# Hiss Reduction: An Exploration on the implementation of Frequency-Domain Noise-Reduction Techniques

Huang, Vincent(Chenzhun)
Schulich School of Music
School of Computer Science
McGill University
Montreal, Canada
chenzhun.huang@mail.mcgill.ca

Depalle, Phillippe
Schulich School of Music
McGill University
Montreal, Canada

*Abstract*—**Noise reduction techniques are of great importance in not only audio signal processing but also applicable to fields such as computer vision, image processing, etc. A high-quality algorithm in such purpose enables us to restore noisy, corrupted, old media and achieve better information retrieval. There are many types of noise present in samples. This paper gives a broad summarizations of different hiss (a kind of noise) reduction techniques and attempts the implementations of two of the most common audio spectral-domain hiss reduction algorithms. An evaluation on the output and performance is also conducted to compare the two techniques. In addition, possible future development of hiss reduction is discussed in the end.**

***Keywords—digital signal processing, noise reduction, frequency domain processing, short-time Fourier transform,***

## I. Introduction

With the advancement of digital technology, we are able to preserve data in an unprecedented scale, from every song ever performed to years of security footage. The data stored presents great values in different areas. Some of the most valuable audio recordings could not escape their own time, thus carrying a certain amount of noise which is deemed "unacceptable" in modern era. Besides, deep learning algorithms also rely on massive amount of data to perform well. However, all the data obtained must be carefully processed to maximize the performance of the algorithms. To process the samples for either purpose above require a significant amount of work, one of the most important of which is noise reduction.

There can be many types of degradation present in an audio recording. Clicks and hiss are often in the spotlight due to their constant presence in audio media. Algorithms are available for the detection and reduction of both types and both are non-trivial.

This paper will focus on the reduction techniques of one aspect of noise, hiss in audio recordings. Hiss noise often originates from "electrical circuit noise, irregularities in the storage medium and ambient noise from the recording environment" (Godsill & W., 1998) [1] and they are usually referred by users as "background noise". In the digital signal processing context, hiss noise can often be modelled as waveforms of random amplitude within a certain range (white noise). This type of noise is typically present in analogue tape recordings from early times along with click noise. (Godsill & W., 1998).

There are multiple algorithms available on the market for hiss reduction in corrupted audio signal. The first few breakthroughs happened in the late 20th centurary where some basic operations on the spectral-domian were proposed. More recently, (Deng et al., 2011) proposed a hiss reduction method for audio signals based on a Modified Discrete Cosine Transform (MDCT).

For this paper, two of these techniques were implemented, namely the Wiener's Solution and the Spectral Subtraction Technique, both of which are noise-reduction algorithms based on frequency-domain processing and utilizes similar techniques to process on signal's spectrum level. In the later part of this paper, I will go through the implementation process of these two algorithms in Matlab, including the basic structure of the repository that comes with this paper, important preconditions on the parameters in order for the algorithms to work properly. The quality of the output as well as the performance of the algorithm will be evaluated and some optimization techniques will be proposed and attempted.

Other algorithms will also be discussed with less technical details.

## II. Background Information

This section gives a brief recap on some basic concepts, which could be helpful for the understanding of hiss-reduction techniques discussed later.

Fourier transform (FT) opens up the possibility to view a time-domain signal from spatial or frequency perspective. It achieves this by a simple change of basis. The result is often complex-valued and reflects different frequency components as complex amplitudes. The FT techniques adopted in this paper are Dicrete Fourier Transform (DFT) and Short Time Fourier Transform (STFT). Analogous to function and its inverse, the operations above also have their inverse operations, which transform a frequency-domain back to its time domain. In this way, series of Fourier transforms have bridged the gap between time domain and frequency domain

Frequency (Spectral)-domain analysis is arguably the most popular techniques for processing and reduction of the noise in corrupted audio signal. Note that, the terms, frequency and spectral domain analysis will be used interchangeably. With the mapping to frequency domain from time domain, we are able to extract information which is usually not visible in time-domain plots of the signal. Spectral-domain analysis and processing has applications in various topics in audio industry such as fundamental frequency detection, additive synthesis, etc.

Unfortunately, when we convert the signal from time domain to frequency domain, we lose the time signature and obtain a rather "static screenshot" of the target signal with all the frequency components listed in the DFT plot. So when we perform an inverse operation on the transformed DFT signal, we get a completely different result. The solution in here is to perform DFT on small segments of the audio and record their sample number. Combined with the sampling rate, we can keep the time domain of the signal to some extent with a small loss, though a loss on the time information is inevitable, but the loss can be minor on the reconstructed signal.

The most common first-steps for spectral-domain analysis and processing on a lengthy audio signal are to divide them into blocks then apply DFT (FFT) to each block. Frequency-level analysis and operations are then applied to the spectrum of the signal. As the final step, the signal is reconstructed from the DFTs with inverse DFT method. Before the transformation from time to frequency domain and vice versa, a window function is normally applied to avoid spectral leakage, which could potentially result in further corruptions in the reconstructed signal. After the reconstruction, some scaling and normalizations are applied and then we have the processed signal.

## III. Algorithm and Implementation Walkthrough

In this section, we will do a walk through of the algorithm. Some formula and explanations will also be included on the way. Implementation considerations as well as decisions will be explained as well.

### A. *Step I: Obtain the Spectral Representation*

As mentioned in section II, the first step is to set the window size so that we could break the signal into multiple blocks. And for each subframe n in each block, the Discrete Fourier Transform is applied (Godsill & W., 1998) [1]:

$$Y(n,m) = \sum_{l=0}^{N-1} g_l \, y_{nM+l} exp(jlm2\pi/N)$$

Where Y is the converted signal in the frequency domain, y is the original signal, n is the subframe number within the block, m is the frequency component, $g_l$ is a window function on the time domain and N is the signal length. After this transformation, we obtain the frequency display of the signal block. And if we perform this operation on each block of time-domain signal, we can obtain a calculated power spectrogram of the signal.

Several parameters in this formula must be carefully chosen to achieve the optimal performance of the implementation. First, when selecting the window size, N, different values often give spectrograms with different properties. For example, a big N normally offers a higher resolution in frequency but a lower resolution in time in the spectrogram, and vice versa, a smaller N offers a higher resolution in time but a lower resolution in the frequency. There is no definitive answer for the best n, and users have to make the choice between the tradeoff. In addition, choosing the amount of overlap, M, between two frames is also important. According to (Godsill & W., 1998) [1], M should be around N/2 and N/4 to allow a significant overlap between two blocks of signal.

During the implementation of this step, the Matlab function, stft() is used to produce a spectrogram of the input signal. This function performs a Short Time Fourier Transform on the signal and produces a spectrogram for visualization. Different windows can also be chosen for the spectrogram. We have chosen the Hamming window for the implementation over the standard rectangular window to achieve a better results from frequency analysis. Figure 1 shows a spectrogram with a sum of three sine waves as input at frequencies 4000Hz, 8000Hz and 12000Hz, respectively with added Gaussian noise. Figure 2 shows the spectrogram of a flute sample playing C4 with added Gaussian noise. When we provide return values to the stft() function, we can obtain the power spectrum of the signal represented with a regular matrix specifying the intensity of each frequency component at different time.

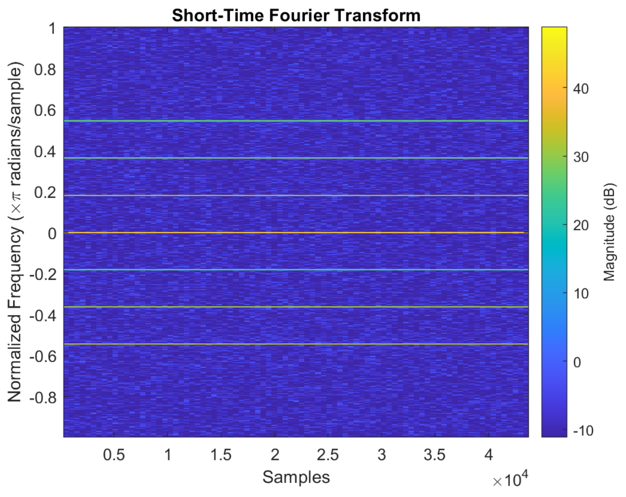

*Figure 1: Spectrogram of the sum of three sine waves at frequency of 4000Hz, 8000Hz, and 12000Hz*

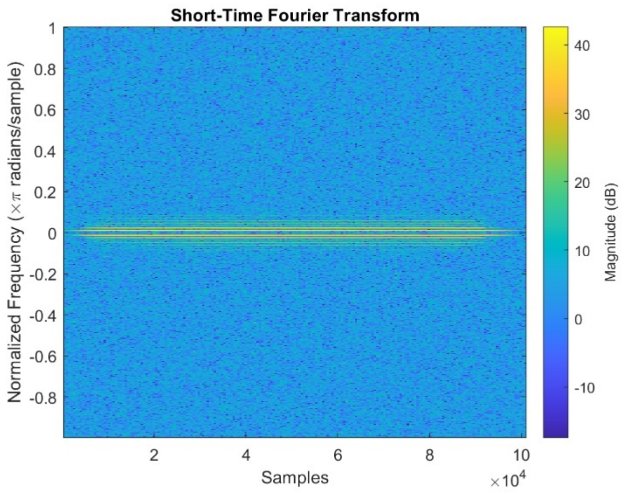

*Figure 2: Spectrogram of a recorded noisy flute sample playing C4*

### B. *Step II: Process the Spectral Components*

The processing then begins on the spectral components of each block "in order to estimate the spectrum of the clean data" (Godsill & W., 1998). In mathematical notation, we could write this process as a function $f(\cdot)$, then we have

$$X(n,m) = f\big(Y(n,m)\big)$$

Where Y(n, m) is the spectral component we calculate from the previous step.

The Wiener solution is based on the assumption that the noise is "additive and independent of the signal" (Godsill & W., 1998). And it minimizes the mean-squared error between the original signal and targeted (cleaned) signal when we reconstruct the signal from the spectrogram. The formula is given by:

$$H(w) = \frac{S_x(w)}{S_x + S_N(w)}$$

Where $S_X(w)$ is the assumed (non-existent) clean signal on the frequency domain and $S_N(w)$ is the predicted noise on the spectral level.

Note that we have information on neither of the terms mentioned in the above formula. Therefore, during the implementation, both quantities must be estimated. For the power spectrum level of the noise, we often pre-record the background noise using the same configurations while recording the noisy signal and obtain the power spectrum of the pre-recorded noise. For the power spectrum of the cleaned signal, (Godsill & W., 1998) provided an estimation using a piece-wise function:

$$S_x = \begin{cases} |Y(m)|^2 - S_N(m), & |Y(m)|^2 > S_N(m) \\ S_x = 0, & |Y(m)|^2 \leq S_N m \end{cases}$$

With all the information known above, we can then process the noisy signal and get the spectrum of the cleaned signal.

During the implementation of this step, the main obstacle was to realize that the power spectrum is the calculated magnitude from the complex-valued points on the frequency domain and is, thus, a real number. In the first implementation I did not notice this part and got incorrect results. When we transform the DFT block, we are multiplying it by a factor. And that factor should be the magnitude intuitively

Another option is using the Spectral Subtraction method, where we subtract the power spectrum of the original signal by root-mean-square noise on each frequency component. Then the noise reduction function $f(\cdot)$ becomes:

$$f(Y(m)) = \begin{cases} \dfrac{|Y(m)|^2 - S_N(m)^{1/2}}{Y(m)} Y(m), |Y(m)|^2 > S_N(m) \\ 0,\ |Y(m)|^2 > S_N(m) \end{cases}$$

This step can take extremely long if executed sequentially. Due to the nature of the small sampling period, we have a massive number of points to process. One way to resolve this is to utilize vectorization feature built-into Matlab. Matlab is optimized to work with Matrix and it can parallelize independent operations in the backstage. When testing the implementation on the sine wave, vectorization shortens the runtime by as much as 100 times. Similar features can also be found in Python's Numpy[2], where vectorization is exploited to optimize the performance of the code. However, parallelization can often hide exceptions to achieve absolute efficiency. For example, Matlab masks overflow with inf values, which can often cause problems that are hard to track down. Numpy also does not raise any exception or issue any warning when an overflow happens. Moreover, it simply returns a random number, which could be very misleading to the users. (numpy.sum)[4].

## *C. Step III: Reconstruct the Signal From Processed Spectrum*

After step II, we have obtained the spectral domain representation of the processed signal. As the final step, we reconstruct the signal with respective inverse of the operations we utilized in step I. The formula for the reconstruction is as follows:

$$w_l = \frac{1}{\sum_{m \in M_0} g_{mM+l}\, h_{mM+l}}$$

Where we apply the same set of parameters to the inverse operation.

For the implementation of this step, the function istft() in Matlab is used as this step closely resembles the inverse Short Time Fourier Transform.

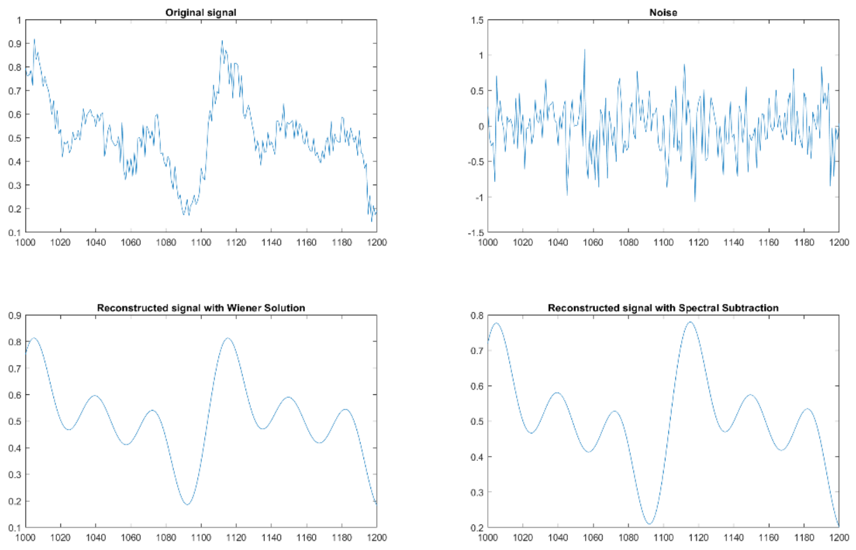

*Figure 4: Output of the implementation with input of sine waves and added Gaussian noise*

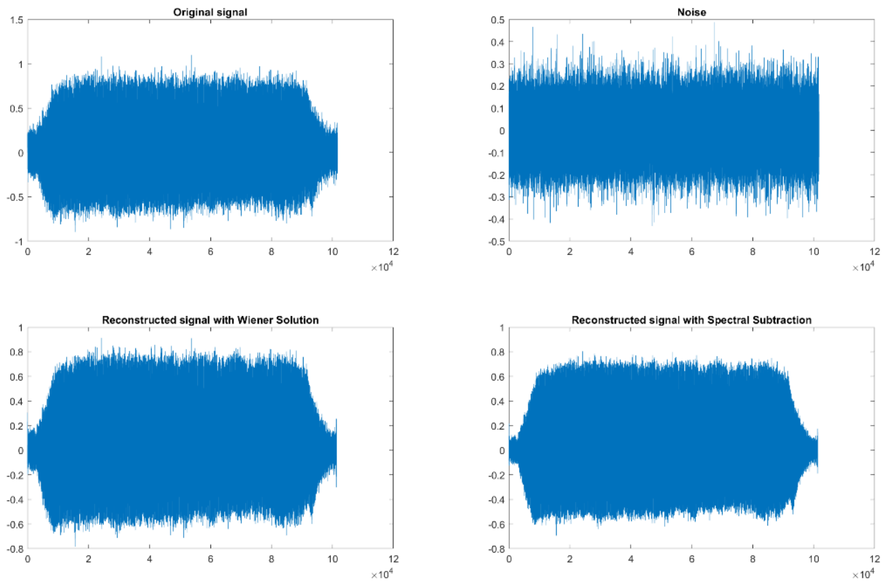

*Figure 3: Output of the implementation with input of recorded flute sample and added Gaussian Noise*

## IV. Evaluation

The following figures illustrates the output of the two respective processing techniques illustrated in the above steps with two different input signals. In Figure 3, a sum of three sine waves with frequency 400Hz, 800Hz and 1200Hz were used, a randomly generated Gaussian noise is then added to the original signal to construct the corrupted signal. Figure 4 illustrates the results when using recorded flute samples added with Gaussian noise. Figure 3 lists in more detailed view of the appearance of the signal after adding the noise while Figure 4 lists out the whole signal.

It can be seen from Figure 3 immediately that both algorithms successfully isolate a fair amount of noise from the corrupted signal. The major visible difference between the two outputs is on the level of "vertical span". Wiener solution outputs an amplitude from 0.18 to 0.82 while Spectral Subtraction gives an amplitude from 0.20 to 0.78. Both methods preserved the higher frequency of the input while isolating the noise. However, neither showed an obvious success at the isolation of the fundamental frequency from the noise, which results in some level of loss in the lower frequency components.

For the flute sample, we can perceive a significant reduction in the level of noise comparing the outputs with the original corrupted signal. However, again, some noise remained in the lower frequency components, which produced a minor noise resembling running water. Spectral Subtraction also supresses the level of the output more than the Wiener Method. Overall, for the flute sample, Spectral Subtraction produces a subtly better-restored original signal. But for the sine wave input, the audible difference between the two methods is very minor.

## V. Other Hiss Reduction Techniques

There are also many other hiss reduction techniques which approach the problem from different perspective such as probabilistic and statistical point of view. (Czyzewski) proposed an algorithm based on ML that recognizes and removes the hiss noise in the audio recording. Other model-based techniques were also proposed by various research papers that work in different scenarios.

## VI. Conclusions and Future Work

In this paper, we have gone through in detail the implementation of two of the basic hiss reduction algorithms, the Wiener Solution and Spectral Subraction as well as evaluated their performance. An optimization technique, vectorization, was also discussed to achieve a much shorter runtime. An evaluation on the output of the two algorithms was illustrated on both visual and auditory aspects.

Future work may include the elimination of the “musical noise”, which normally results from overestimations of the power spectrum of the cleaned signal. (Godsill & W., 1998) [1]. Algorithms that focus on hiss reduction in more specific scenarios may also be explored.

## References

[1] S. J. Godsill and R. P. J. W., *Digital audio restoration: a statistical model based approach*. New York, New York: Springer, 1998.

[2] F. Deng, C.-chun Bao, B.-yin Xia, and Y. Liang, “A novel hiss noise reduction method for audio signals based on MDCT,” *2011 International Conference on Wireless Communications and Signal Processing (WCSP)*, 2011.

[3] *NumPy*, n.d.. [Online]. Available: https://numpy.org/. [Accessed: 26-Apr-2021].

[4] “numpy.sum,” *numpy.sum - NumPy v1.20 Manual*. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.sum.html. [Accessed: 26-Apr-2021].

[5] Czyzewski, A. Learning Algorithms for Audio Signal Enhancement, Part 2: Implementation of the Rough-Set Method for the Removal of Hiss. Journal of the Audio Engineering Society, 45(11), 931-943. 1997.