DCSP: Implementing Background Noise Removal

Kieran Molloy

March 2020

Abstract

Removing unwanted background noise from sound signals using Discrete Fourier Transform and its inverse (DFT/IDFT) manually by varying a threshold value.

1 Introduction

The sound file being used is a .wav file, produced in previous work, and is represented in Figure.1 this is a simple filestream data format where the data is simply stored as numerical values representing the amplitude, it is often considered the simplest audio format and as such is very useful for the task at hand. A .wav file can be read and written with ease, as it was partially designed for this purpose by IBM and Microsoft *WAV standard* 2019. When the file is read, it will be converted into numbers and then it will be ready for processing. When processing has finished, it can simply be written to a new file and stored. The processing that will be performed will remove all background/unwanted noise.

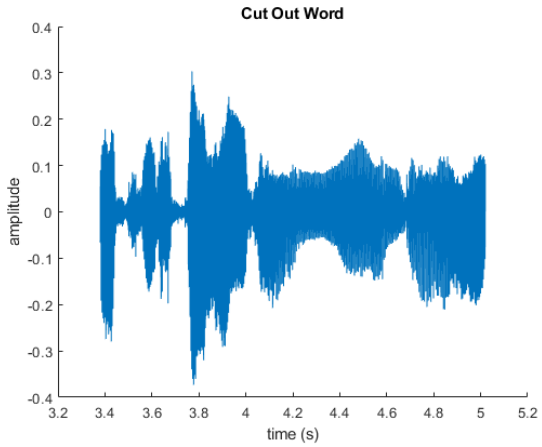


Figure 1: Speech Cutout from Original Signal

2 Method

The file that is being worked is 1.64 seconds long and consists of a woman saying "Let us go then, you and I" with a high pitch beeping noise in the background. To remove this noise we must perform a spectral analysis based method of removing amplitudes higher than a threshold. Mingham 2019

2.1 Discrete Fourier Transform

The discrete Fourier transform, or DFT, is the primary tool of digital signal processing. The foundation of the product is the fast Fourier transform (FFT), a method for computing the DFT with reduced execution time. Equation 1 gives the DFT Equation.

$$c_n \approx F_n = \frac{1}{N} \sum_{k=0}^{N-1} s(kT) e^{-2\pi i k n/N} \tag{1}$$

It can be noted that the negative values equate as shown in Eq.2

$$F_{-1} = F_{N-1}, \qquad F_{-2} = F_{N-2} \tag{2}$$

Equation 1 takes a sequence of N sample values of an analogue signal $s(t)$ and then transforms then to N complex numbers, this is called the Discrete Fourier Transform. Owens 1992

The inverse discrete fourier transform takes a sequence of N complex numbers and transforms them back into the original time sequence, as defined in Eq.3

$$s(nT) = \sum_{k=0}^{N-1} F_k e^{2\pi i k n/N} \tag{3}$$

for $n = 0, 1, \ldots, N-1$

The MATLAB environment provides the functions *fft* and *ifft* to compute the discrete Fourier transform and its inverse, respectively. These are based on a C implementation documented in *FFTW* 2020. For the input sequence x and its transformed version X (the discrete-time Fourier transform at equally spaced frequencies around the unit circle), the two functions implement the relationships defined in Eq.1 and Eq.3. (It must be noted that the MATLAB convention is to use a negative j for the *fft* function. This is an engineering convention; physics and pure mathematics typically use a positive j.)(Ingle 2017)

By viewing a signal in the frequency domain it may be possible to remove unwanted frequency components by setting them to zero then going back to the time domain (via the IDFT).

To perform these algorithms the signal must be folded around 0, the folding process is performed by Listing 1 and unfolding process is performed by Listing 2.

```
function y=fold(x,M,N)
    y=zeros(1,N);
    y(1:M)=x(M+2:N);
    y(M+1)=x(1);
    y(M+2:N)=x(2:M+1);
end
```

Listing 1: Matlab Folding Code Snippet

```
function y=unfold(x,M,N)
    y(1)=x(M+1);
    y(2:M+1)=x(M+2:N);
    y(M+2:N)=x(1:M);
end
```

Listing 2: Matlab Un-Folding Code Snippet

To remove the noise, specific values must be removed. This process is done by using a threshold value, α, due to using a .wav file we are able to use MATLAB logical expressions to find all values exceeding α and set them to 0, as shown in Listing 3.

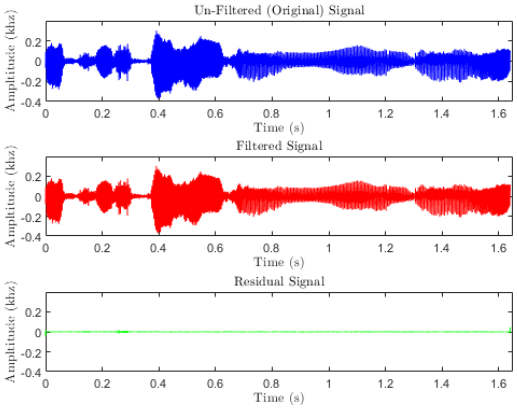
```
alpha=5000;
Index=find(abs(freq) > alpha);
indexedF=foldedF;
indexedF(Index)=0;
```

Listing 3: Matlab Threshold Removal Code Snippet

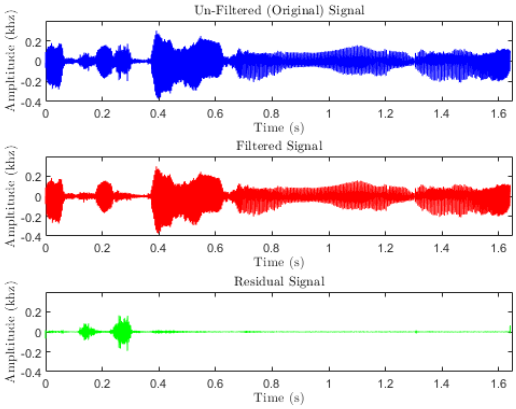
It is important to remove background noise whilst keeping as much detail as possible, to do this we visually inspect the plots produced (See Sec.3) and listen to the output file.

3 Results

The initial α value is 10000 and the results can be seen in Figure 2a, in the residual signal there is very little removed, so α is halved. $\alpha = 5000$ displays significantly more removal, however only for a small section. So α must be reduced further. Since the high pitch noise remains in $\alpha = 4000$ (shown in Figure 3a), it is again reduced to 3000, the residual signal in Figure 3b shows removal across the entire signal, however the high pitch noise can still be heard For $\alpha = 2500$, the high pitch noise is gone, however it is important to keep as much detail as possible, so a higher interval is chosen, $\alpha = 2700$, shown in Figure 4 The final threshold value is 2663, as $\alpha = 2664$ keeps the high pitch noise we know this is the clearest

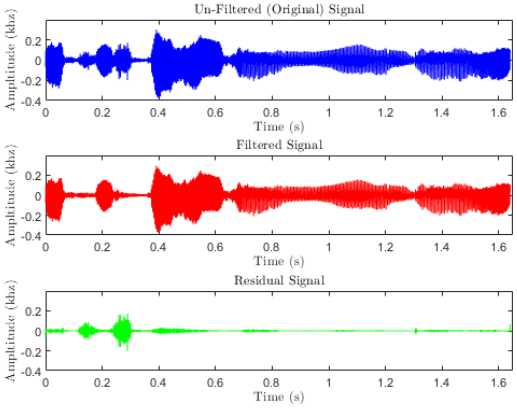


(a) α =10000 Signal

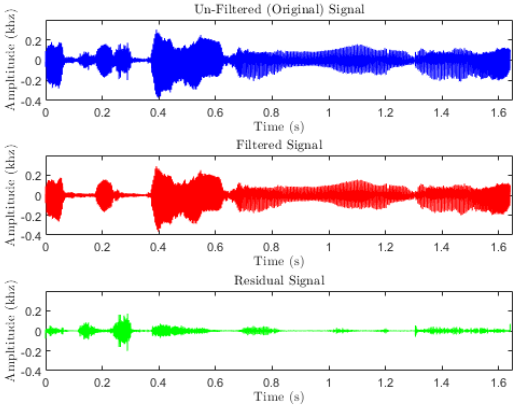


(b) α =5000 Signal

Figure 2: Initial α Values



(a) α =4000 Signal



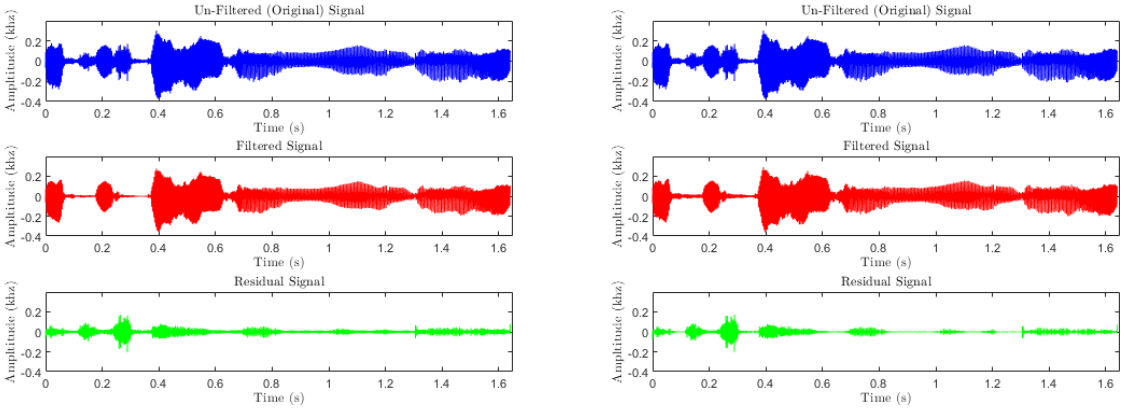
(b) α =3000 Signal

Figure 3: Secondary α Values

threshold value whilst still removing any background noise. Shown in figure 5 Figure 5b demonstrates the folding and unfolding process of the audio clip, from top to bottom. The middle subplots show the threshold value being applied, the solitary spike is at $x = 2664$, hence why 2663 is the optimum threshold value.

4 Conclusions

Removing unwanted noise can be a trial and error process, whereby you must remove enough frequencies to remove the noise, whilst retaining enough frequencies to accurately represent the signal. Although a threshold is used here, each individual frequency could be selected and removed or kept, however this would be incredibly hard and time consuming. But would probably produce a better representation, with only the noise being removed. The final output signal produced sounds dull, this is due to higher



(a) α =2500 Signal

(b) α =2700 Signal

Figure 4: Tertiary α Values

frequencies being removed with the noise.

References

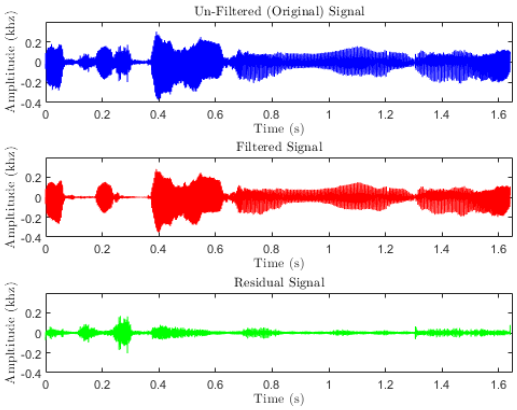
FFTW (2020). FFTW. URL: `http://www.fftw.org/` (visited on 03/02/2020).

Ingle, John G. Proakis; Vinay K. (2017). *Digital signal processing using MATLAB: a problem solving companion*. Cengage Learning.

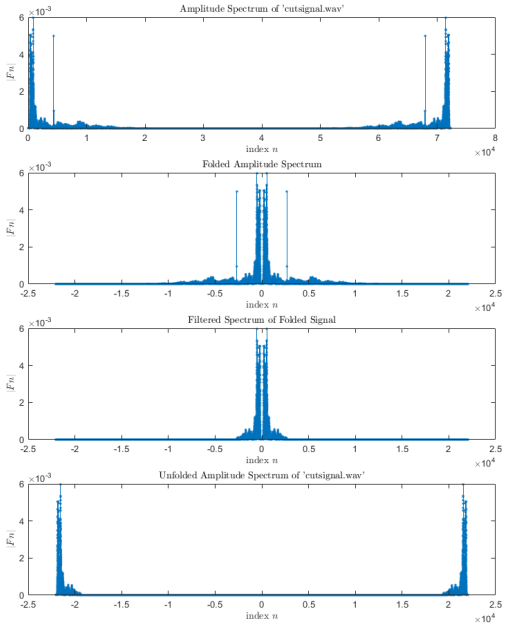
Mingham, Clive (2019). *Lecture Notes and Exercises for DIGITAL COMMUNICATIONS & SOUND PROCESSING*. MMU.

Owens, Frank J (1992). *Signal processing of speech*. Macmillan P.

WAV standard (2019). Sustainability of Digital Formats: Planning for Library of Congress Collections. URL: `https://www.loc.gov/preservation/digital/formats/fdd/fdd000001.shtml` (visited on 07/11/2019).



(a) α =2663 Signal



(b) α =2663 Spectrum

Figure 5: Final α Value

A Full Matlab Code

```
% Kieran Molloy Submission for MMU Year 3: Digital Communications & Sound
% Processing Coursework Part 2

clear; clc; clf
% Read File
readFilename='cutsignal.wav';
[s,fs]=audioread(readFilename);

% Number of Samples
N=length(s);
% Sampling Interval
T=1/fs;
% Length of Signal
P=N*T;
% T vector
t=0:T:P-T;
% f0 Value
f0=1/P;

% Discrete Fourier Transform
F=fft(s)/N;

% Folding Logic
M=(N-1)/2;
foldedF=fold(F,M,N);
% Outline x-axis domain
freq=-M*f0:f0:M*f0;
% Remove Frequences higher than abs(alpha)hz
alpha=5000;
Index=find(abs(freq) > alpha);
indexedF=foldedF;
indexedF(Index)=0;


% Return to time domain but reorder F before taking its IDFT.
newF=unfold(indexedF,M,N);

% Inverse Discrete Fourier Transform
sfilt=N*real(ifft(newF));

% Difference between original \& filtered
sdiff = s'-sfilt;

```

```
% Write to file
writeFilename = 'filteredSignal.wav';
audiowrite(writeFilename,sfilt,fs);

% //////////
% PLOTTING
% //////////

% Figure 1 - Spectral Analysis Comparisons
figure(1)
% Subplot 1 - Original Spectral Analysis
subplot(4,1,1),stem(abs(F),'.-');   % plot the absolute value against index number
title("Amplitude Spectrum of 'cutsignal.wav'",'interpreter','latex');
xlabel('index $n$','interpreter','latex');
ylabel('$\left|Fn\right|$','interpreter','latex');
% Subplot 2 -  Folded Aamplitude Spectrum
subplot(4,1,2),stem(freq,abs(foldedF),'.-');
title("Folded Amplitude Spectrum",'interpreter','latex');
xlabel('index $n$','interpreter','latex');
ylabel('$\left|Fn\right|$','interpreter','latex');
% Subplot 3 - Filtered + Folded Amplitude Spectrum
subplot(4,1,3),stem(freq,abs(indexedF),'.-');
title("Filtered Spectrum of Folded Signal",'interpreter','latex');
xlabel('index $n$','interpreter','latex');
ylabel('$\left|Fn\right|$','interpreter','latex');
% Subplot 4 - Filtered + UnFolded Amplitude Spectrum
subplot(4,1,4),stem(freq,abs(newF),'.-');
title("Unfolded Amplitude Spectrum of 'cutsignal.wav'",'interpreter','latex');
xlabel('index $n$','interpreter','latex');
ylabel('$\left|Fn\right|$','interpreter','latex');

% Figure 2 - Full Signal Comparison
figure(2)
% Subplot 1 - Original cutsignal.wav
subplot(3,1,1),plot(t,s,'b')
axis([0 1.65 -0.4 0.4])
title('Un-Filtered (Original) Signal','interpreter','latex')
xlabel('Time (s)','interpreter','latex')
ylabel('Ampltitude (khz)','interpreter','latex')

% Subplot 2 - Filtered cutsignal.wav
subplot(3,1,2),plot(t,sfilt,'r');
axis([0 1.65 -0.4 0.4])
title('Filtered Signal','interpreter','latex')
```

```
88 xlabel('Time (s)','interpreter','latex')
89 ylabel('Ampltitude (khz)','interpreter','latex')
90
91 % Subplot 3 - Filtered cutsignal.wav
92 subplot(3,1,3),plot(t,sdiff,'g');
93 axis([0 1.65 -0.4 0.4])
94 title('Residual Signal','interpreter','latex')
95 xlabel('Time (s)','interpreter','latex')
96 ylabel('Ampltitude (khz)','interpreter','latex')
97
98 % ////////////////////
99 % AUXILLARY FUNCTIONS
100 % ////////////////////
101
102 function y=fold(x,M,N)
103     y=zeros(1,N);
104     y(1:M)=x(M+2:N);
105     y(M+1)=x(1);
106     y(M+2:N)=x(2:M+1);
107 end
108
109 function y=unfold(x,M,N)
110     y(1)=x(M+1);
111     y(2:M+1)=x(M+2:N);
112     y(M+2:N)=x(1:M);
113 end
```