ENAE 380: Lab 04

Due on October 27, 2024 at 11:59 PM

Dr. Mumu Xu, 0106

Vai Srivastava

October 27, 2024

Problem 6.1: Inserting Noise

Solution

Part 1

```
    def noisy():
        f_main = 1000  # main frequency in Hz
        f_noise = 50   # noise frequency in Hz

        sample_rate = 48000  # samples per second
        duration = 10        # duration in seconds
        t = np.linspace(0, duration, duration * sample_rate, endpoint=False)
 # time vector

        w_main = np.sin(2 * np.pi * f_main * t)  # main sine wave
        w_noise = np.sin(2 * np.pi * f_noise * t)  # noise sine wave
        w_combo = w_main + w_noise  # combined wave
        ...

```

Problem 6.1 continued on next page. . .

Part 2

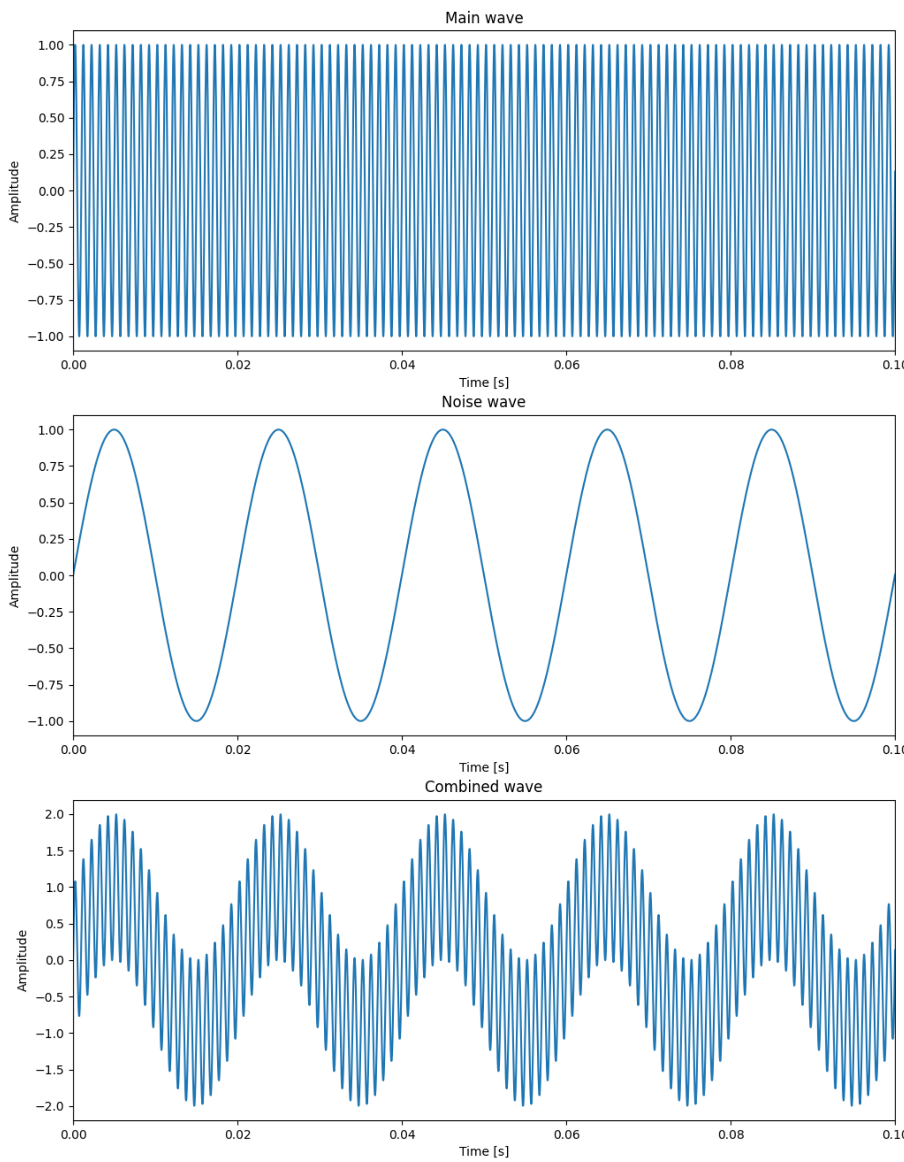

Figure 1: Plots of Original, Noise, and Combined Waves

Problem 6.1 continued on next page. . .

Part 3

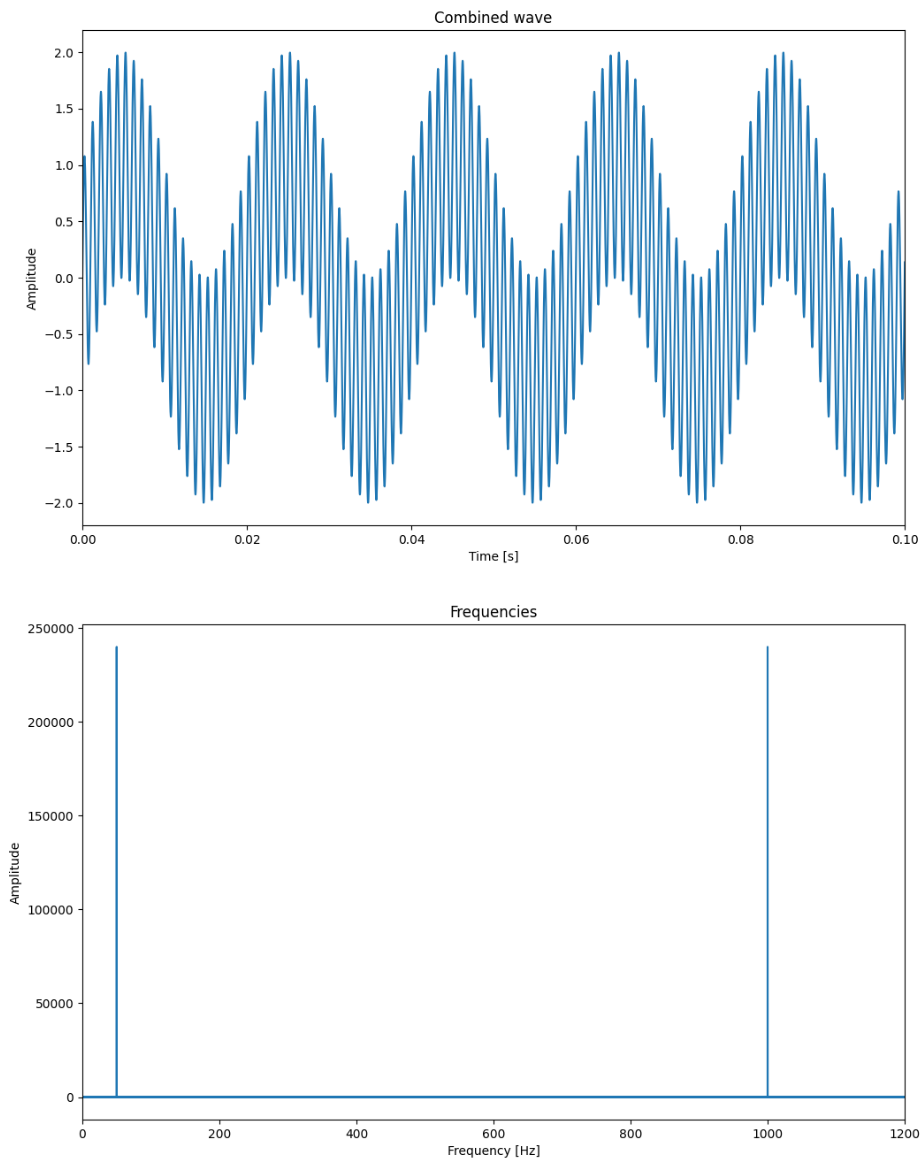

Figure 2: Frequencies of Combined Wave

Problem 6.1 continued on next page...

Part 4

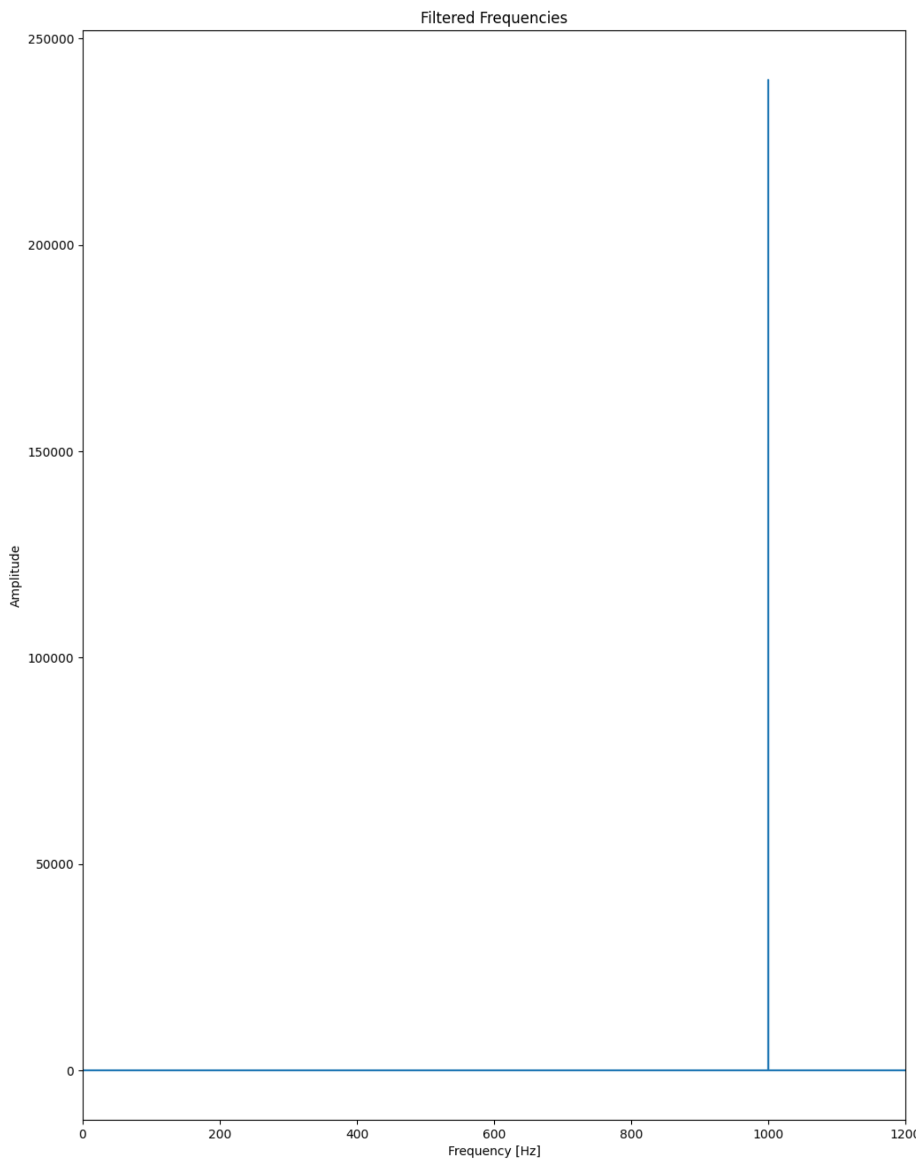

Figure 3: Frequencies of Filtered Wave

Problem 6.1 continued on next page...

Part 5

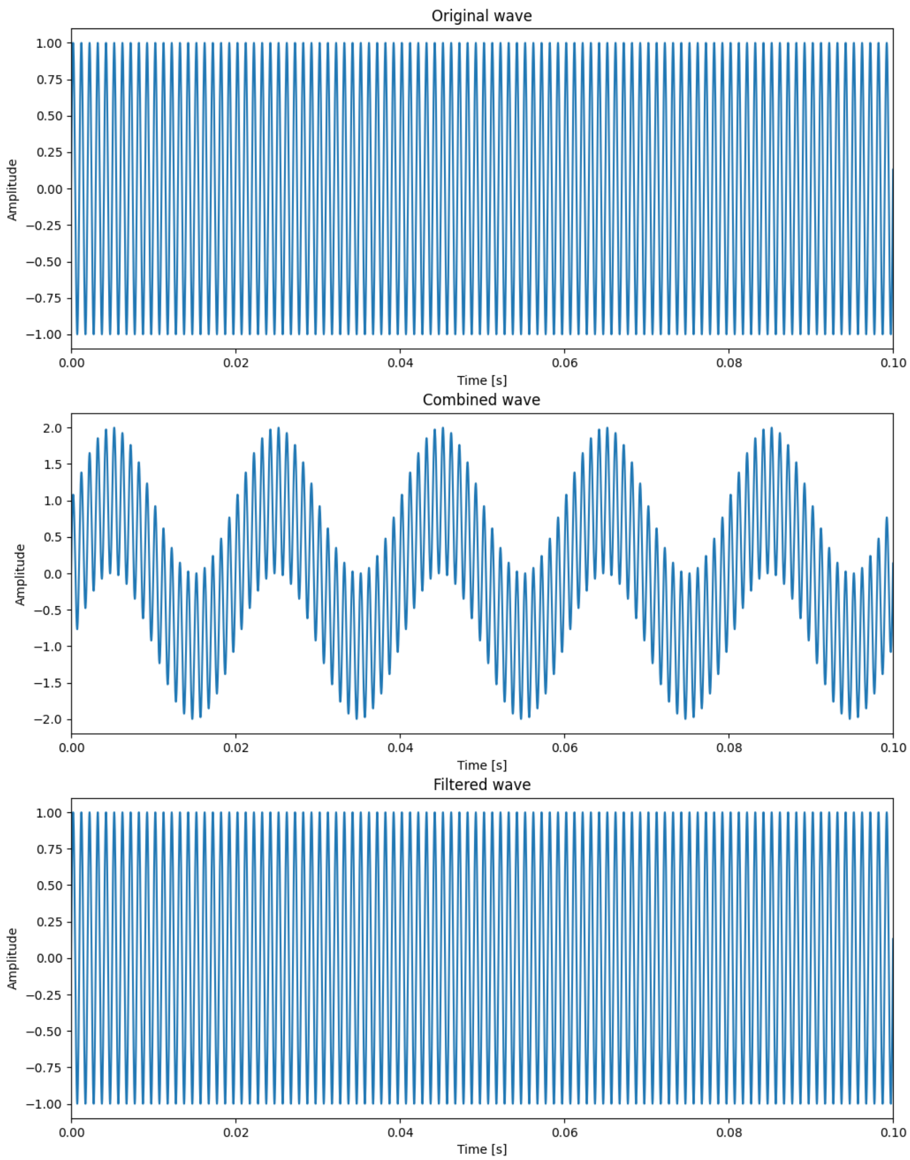

Figure 4: Plots of Original, Combined, and Filtered Waves

Problem 6.2: Pitch

Solution

Part 1

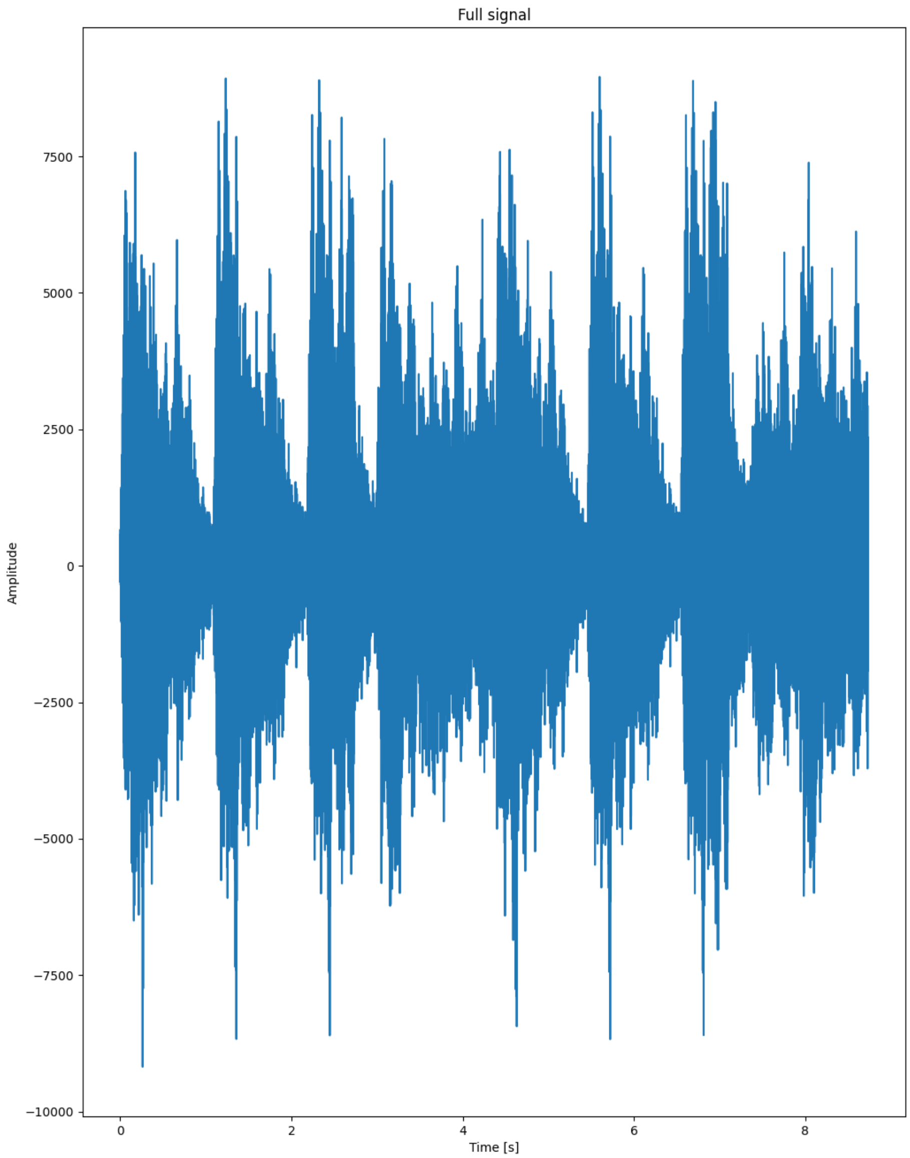

Figure 5: Plot of `trumpet.wav` Signal

Problem 6.2 continued on next page. . .

Part 2

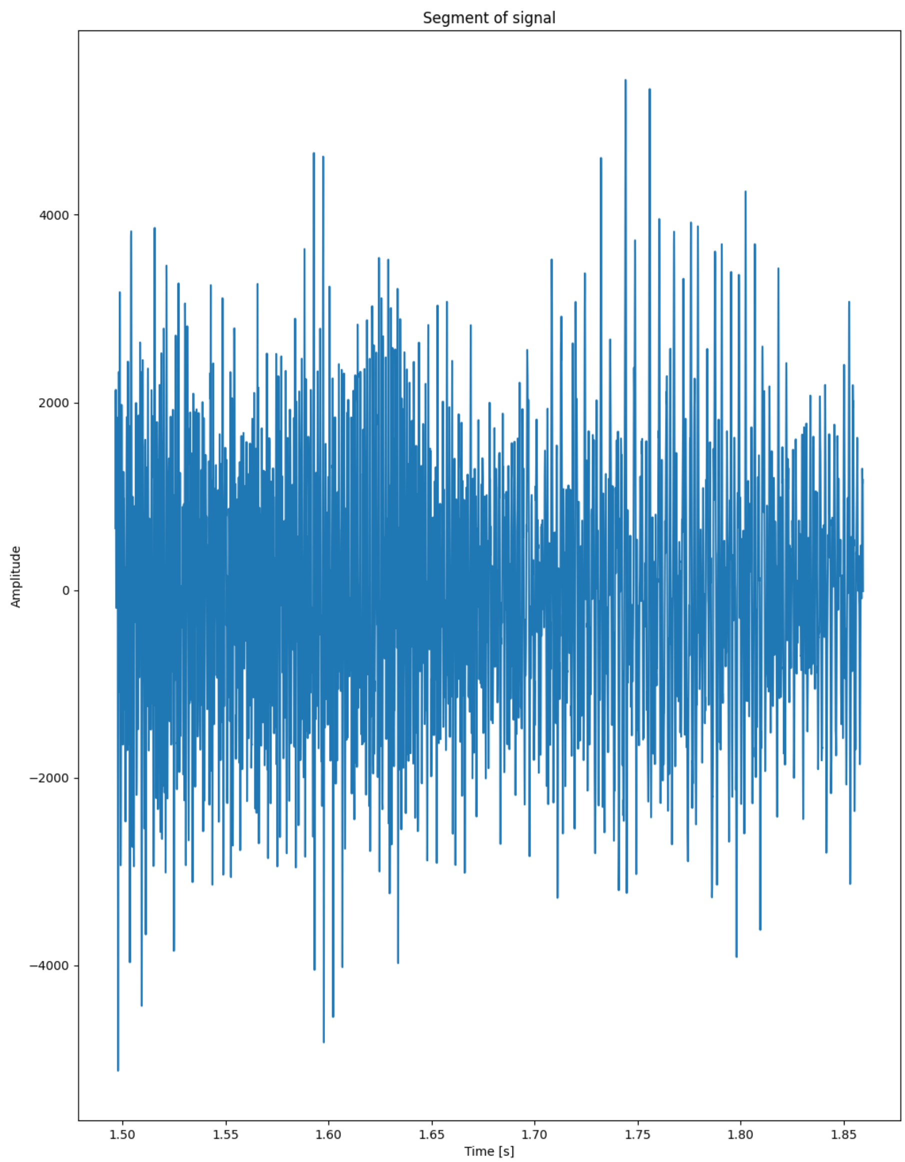

Figure 6: Plot of Segment of Signal

Part 3

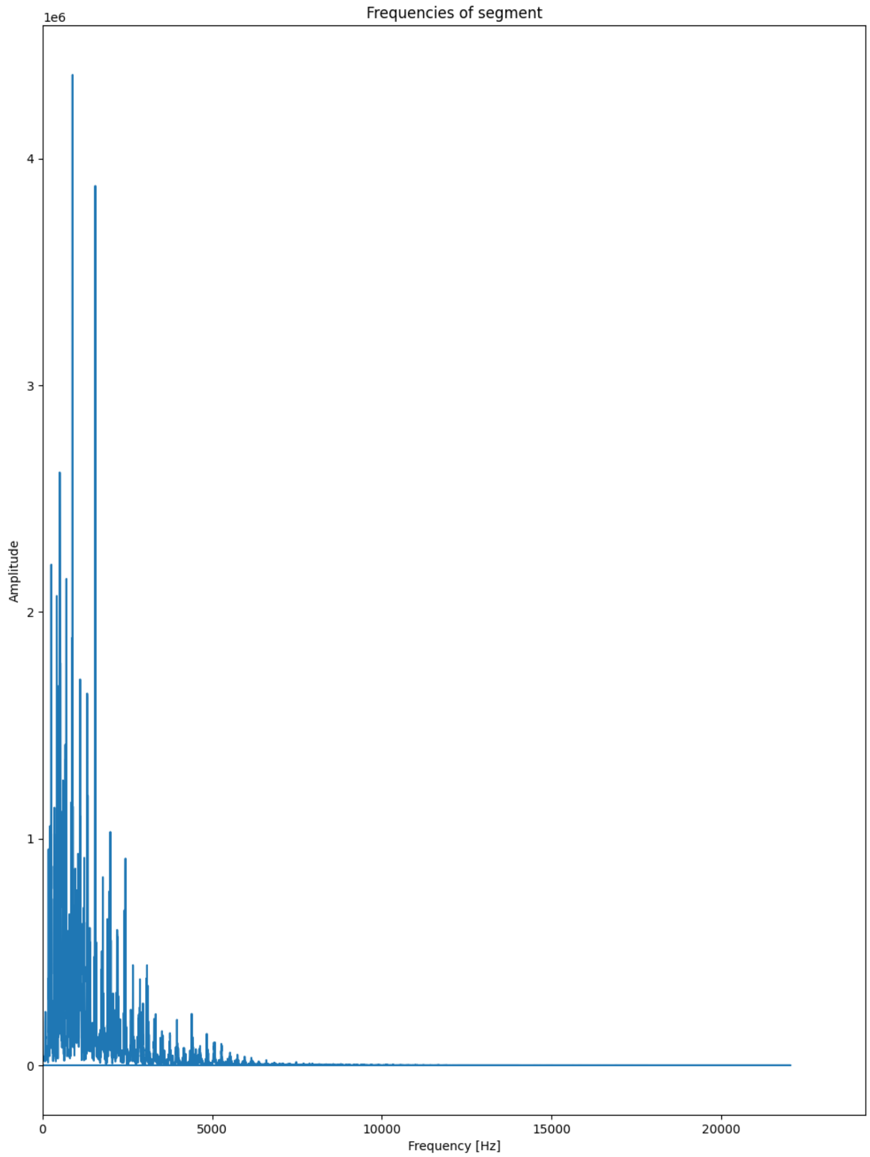

Figure 7: Frequencies of Segment of Signal

Part 4

```python
    def pitch():
        ...
        positive_freqs = fftf_seg[: fftf_seg.size // 2]
        positive_fft = fftw_seg[: fftw_seg.size // 2]

        three_highest_locs = np.argpartition(np.abs(positive_fft), -3)[-3:]

        three_highest_freqs = positive_freqs[three_highest_locs]
        three_highest_vals = positive_fft[three_highest_locs]

        print("Corresponding frequencies [Hz]:", three_highest_freqs)
        print("Amplitudes of the three highest frequencies:", np.abs(three_highest_vals))
        ...

```

```
    Corresponding frequencies [Hz]: [104.7375  107.49375 124.03125]
    Amplitudes of the three highest frequencies: [ 8628544.04894822 9302906.62716424 10746634.73638791]

```

Part 5

```python
        def pitch():
            ...
            # Zero out the three highest frequencies
            fftw_seg[three_highest_locs] = 0
            fftw_seg[-three_highest_locs] = 0
            ...
```

Part 6

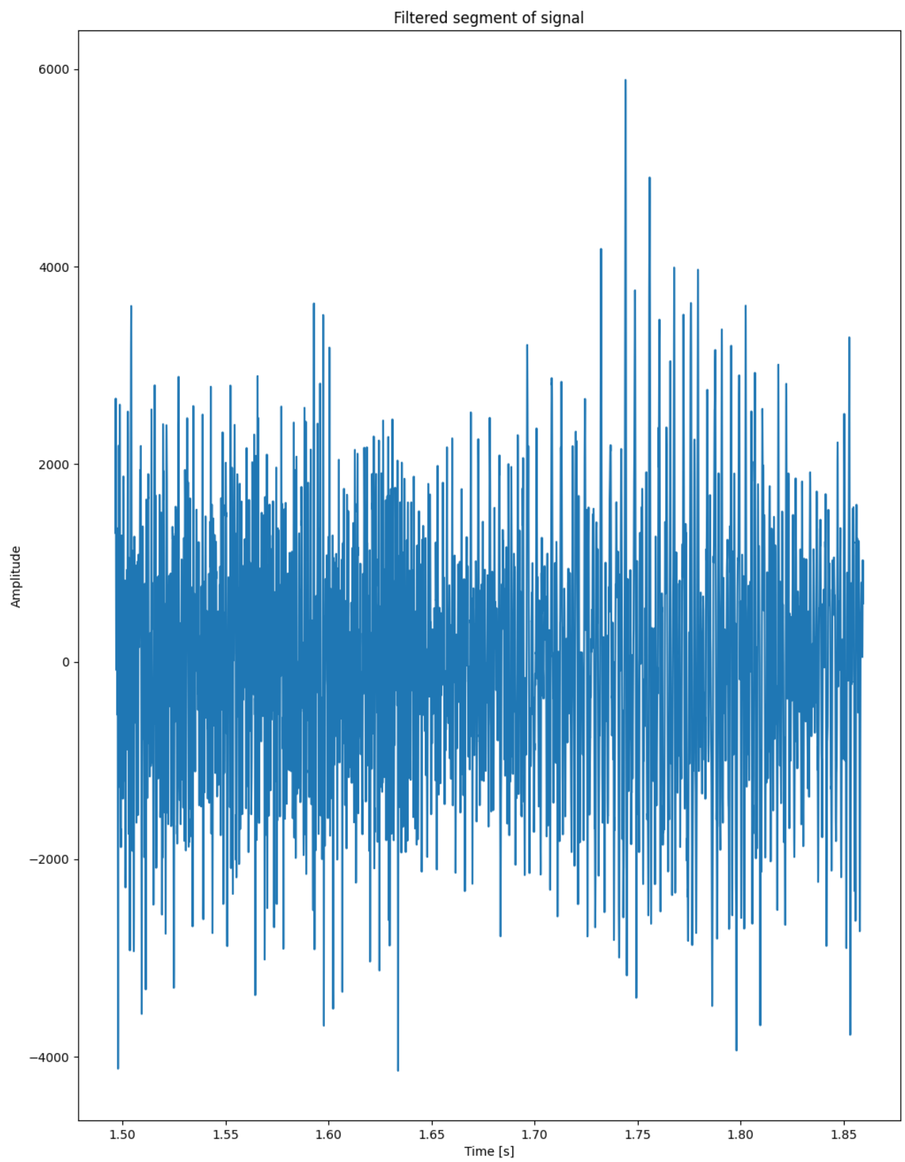

Figure 8: Plot of Filtered Segment of Signal

Part 7

```python
    def pitch():
        ...
        # Normalize and save the filtered segment as a wav file
        w_filtered_norm = (w_filtered / np.max(np.abs(w_filtered)) * 32767).astype(np.int16)

        wave.write("segment_filter.wav", sample_rate, w_filtered_norm)
        ...

```

Problem 6.3: Mixing Signals

Solution

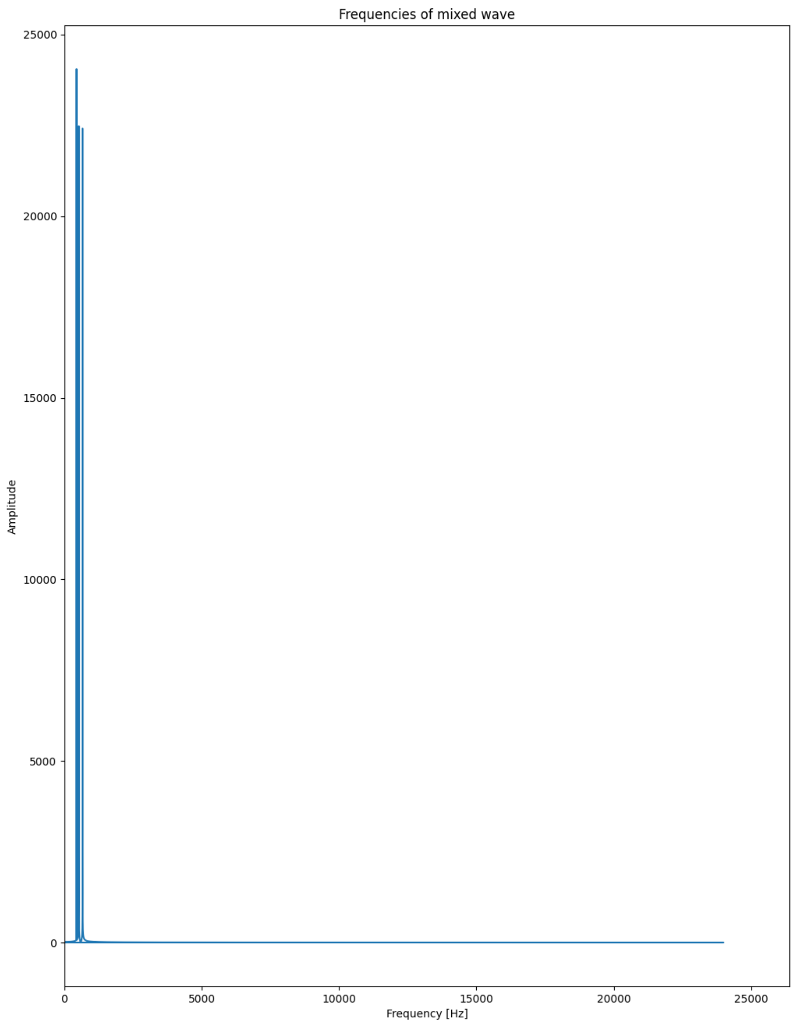

Figure 9: Frequencies of Mixed Wave

When adding frequencies that are not multiples of the fundamental frequency, you get a tone that is out of phase with the original signal. This can be tonally harsh, or it can be tonally good. How it sounds is really up to the listener, which is why music is subjective and can take on many forms.

Problem 6.4: Stretch

Solution

```python
def stretch(input, output, factor):
    """
    Stretch or compress a wav file by changing its sample rate.

    Parameters:
    - input: path to input wav file
    - output: path to output wav file
    - factor: stretching factor (e.g., 0.75 to compress, 1.25 to stretch)
    """
    sample_rate, w_input = wave.read(input)
    wave.write(output, int(sample_rate * factor), w_input)

```

Problem 6.5: Sampling

Solution

Part 1

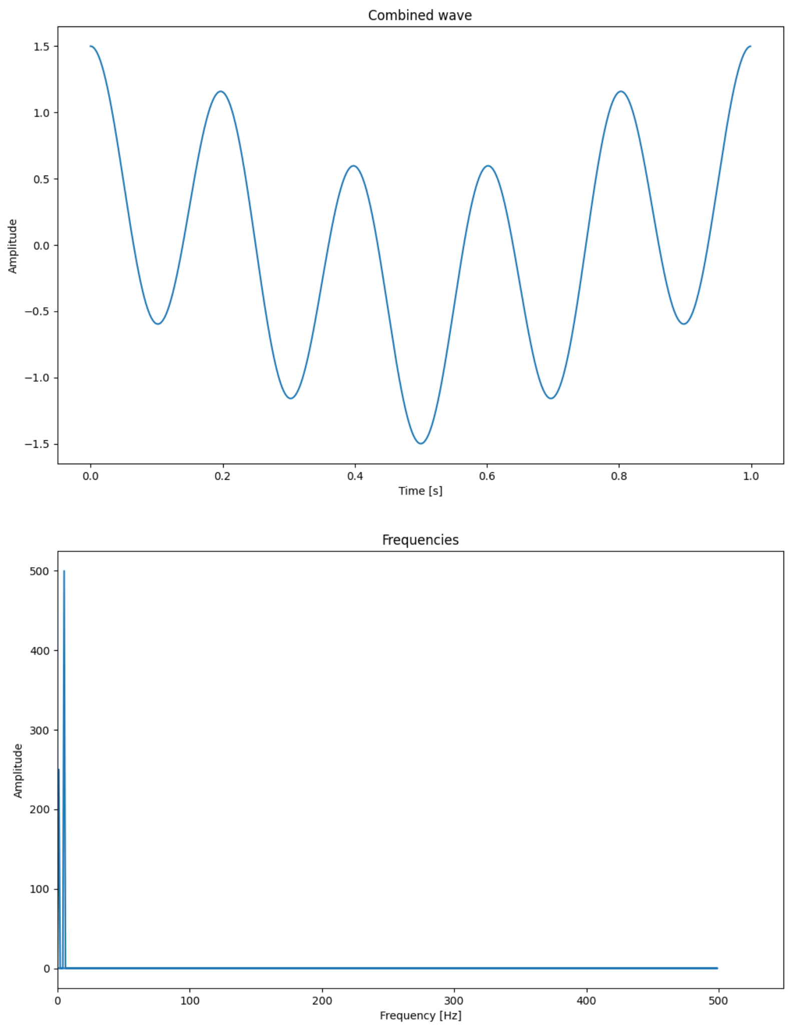

Figure 10: Plot of Provided Wave and Frequencies

Part 2

```python
def sampling():
    ...
    # Subsample the signal
    F = 125  # new sample rate in Hz
    subsample_factor = int(sample_rate / F)

    w_sub = w_og[::subsample_factor]  # subsampled wave
    t_sub = t_og[::subsample_factor]  # subsampled time vector
    ...
```

Part 3

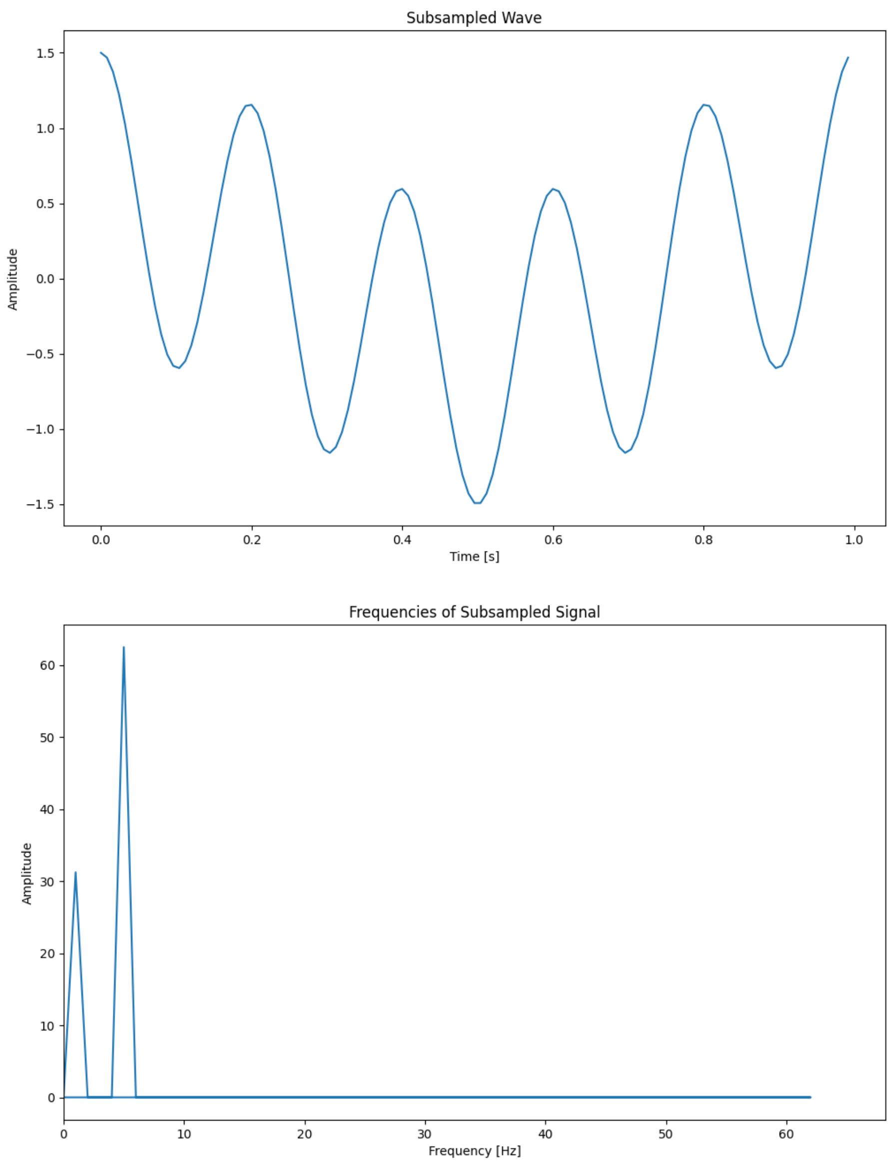

Figure 11: Plot of Sub-Sampled Wave and Frequencies

Part 4

The signal starts degrading at all frequencies below the half the sampling rate (i.e. $\forall f \in f := \frac{1}{F}, F < B, B := \frac{f_s}{2}$). This is due to the bandwith of our signal being limited by the Nyquist sampling criterion[1], which defines the above mathematical model, and states that any reconstruction of a signal that was analyzed at such framerate will only have a perfect reconstruction when B, the bandlimit, is less than half of the original sampling rate (i.e. $2B < f_s$). In regular terms, this means that we are sampling at intervals that are too small to contain a complete spectrum of information, and thus, our reconstruction is negatively impacted.

[1] Ref: DOI: 10.1109/JRPROC.1949.232969