Noise removing and amplifying method using Wavelet Transform

Wavelet transform is a mathematical tool used for analyzing signals and images in terms of different frequency components. Unlike the Fourier transform, which represents a signal in the frequency domain, the wavelet transform represents both frequency and time information. This makes wavelet transform particularly useful for analyzing signals with non-stationary or time-varying characteristics.

Here's a brief overview of the wavelet transform and its application in signal denoising:

Wavelet Transform:

The wavelet transform decomposes a signal into a set of basis functions called wavelets. These wavelets are small, localized functions that are used to analyze different components of a signal at various scales and positions. The continuous wavelet transform (CWT) and the discrete wavelet transform (DWT) are two common types.

1. Continuous Wavelet Transform (CWT):
- CWT involves convolving the signal with a scaled and translated version of the mother wavelet function at every point in time.
- It provides a continuous view of the signal's frequency content over time.

2. Discrete Wavelet Transform (DWT):
- DWT is a sampled version of CWT and is more computationally efficient.
- It involves a process of iteratively decomposing a signal into approximation and detail coefficients at different scales.

Signal Denoising using Wavelet Transform:

Wavelet transform is widely used for denoising signals, especially when the noise characteristics vary across different frequency bands. The general process involves the following steps:

- Decomposition:
- Apply the DWT to decompose the signal into approximation and detail coefficients at multiple scales.
- The approximation coefficients represent the coarse, low-frequency components, and the detail coefficients represent the high-frequency details.
- Thresholding:

- Threshold the detail coefficients to remove or attenuate noise. Soft or hard thresholding techniques are commonly used.
- Soft thresholding sets coefficients below a certain threshold to zero and attenuates others, providing a smoother denoised signal.
- Hard thresholding sets coefficients below a threshold to zero, effectively removing noise.
- Reconstruction:
- Reconstruct the denoised signal using the modified coefficients.
- The approximation coefficients from the highest scale can be omitted if high-frequency details are not crucial.
- Iterative Processing (Optional):
- The above steps can be iteratively applied to further enhance the denoising performance.

Wavelet-based denoising is effective when the signal and noise have different frequency characteristics. Choosing an appropriate wavelet, thresholding method, and threshold value are critical in achieving optimal denoising results.

Wavelet Transform using Python:

The denoising process using wavelet transform with python has few different steps:

- Import libraries.
- Load and Normalizing .wav files
- Applying wavelet transform to denoise signals.
- Visualizing signals
- Saving denoised .wav file

Let us see one by one.

1. Import Libraries
   Here we wanted to import some libraries for applying wavelet transform using python, handling sound files, visualizing waves and file handling.

```
from scipy.io import wavfile
import numpy as np
from skimage.restoration import denoise_wavelet
import matplotlib.pyplot as plt
import soundfile as sf
import os
import scipy.io.wavfile as wavfile
```

2. Load and normalize .wav files.
   The reason for normalizing signal is just to amplify the signal.

```
#The example file = 00a49d6b07.wav
Fs ,x = wavfile.read("00a49d6b07.wav")
x=x/max(x) #Normalizing
```

3. Applying wavelet transform to denoise signals.

```
x_denoise = denoise_wavelet(x, method='VisuShrink', mode='soft', wavelet_levels=3, wavelet='sym8', rescale_sigma='True')
```

4. Visualizing signals

```
plt.figure(figsize=(20, 10), dpi=100)
plt.plot(x)
plt.plot(x_denoise)
```

5. Saving denoised .wav file

```
sf.write('denoise.wav', x_denoise, 16000)  #Writing denoise file into .wav form
```

Note:

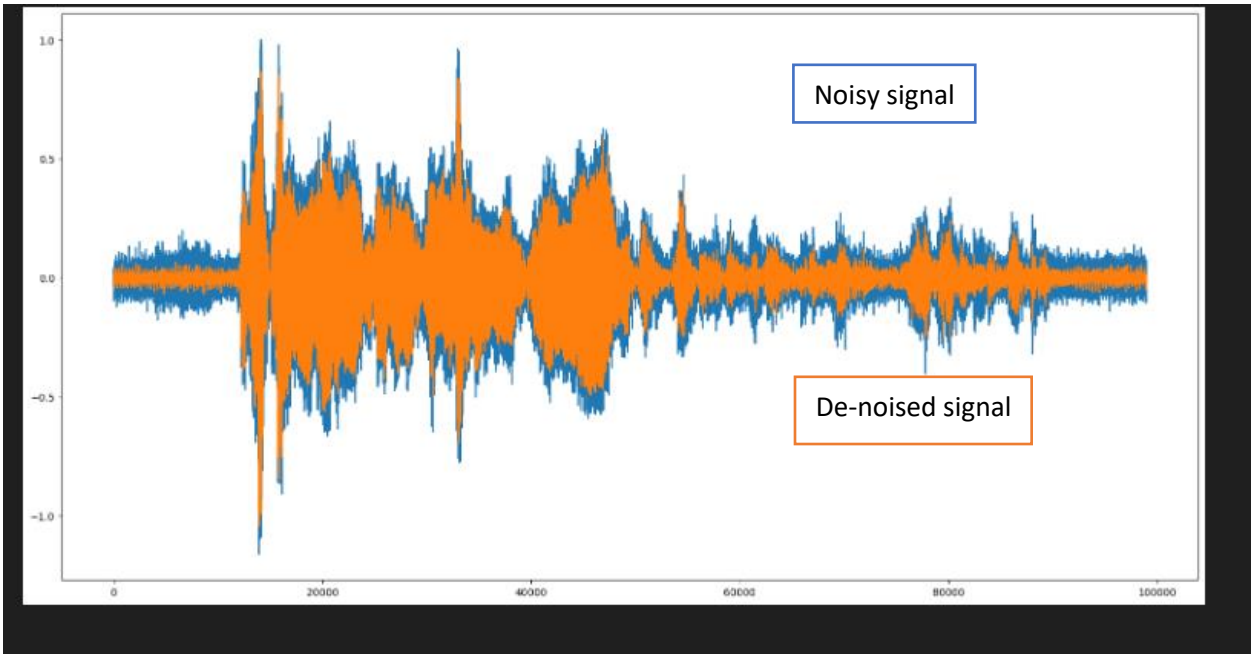
The sampling rate used in both noisy and de-noised signal is 16kHz. There for we did not experience any time scaling.

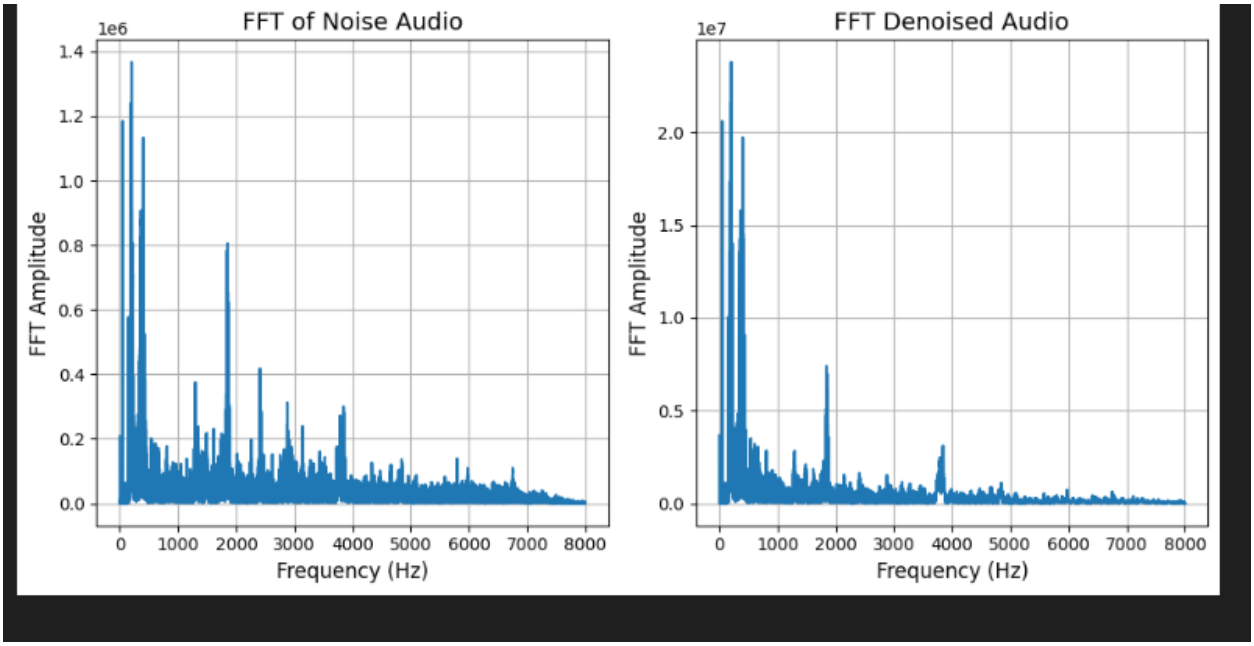
Let us see how we applied the above process for the whole set of signals:

```python
folder_path = "C:/Wavelet Transform/noise audio"  # Replace with the actual path to folder
output_folder = "C:/Wavelet Transform/denoise audio"

wav_file_paths = []

# Loop through files in the specified folder
for root, directories, files in os.walk(folder_path):
    for file in files:
        if file.endswith(".wav"):
            file_path = os.path.join(root, file)
            wav_file_paths.append(file_path)

# Create the output folder if it doesn't exist
os.makedirs(output_folder, exist_ok=True)

# Denoise each .wav file and save in the output folder
for file_path in wav_file_paths:
    Fs, x = wavfile.read(file_path)
    x_denoise = denoise_wavelet(x, method='VisuShrink', mode='soft', wavelet_levels=3, wavelet='sym8', rescale_sigma='True')
    x_denoise = x_denoise / max(x_denoise)
    filename = os.path.splitext(os.path.basename(file_path))[0]
    output_file_path = os.path.join(output_folder, filename + ".wav")
    sf.write(output_file_path, x_denoise, 16000)  # Use soundfile for writing
```

After going through this process, we could denoise all the signals using wavelet transform.

Just for having an idea, we have provided the time domain and frequency domain representations of one noisy and denoised signal below:





Reference:

- https://dergipark.org.tr/tr/download/article-file/496722#:~:text=In%20order%20to%20de%2Dnoise,coefficients%20at%20different%20frequency%20levels.
- https://ieeexplore.ieee.org/document/8404418
- https://www.mathworks.com/help/wavelet/ug/wavelet-denoising.html