NOISE REDUCTION

It aims to enhance the quality of a signal or image by minimizing interference, disturbances

01 FILTERS

02 WAVELET

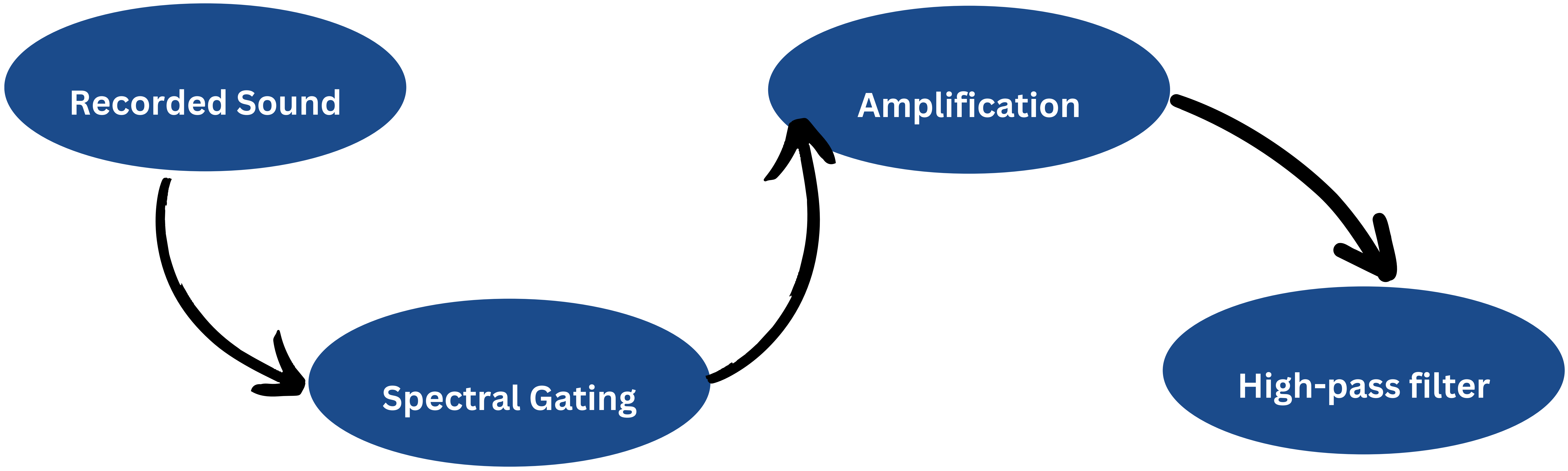
03 WEINER FILTER

04 HIGH PASS

05 SPECTRAL SUBSTRACTION

06 SPECTRAL GATING

WORKING SETUP OF NOISE REDUCTION

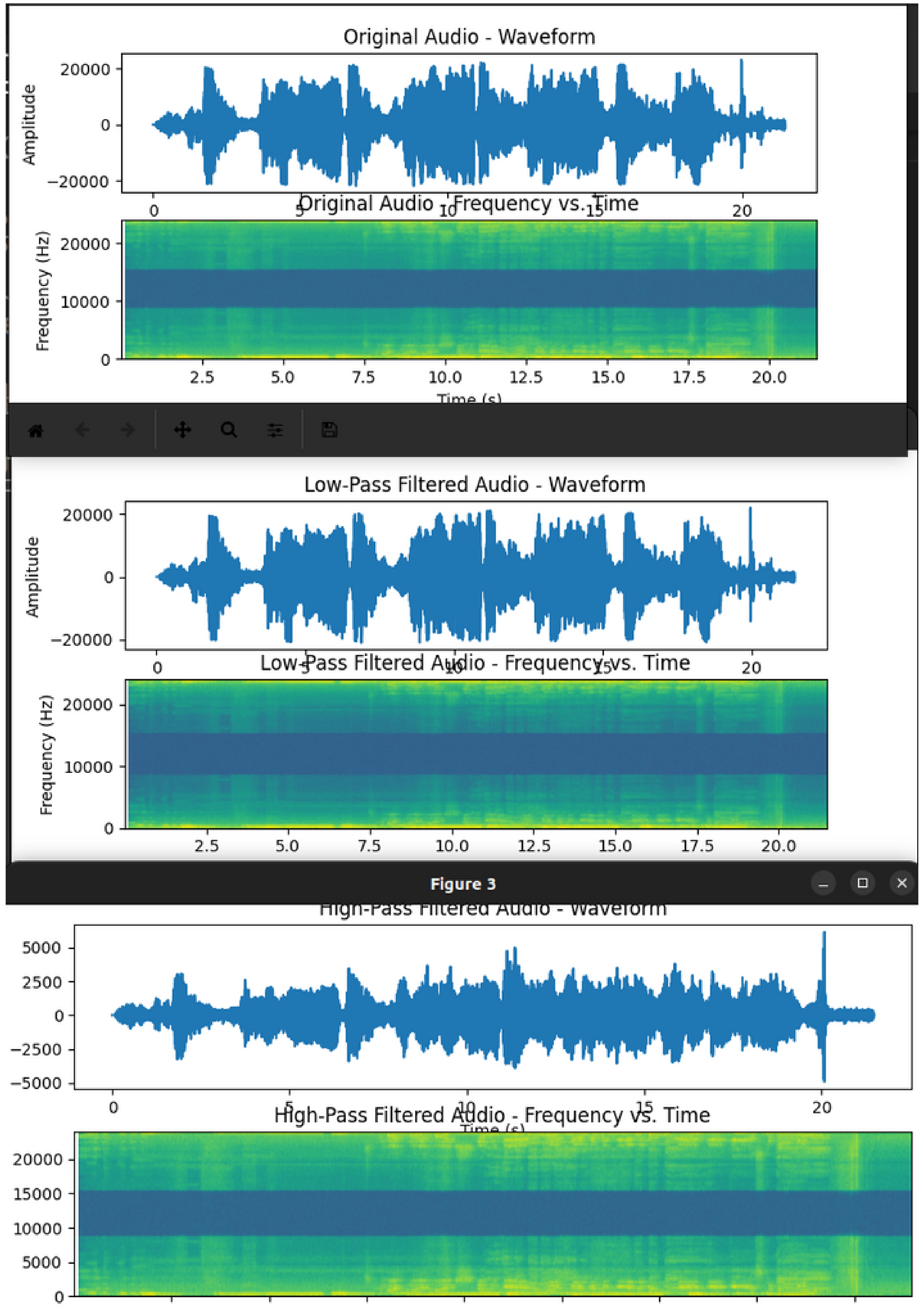


LOW PASS,HIGH PASS

A low-pass filter allows frequencies below a certain cutoff frequency to pass through, attenuating higher frequencies. In the provided code, the original audio is filtered to retain frequencies below 1500 Hz, resulting in a low-pass filtered audio signal.

A high-pass filter allows frequencies above a specified cutoff frequency to pass through, attenuating lower frequencies. In the code, the original audio is also processed to preserve frequencies above 3000 Hz, creating a high-pass filtered audio signal.

HIGH PASS is used here to remove the heart beats from the recorded lung sounds. Its working fine.




```python
from pydub import AudioSegment
import matplotlib.pyplot as plt
import numpy as np

# Load the original audio
audio = AudioSegment.from_wav("4.wav")

# Apply a low-pass filter (remove frequencies below 1500 Hz)
low_pass_filtered_audio = audio.low_pass_filter(1500)

# Apply a high-pass filter (remove frequencies below 3000 Hz)
high_pass_filtered_audio = audio.high_pass_filter(3000)

# Export the low-pass and high-pass filtered audio to separate files
low_pass_filtered_audio.export("low_pass_filtered.wav", format="wav")
high_pass_filtered_audio.export("high_pass_filtered.wav", format="wav")

# Function to plot waveform and frequency vs. time in one tab
def plot_waveform_and_frequency_vs_time(audio_segment, title):
    samples = np.array(audio_segment.get_array_of_samples())
    sample_rate = audio_segment.frame_rate

    # Create a figure with two subplots
    fig, axes = plt.subplots(2, 1, figsize=(12, 8))

    # Plot waveform
    axes[0].plot(np.linspace(0, len(samples) / sample_rate, num=len(samples)), samples)
    axes[0].set_title(f'{title} - Waveform')
    axes[0].set_xlabel('Time (s)')
    axes[0].set_ylabel('Amplitude')

    # Plot frequency vs. time
    Pxx, freqs, times, im = axes[1].specgram(samples, Fs=sample_rate, cmap='viridis')
    axes[1].set_title(f'{title} - Frequency vs. Time')
    axes[1].set_xlabel('Time (s)')
    axes[1].set_ylabel('Frequency (Hz)')

    return fig

# Create a figure for all plots

# Plot waveform and frequency vs. time for the original, low-pass, and high-pass filtered audio
fig_all = plot_waveform_and_frequency_vs_time(audio, 'Original Audio')
fig_all = plot_waveform_and_frequency_vs_time(low_pass_filtered_audio, 'Low-Pass Filtered Audio')
fig_all = plot_waveform_and_frequency_vs_time(high_pass_filtered_audio, 'High-Pass Filtered Audio')

# Show the figure after all plots have been added
plt.tight_layout()
plt.show()
```

WAVELET DENOISING

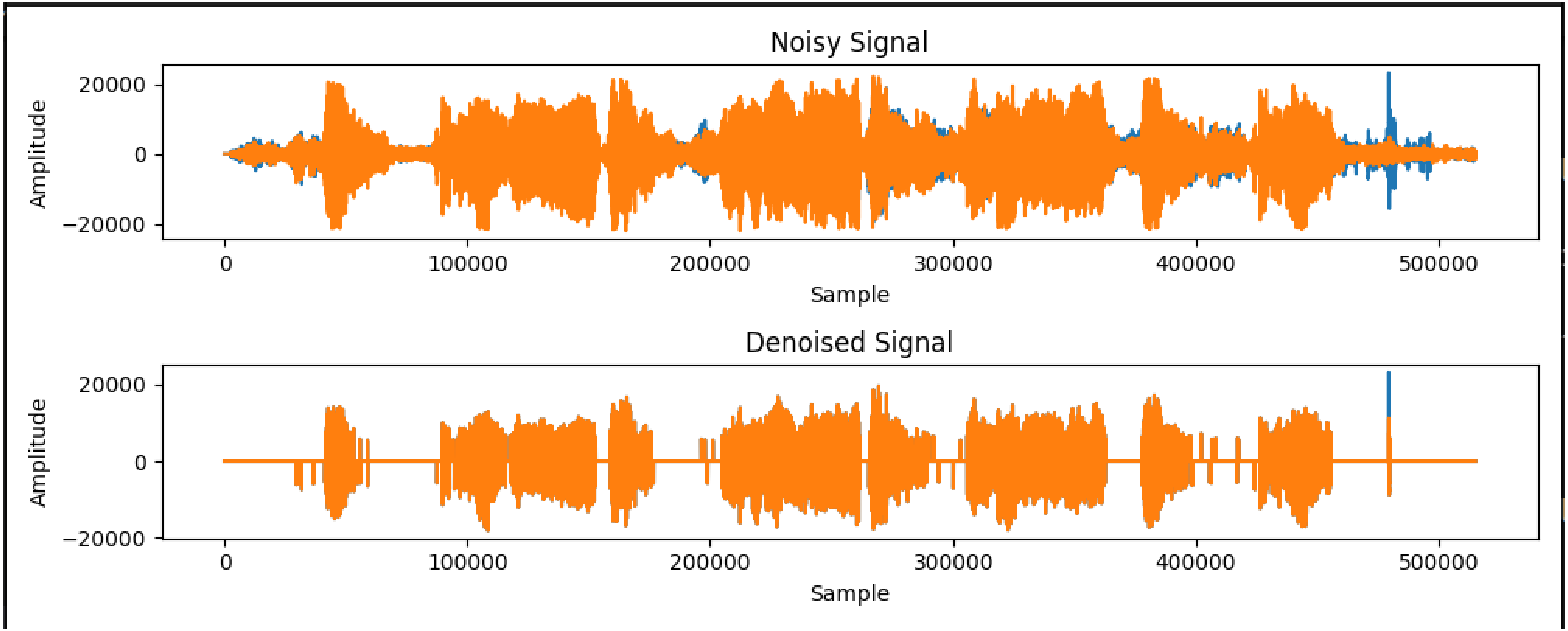
Wavelet is a mathematical tool used in signal processing and image compression. It decomposes a signal into components at different frequency bands, capturing both high and low-frequency information.

This allows for a more localized analysis of signals compared to traditional Fourier transform methods, making wavelets particularly useful for tasks such as denoising and feature extraction.

Its failing here, because its adding other noise in place of the original noise
its removing the noise to some extent but its disturbing the lung sounds too

```
import numpy as np
import pywt
import pywt.data
import scipy.io.wavfile as wavfile
sample_rate, noisy_signal = wavfile.read('4.wav')

wavelet = 'haar'

level = 3

threshold_mode = 'hard'

coeffs = pywt.wavedec(noisy_signal, wavelet, level=level)

threshold_value = np.std(coeffs[-1]) * np.sqrt(2 * np.log(len(noisy_signal)))

coeffs = [pywt.threshold(c, threshold_value, mode=threshold_mode) for c in coeffs]

denoised_signal = pywt.waverec(coeffs, wavelet)

wavfile.write('_song_.wav', sample_rate, denoised_signal.astype(np.int16))

import matplotlib.pyplot as plt

# Plot the input signal
plt.figure(figsize=(10, 4))
plt.subplot(2, 1, 1)
plt.plot(noisy_signal)
plt.title('Noisy Signal')
plt.xlabel('Sample')
plt.ylabel('Amplitude')

# Plot the denoised signal
plt.subplot(2, 1, 2)
plt.plot(denoised_signal)
plt.title('Denoised Signal')
plt.xlabel('Sample')
plt.ylabel('Amplitude')

plt.tight_layout()
plt.show()
```

https://www.sciencedirect.com/topics/computer-science/wavelet-denoising

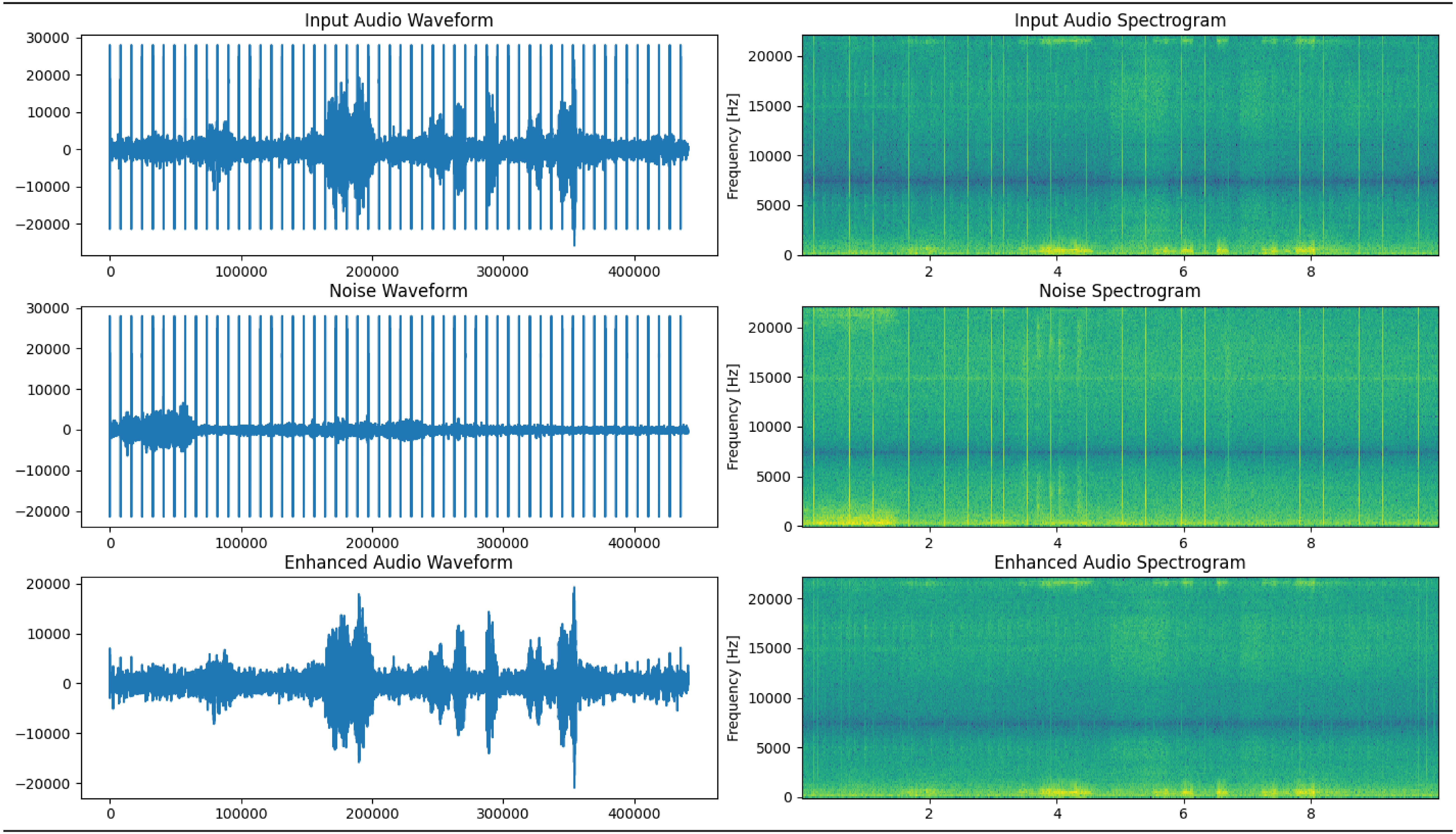

Noisy Signal
Amplitude
20000
0
−20000
0
100000
200000
300000
400000
500000
Sample
Denoised Signal
Amplitude
20000
0
−20000
0
100000
200000
300000
400000
500000
Sample

SPECTRAL SUBSTRACTION

It aims to enhance the quality of a signal by subtracting an estimated noise spectrum from the original signal's spectrum, leaving behind a cleaner version.

This method relies on assuming that noise is additive and stationary, enabling the suppression of unwanted noise components from the signal's frequency domain representation, often resulting in improved intelligibility and clarity.

Here, its subtracting the known noise- Mic’s beat from the recorded lung sounds.



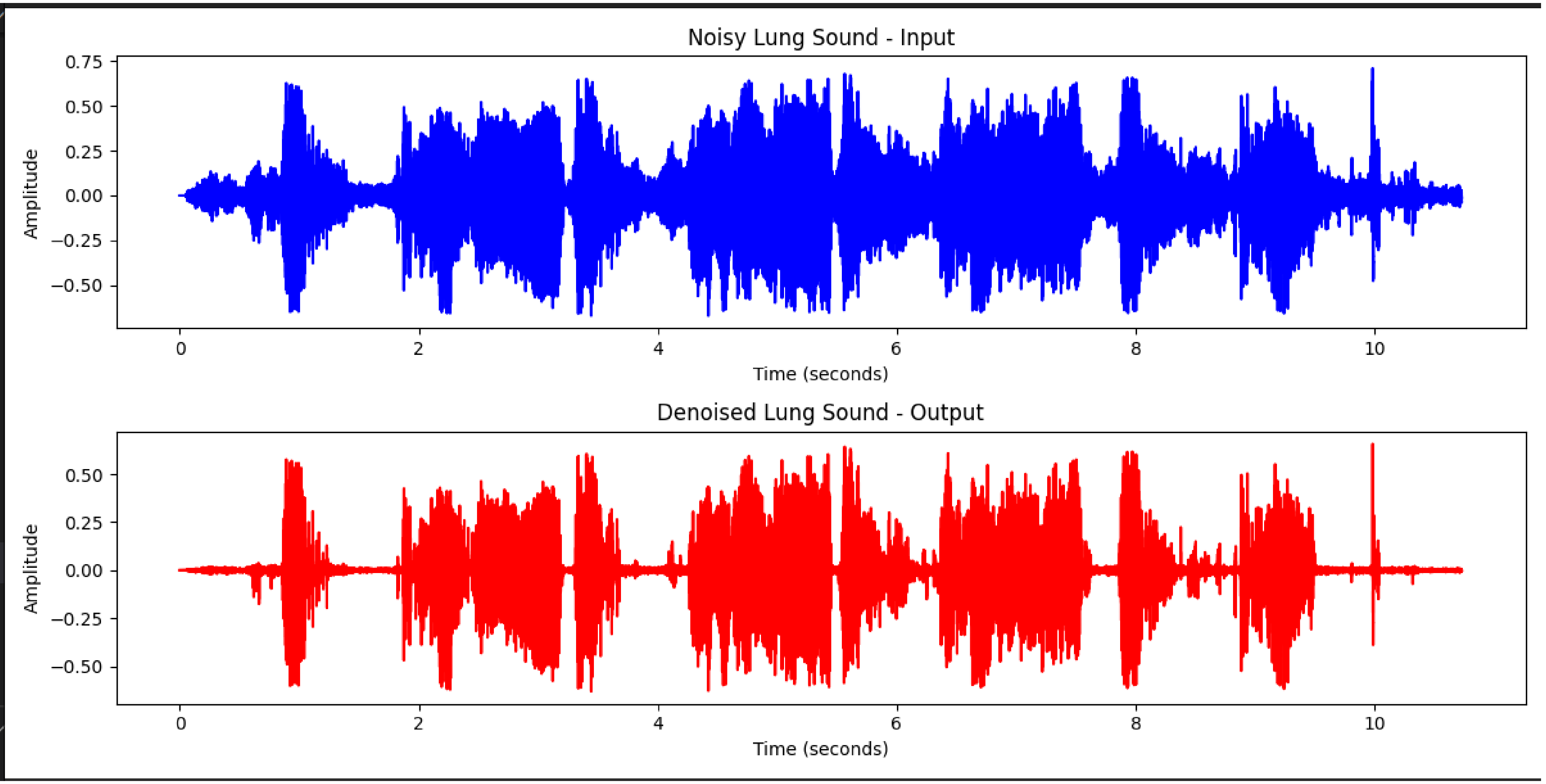
Input Audio Waveform
Input Audio Spectrogram
Noise Waveform
Noise Spectrogram
Enhanced Audio Waveform
Enhanced Audio Spectrogram
Frequency [Hz]
30000
20000
10000
0
−10000
−20000
100000
200000
300000
400000
15000
5000
2
4
6
8

WEINER FILTERING

It works by minimizing the mean-square error between the desired signal and the filtered output. The filter estimates the true signal by taking into account both the noisy input signal and the statistical properties of the noise.

The Wiener filter is particularly effective when the characteristics of the signal and noise are known or can be estimated accurately.
The noise is being remioved, but its not that effective

```
import soundfile as sf
from scipy.signal import wiener

# Load the noisy lung sound recording
noisy_audio, sr = sf.read('4.wav')

# Apply Wiener filtering with an adjusted width parameter
denoised_audio = wiener(noisy_audio, mysize=9)  # You can experiment with

# Save the denoised audio
sf.write('_song_.wav', denoised_audio, sr)

import matplotlib.pyplot as plt
import numpy as np

# Plot the input signal
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(np.arange(len(noisy_audio)) / sr, noisy_audio, color='b')
plt.title('Noisy Lung Sound - Input')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')

# Plot the output signal after applying Wiener filter
plt.subplot(2, 1, 2)
plt.plot(np.arange(len(denoised_audio)) / sr, denoised_audio, color='r')
plt.title('Denoised Lung Sound - Output')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')

# Adjust layout for better visualization
plt.tight_layout()

# Show the plots
plt.show()
```



https://www.owlnet.rice.edu/~elec539/Projects99/BACH/proj2/wiener.html

SPECTRAL GATING

Spectral gating is a noise reduction technique that works by selectively suppressing noise components in the frequency domain of an audio signal. It is particularly effective for removing noise that is relatively constant in frequency, such as background hum or hiss.

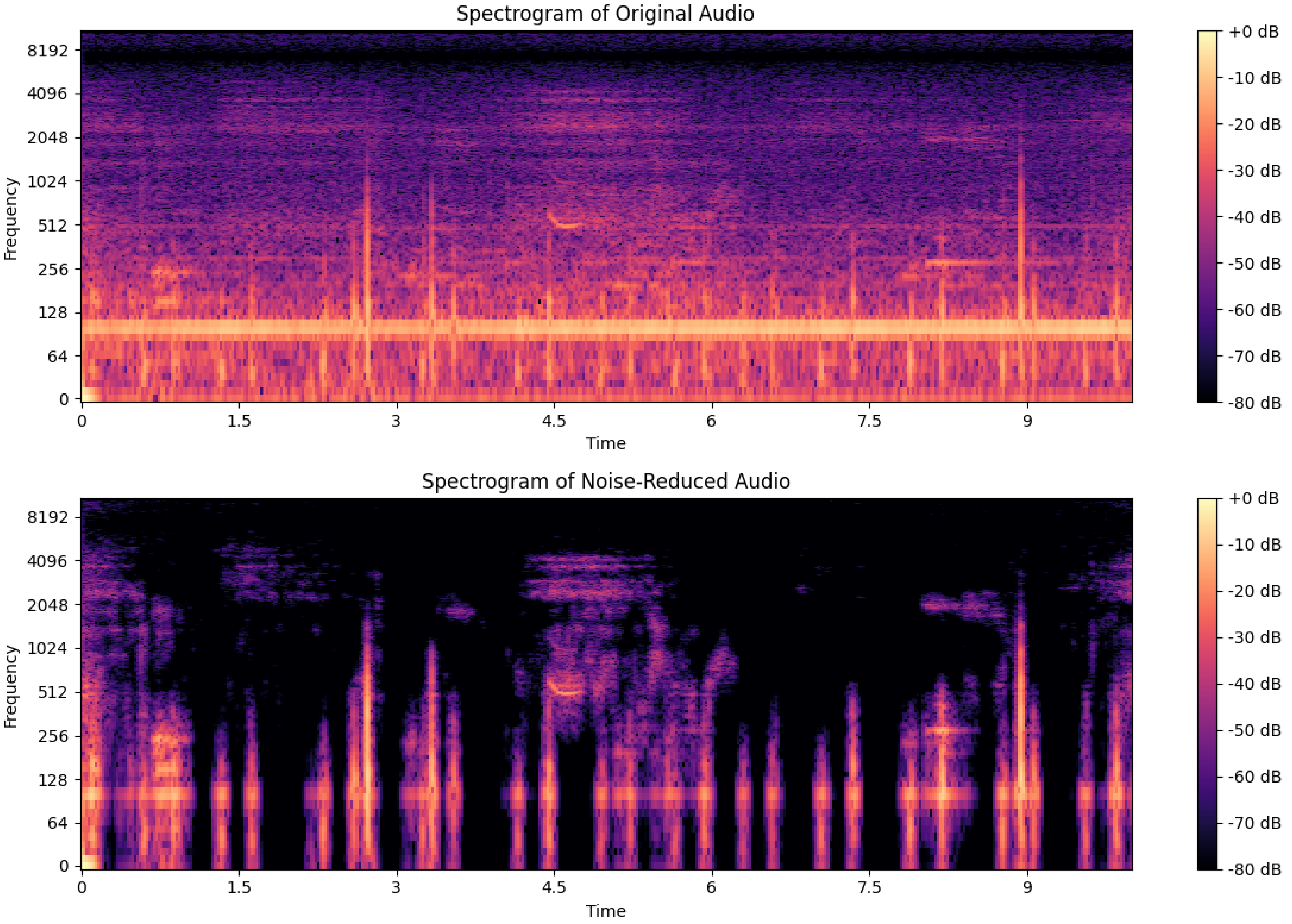
How it works:

Spectrogram: The audio signal is transformed into a spectrogram, which represents the energy distribution of the signal across different frequencies.

Noise Threshold: A noise threshold is estimated, typically based on the median or mean of the noise spectrogram. This threshold determines which frequency components are considered noise and which are considered signal.

Gating: The signal spectrogram is gated by setting the magnitude values of frequencies below the noise threshold to zero. This effectively removes noise components from the spectrogram.

Inverse Transform: The gated spectrogram is then inverse transformed back into the time domain, resulting in the noise-reduced audio signal.



Spectrogram of Original Audio
Frequency
8192
4096
2048
1024
512
256
128
64
0
Time
0
1.5
3
4.5
6
7.5
9
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB
Spectrogram of Noise-Reduced Audio
Frequency
8192
4096
2048
1024
512
256
128
64
0
Time
0
1.5
3
4.5
6
7.5
9
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB