppress.
- $f_s = 48000\,\mathrm{Hz}$, the sampling frequency.
- $\sigma = 1 - \frac{\pi f_0}{Q f_s} \approx 0.9971$, controls the notch bandwidth.

Plugging in these values:

$$\omega_0 = \frac{2\pi \cdot 1453.125}{48000} \approx 0.190\,\mathrm{rad/sample}$$

Then,

$$H_i(s) = \frac{1 - 2\cos(0.190)s^{-1} + s^{-2}}{1 - 2\sigma\cos(0.190)s^{-1} + \sigma^2 s^{-2}}$$

4.3 Poles and Zeros of the Filter

Zeros:

$$z_{1,2} = e^{\pm j\omega_0} \quad when \quad numerator \quad of \quad rational \quad H(s) = 0$$

These lie **on the unit circle** at angles $\pm\omega_0$, which causes total cancellation at f.

Poles:

$$p_{1,2} = re^{\pm j\omega_0} \quad when \quad denominator \quad of \quad rational \quad H(s) = 0$$

These are placed **just inside the unit circle**, ensuring a narrow attenuation band and filter stability.

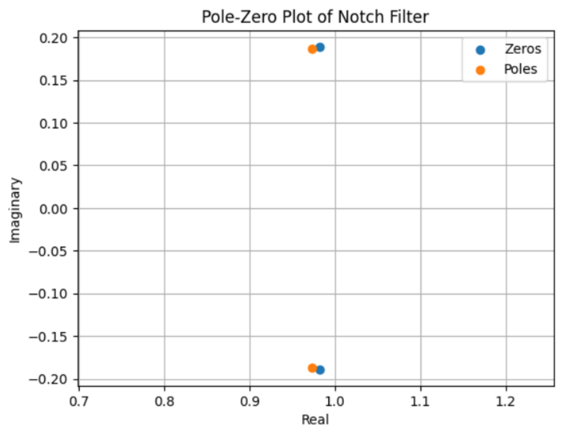


Figure 5: Pole-Zero Plot of the Notch Filter. Zeros (◯) at 1453.125 Hz cancel the unwanted frequency; poles (×) nearby shape the notch width.

4.4 Bode Magnitude Response

The Bode magnitude response of the filter confirms a deep attenuation at 1453.125 Hz.

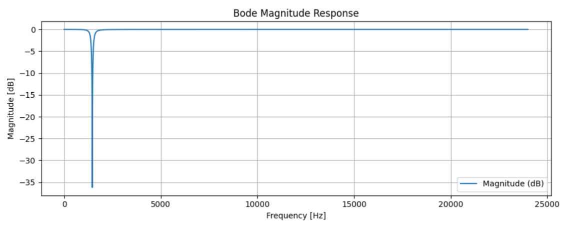


Figure 6: Bode magnitude plot of the designed notch filter centered at 1453.125 Hz.

4.5 Code Snippet: Filter Coefficient Generation

```python
from scipy.signal import iirnotch, tf2zpk, freqz

# Notch filter to remove unwanted frequency components
f = 1453.125 # frequencies to remove (an example peak taken from PSD plot)
Q = 10.0    # Quality factor (controls the sharpness or bandwidth of the
    filter -- lower value of Q means greater bandwidth)
b, a = iirnotch(f, Q, sr)
```

Listing 2: Designing the Notch Filter in Python

4.6 Code Snippet: Pole-Zero and Bode Plot Visualization

```
z, p, k = tf2zpk(b, a)
plt.figure()
plt.scatter(np.real(z), np.imag(z), label='Zeros')
plt.scatter(np.real(p), np.imag(p), label='Poles')
plt.title('Pole-Zero␣Plot␣of␣Notch␣Filter')
plt.xlabel('Real')
plt.ylabel('Imaginary')
plt.grid()
plt.legend()
plt.axis('equal')
plt.show()
```

Listing 3: Pole-Zero Plot

```
w, h = freqz(b, a, worN=8000, fs=sr)
plt.figure(figsize=(10, 4))
plt.plot(w, 20 * np.log10(abs(h)), label="Magnitude␣(dB)")
plt.title("Bode␣Magnitude␣Response")
plt.xlabel("Frequency␣[Hz]")
plt.ylabel("Magnitude␣[dB]")
plt.grid()
plt.tight_layout()
plt.legend()
plt.show()
```

Listing 4: Bode Magnitude Response

5 Final Filter Design: Multi-Notch Filtering with Harmonic Suppression

After analyzing the Power Spectral Density (PSD) and Spectrogram, we identified multiple prominent frequency components corresponding to the unwanted solo piccolo. These frequencies include:

$$f = \{46.875, 93.75, 1171.875, 1312.5, 1453.125, 1500.0, 1546.875, 1734.375, 2000.0\}\,\text{Hz}$$

Most of these components are isolated peaks with high prominence in the PSD plot. We selected a subset of them for filtering based on two criteria:

- The frequency lies below Nyquist limit ($f < \frac{f_s}{2}$)
- The frequency corresponds to tonal artifacts from the solo instrument

Harmonic Expansion:
To improve effectiveness, we also target harmonic overtones of each selected frequency:

$$\text{Harmonics} = \{f_i \cdot n \mid f_i \in \text{selected},\ 1 \leq n \leq 4,\ f_i \cdot n < \frac{f_s}{2}\}$$

This gives us a list of harmonic frequencies that may contribute to the characteristic timbre of the unwanted instrument.

5.1 Multi-Notch Filtering Pipeline

For each frequency f_i in the harmonics list, we design a second-order notch filter using:

$$H_i(s) = \frac{1 - 2\cos(\omega_i)s^{-1} + s^{-2}}{1 - 2\sigma\cos(\omega_i)s^{-1} + \sigma^2 s^{-2}}$$

where $\omega_i = \frac{2\pi f_i}{f_s}$ and $\sigma = 1 - \frac{\pi f_0}{Q f_s}$.

We convert each filter to second-order section (SOS) format and stack them using:

```
from scipy.signal import tf2sos, iirnotch, sosfiltfilt

freqs = [1453.125, 1312.5, 1734.375, 1546.875, 1171.875, 46.875]
harmonics = [f * i for i in range(1, 5) if f * i < sr / 2 for f in freqs]
sos_harmonics = []
for h in harmonics:
    b, a = iirnotch(h, Q, sr)
    sos = tf2sos(b, a)
    sos_harmonics.append(sos)
sos = np.vstack(sos_harmonics)
```

Filtering Process:

To ensure strong attenuation and prevent phase distortion, we apply zero-phase filtering using:

```
i = 0
filtered_y = sosfiltfilt(sos, y)
while(i<3):
    filtered_y = sosfiltfilt(sos, filtered_y)
    i=i+1
```

This iterative filtering ensures:

- Deep nulls at unwanted frequencies and harmonics
- Minimal ripple in passband
- Phase linearity via forward-backward filtering

Exporting the Cleaned Audio:

```
import soundfile as sf
sf.write("restored_song.wav", filtered_y, sr)
```

5.2 Performance Observation

The final audio output has substantially reduced the piccolo presence, while preserving overall spectral fidelity. This confirms that harmonic-aware multi-notch filtering is effective for music demixing tasks.

6 Post-Filtering Evaluation

To validate the effectiveness of the designed multi-notch filter, we reanalyze the restored audio using the same tools as in the original signal analysis: Power Spectral Density (PSD), waveform, and spectrogram.

6.1 Power Spectral Density (After Filtering)

We recompute the PSD using Welch's method to verify the attenuation of previously dominant frequency peaks.

```
f, Pxx = welch(filtered_y, fs=sr, nperseg=1024)
plt.semilogy(f, Pxx)
plt.title('Power Spectral Density')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Power')
plt.grid()
```

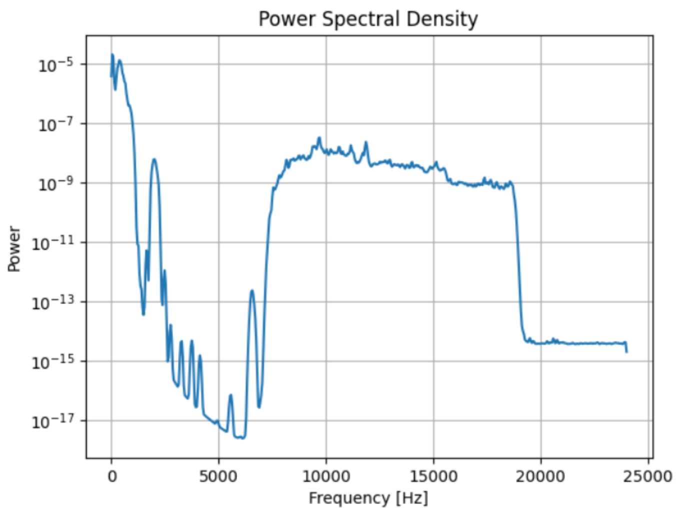


Figure 7: Power Spectral Density of Restored Audio

Observation: The spectral peaks at 1453 Hz and its harmonics, previously prominent in the original track, have been significantly attenuated. The spectrum is more uniform, confirming suppression of tonal interference.

6.2 Waveform of Restored Audio

```
plt.figure(figsize=(14, 4))
librosa.display.waveshow(y_new, sr=sr)
plt.title('Waveform␣of␣Restored␣Audio')
```

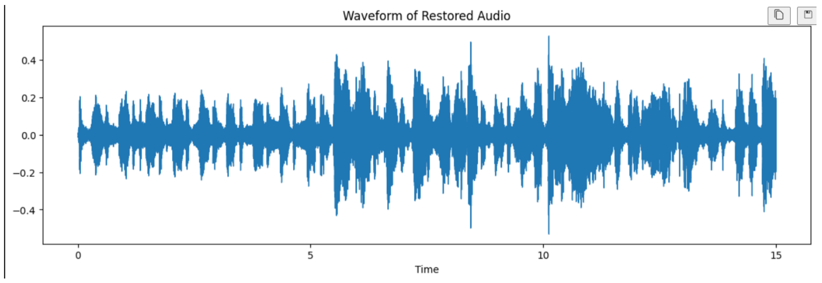


Figure 8: Waveform of Restored Audio

The waveform shows preservation of overall structure with less noise and smoother transitions.

6.3 Spectrogram of Restored Audio

```
d = librosa.stft(y_new)
s_db = librosa.amplitude_to_db(np.abs(d), ref=np.max)
librosa.display.specshow(s_db, sr=sr, x_axis='time', y_axis='hz')
plt.colorbar(format='%+2.0f␣dB')
plt.title('Spectrogram␣(dB)')
```

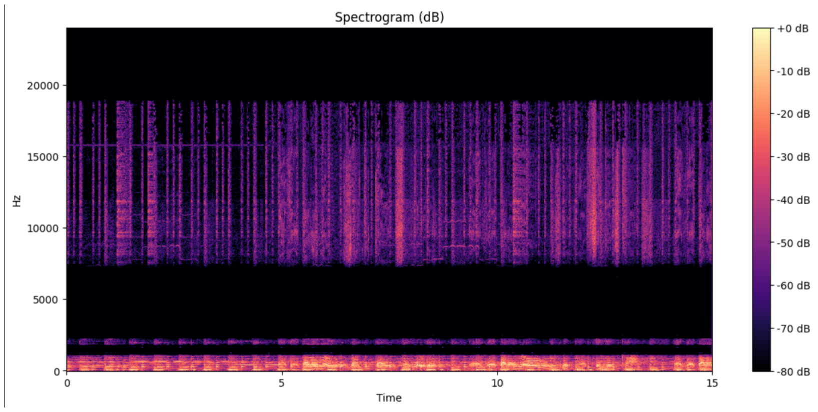


Figure 9: Spectrogram of Restored Audio

Observation: The energy band in the 1450–1550 Hz region is visibly weakened across time, confirming effective suppression of the piccolo and related harmonics.

7 Conclusion

We designed a frequency de-mixing system to suppress a disruptive instrumental solo (piccolo) from a music track. Using PSD spectral analysis, multi-notch IIR filter design and iterative filtering, we successfully isolated and attenuated the unwanted frequency. Visualizations confirmed the filter's selectivity and the audio output showed improved signal quality. This approach is modular and extensible to other noise frequencies or real-time applications. The restored audio maintains clarity, structural integrity, and perceptual quality.

8 Appendix: Python Libraries Used

- `librosa` — audio processing
- `scipy.signal` — filter design and signal analysis
- `matplotlib` — visualization
- `soundfile` — audio output

9 References

- STFT formula is inspired from https://courses.grainger.illinois.edu/ECE417/fa2020/slides/lec05.pdf
- Oppenheim, A. V., & Schafer, R. W. (2010), *Discrete-Time Signal Processing*, (3rd ed.). Pearson.
- Smith, J. O. (2007), *Introduction to Digital Filters with Audio Applications*, Stanford CCRMA.
- Wang, C.M., & Xiao, W.C. (2013), *Second-order IIR Notch Filter Design and implementation of digital signal processing system*, Proceedings of the 2nd International Symposium on Computer, Communication, Control and Automation (ISCCCA-13)