Neuroprothetics Exercise 7
CI Vocoder

Chutong Ren

02. February 2024

1 Audio signals

In this exercise, we implement a vocoder to simulate the function of a Cochlear Implant to better understand what the user hears. Initially, we used the sounddevice module in Python to record the word 'Klaustrophobie' at a sampling frequency of 21 kHz for a duration of 3 seconds. The spectrogram for the original recorded word can be seen in Figure 1.
Then, we applied thresholding to the recorded word to reduce background noise. The spectrogram for the recorded word after thresholding can be seen in Figure 2.
Next, we generated white noise matching the dimensions of the recorded speech signal, consisting of normally distributed random numbers with an average of 0 and a standard deviation of 1, as shown in Figure 3.
A spectrum can only indicate which frequencies are present in the entire recording and their amplitudes. However, it lacks temporal information and cannot tell you when these frequencies occur. A spectrogram, on the other hand, is a time-frequency analysis tool that displays the distribution and intensity of a signal's frequencies over time. The horizontal axis represents time, the vertical axis represents frequency, and the color indicates the intensity at specific time and frequency points.

2 Filter bank

A filter bank consisting of 10 filters is utilized to simulate a Cochlear Implant (CI) with 10 stimulating electrodes, where each filter is designed to isolate and process a distinct range of frequencies. The analysis covers a frequency range from 100 Hz to 8 kHz. To begin, the border frequencies are determined using the np.logspace function in Python, specified as: np.logspace(np.log10(100), np.log10(8000), num=11). These border frequencies are depicted in Figure 4.

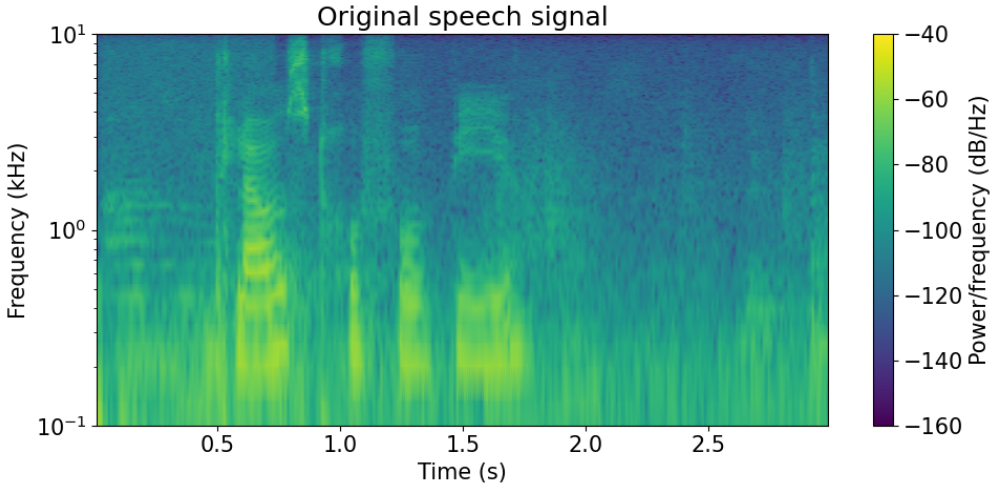

Figure 1: Spectrogram of a original speech signal

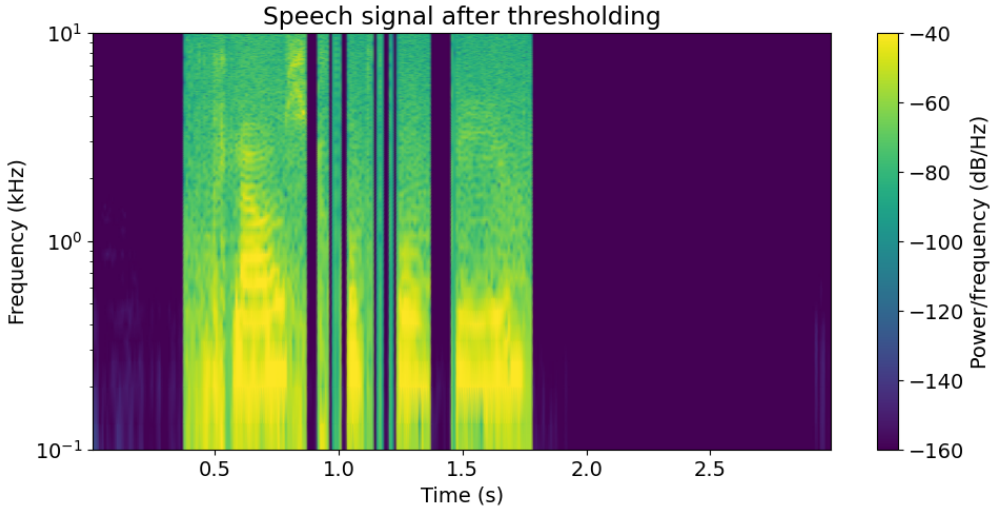

Figure 2: Spectrogram of a thresholded speech signal

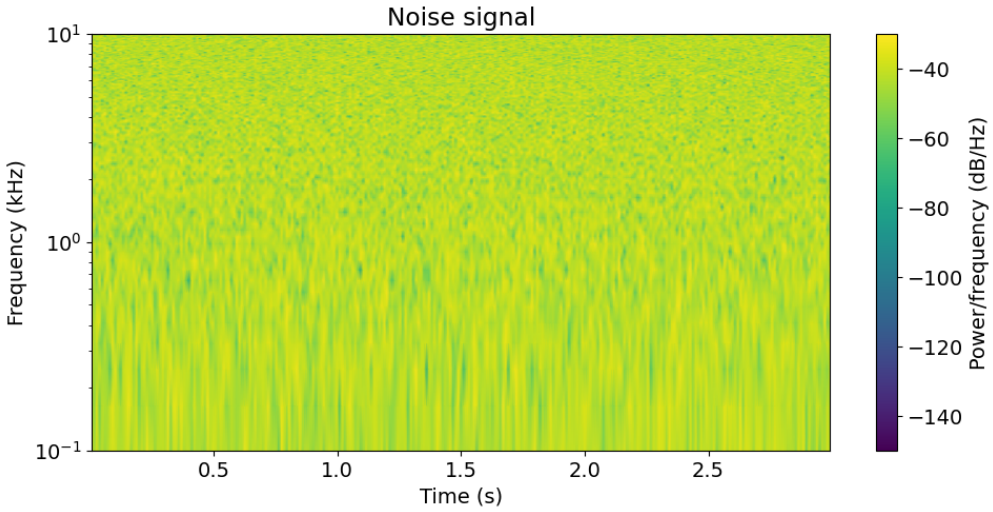

Figure 3: Spectrogram of a noise signal

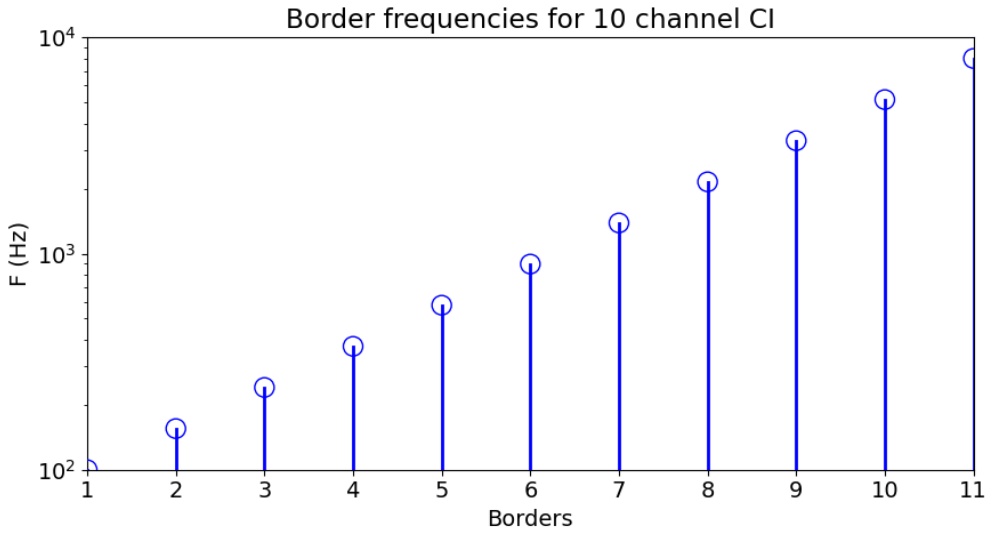

Figure 4: Border frequencies of a 10-channel filter bank

2.1 Magnitude response of a 10-channel filter bank

Among the various filtering methods available, such as the Fast Fourier Transform (FFT), Laplace Transform(LT), etc. In this exercise, we use a fourth-order Butterworth digital filter with -3 dB at the corresponding border frequencies for this exercise. It is important to note that when implementing a bandpass filter, the actual order of the filter will be double the value specified by the 'N' parameter. Therefore, to achieve a fourth-order Butterworth digital filter, the 'N' parameter should to be set to 2 in the butter function, as follows: butter(N=2, Wn=[low, high], btype='band'). The magnitude response of the filter bank can be seen in 5.

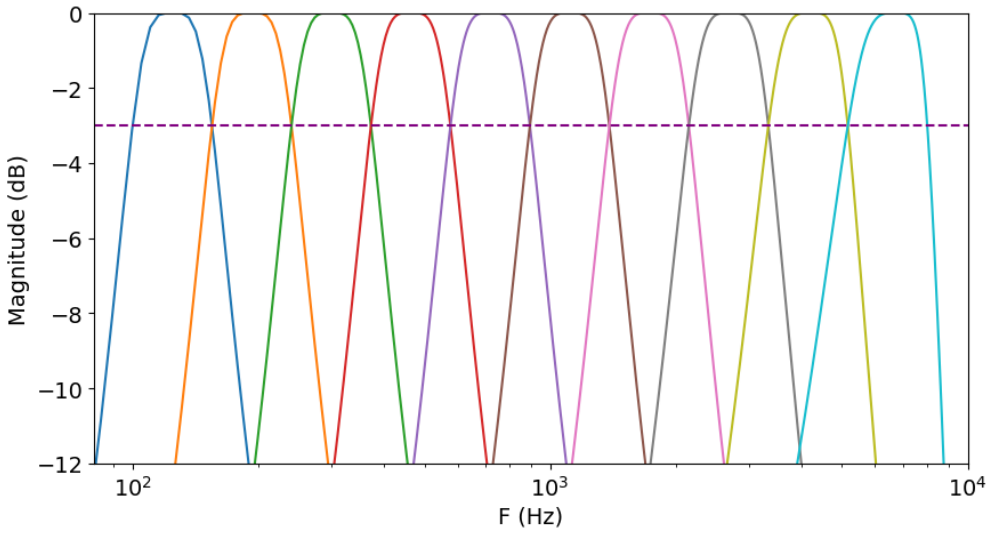

Figure 5: Magnitude response of a 10-channel filter bank

2.2 Filtering

Then, both the speech signal and the noise signal are filtered by the filter bank discussed in 2.1, resulting in 10 band-limited speech channels and 10 band-limited noise channels. The spectrograms of these band-limited signals, specifically for the channels with the second lowest and second highest border frequencies (filters 2 and 9), for both speech and noise, are displayed separately in Figures 6, 7, 8, and 9. After listening to the two band-limited speech signals plotted, I found the second band-filter channel extract lower frequency signals, which sounds lowerfreq, for example vowels, like "au", "o" and "ie" in "Klaustrophobie", while the ninth band-filter channel extract higher frequency signals, which sounds higherfreq, for example consonant, like "s", "ph" and "b" in "Klaustrophobie". The spectrogram of the original speech signal reveals the energy distribution of specific frequencies at different time intervals, which corresponds to various phonemes in speech. For instance, from 0.5s to 0.7s, there is a high intensity in the low-frequency

range, likely indicating the "au" sound. Between 0.8s and 0.9s, the high-frequency intensity peaks, corresponding to the consonant "s". The low frequencies are emphasized again at 1.1s and from 1.3s to 1.4s, corresponding to the "o" sound. Lastly, in the interval from 1.5s to 1.8s, there is an increase in low-frequency intensity, potentially representing the "ie" sound.

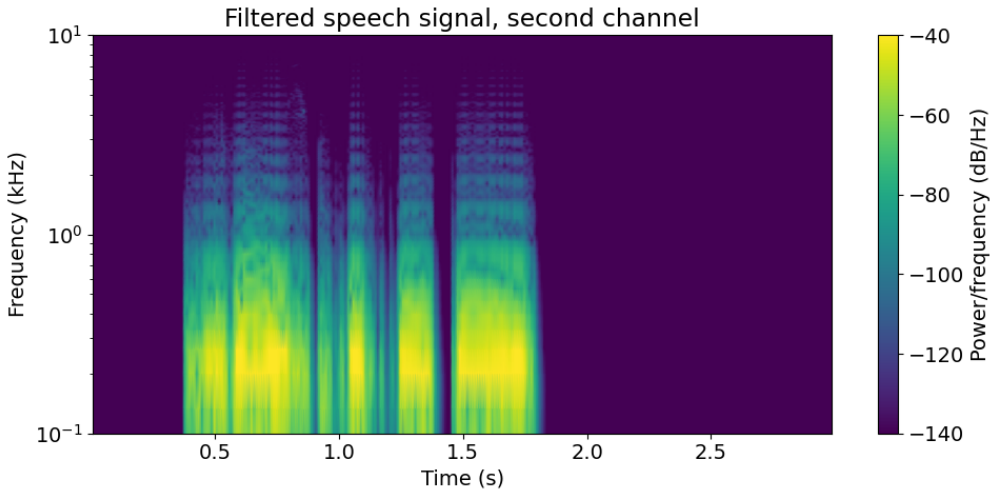

Figure 6: Spectrogram of a speech signal filtered by the second lowest frequency channel

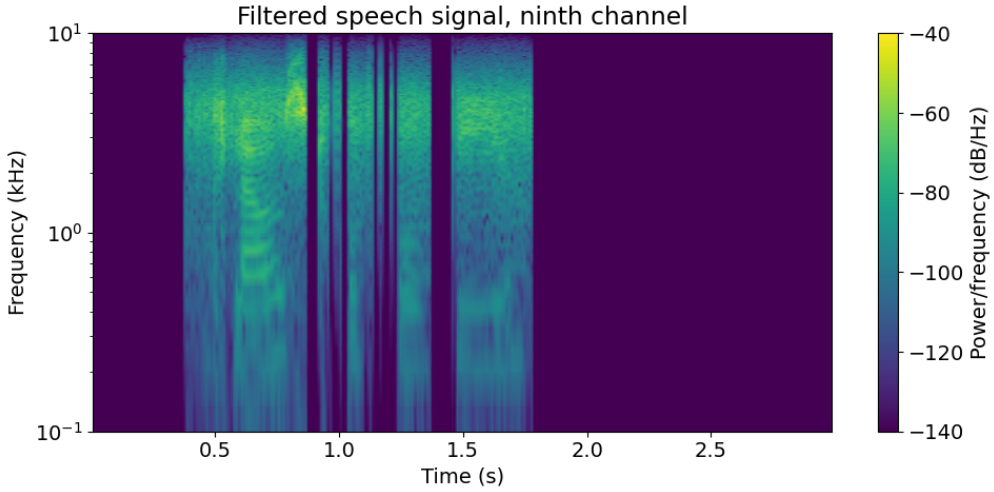

Figure 7: Spectrogram of a speech signal filtered by the second highest frequency channel

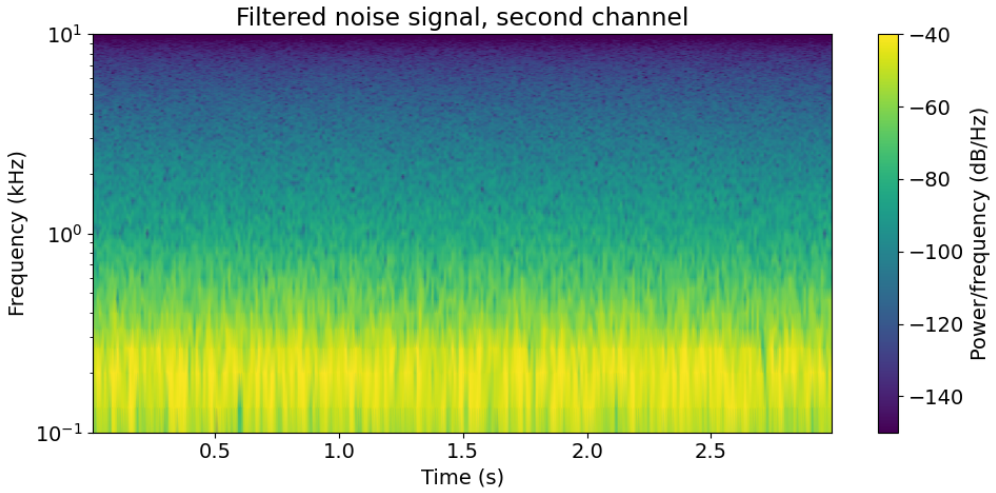

Figure 8: Spectrogram of a noise signal filtered by the second lowest frequency channel

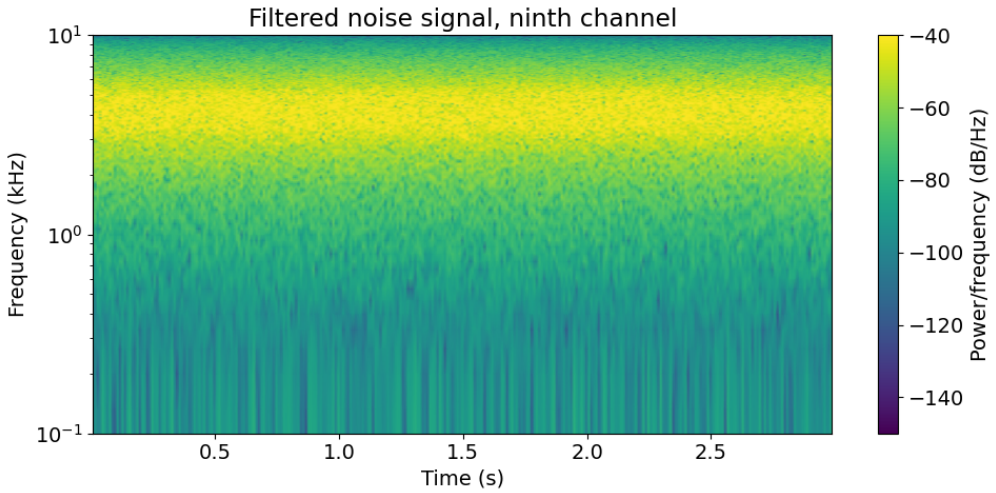

Figure 9: Spectrogram of a noise signal filtered by the second highest frequency channel

3 Vocoder

In order to implement the vocoder, we first extract the envelope of the filtered speech signal using the hilbert transform function in Python. Subsequently, we compress the signal using the equation 1 to adjust the dynamic range from the threshold (THR) to the Most Comfortable Level (MCL). Finally, we modulate the band-filtered noise with the respective envelopes. Thereafter, we sum up all the 10 channels to obtain the final vocoded signal, as depicted in Figure 10. Compared to the spectrogram of the original audio signal, Figure 10 shows that the harmonic structure is no longer as clear or regular. This indicates that harmonic details are lost during the vocoding process, affecting the naturalness and quality of the sound.

$$env_{compressed} = \frac{\log_{10}(1 + 300 \cdot env)}{\log_{10}(1 + 300)} \tag{1}$$

After listening to both the original speech signal and the vocoded signal, I found that the original speech signal is typically clearer, more natural, and easier to comprehend, whereas the vocoded signal sounds more synthesized and mechanical. This is due to the compression of dynamic range and reduction in frequency resolution, resulting in the loss of certain details in the speech signal. Consequently, this leads to decreased clarity and intelligibility.

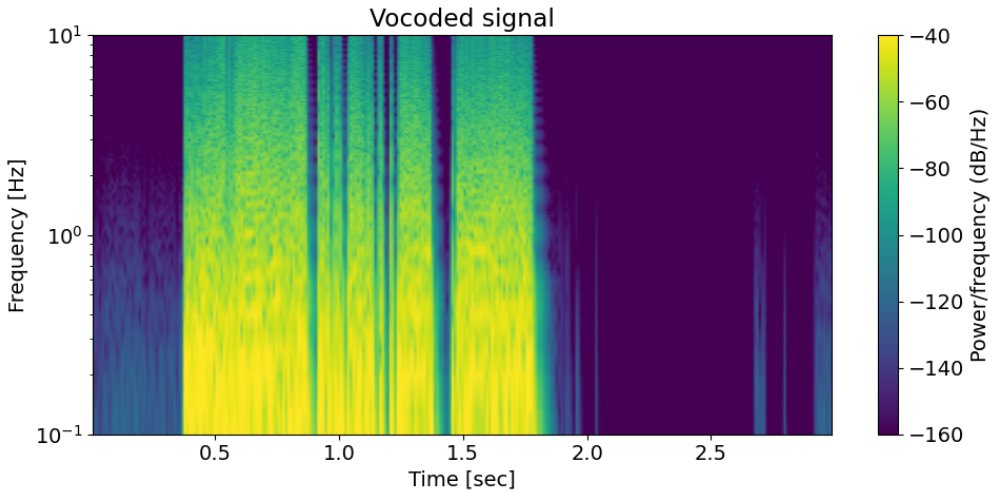

Figure 10: Spectrogram of the vocoded signal