Faculty of Engineering
Department of Mechanical & Mechatronics Engineering

TIME DOMAIN ANALYSIS

PROJECT 1

SYDE - 252 Systems and Signals
Professor: John Zelek
Prepared by:

Teodor Mihai Tuica

Pavel Shering

October 23, 2015

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

The purpose of this report is to document the analysis and filtration noise in different signals to extract useful information. During the process, three signals were analysed and noise was defined in each. This definition can be found in section 2 of the report. To minimize noise in each signal, three different low pass filters were designed: an average filter, a weighted average filter, and a median filter. Although these filters were designed using MATLAB, they can each be modelled using difference equations and impulse response functions, except the median filter. In depth breakdowns of each filter can be found in section 2.2 of this report. To extract important information from each signal, the filters were supplemented with a peak riding function. A breakdown of the peak riding function and the methods used to extract information from each signal can be found in section 3 of this report. Section 5 and 6 will follow with conclusions and recommendations, respectively.

2.0 TASK ONE – APPLICATION OF LOW PASS FILTER

This section outlines the design of low pass filters and their application onto each of the input signals. In order to determine which window size works the best for each signal an iterative experimental approach was used. This approach involves choosing various values for the parameters to pass into the filter function and documenting each step in a table. The table includes the feedback in terms of the audibility of the signal compared to the original signal for the first line and an indication of how it compares to the previous combination trial. This will ultimately determine which filter with which parameters works the best for each signal.

The different filter function types created in MATLAB are modular, meaning each filter is structured for input parameters such as sampled data, window size, and sampling rate. This allows for the filters to be universal for any audio input, thus only the window size and sampling rate will be specific to the three audio signals (ClayColoredRobin.wav, tapestry.wav and drumloop1.wav)

2.1 Noise Analysis

In general noise is defined as unwanted and unknown modifications applied to the signal during capture, transmission or processing [1]. The noise in signals can be described mathematically, using Equation 1 for continuous time domains and Equation 2 for discrete time domains.

$$x(t)_{signal} = x(t)_{filtered} + x(t)_{noise}$$
$$x(t)_{noise} = x(t)_{signal} - x(t)_{filtered}$$

Equation 1 - Noise for continuous signals

$$x(n)_{signal} = x(n)_{filtered} + x(n)_{noise}$$
$$x(n)_{noise} = x(n)_{signal} - x(n)_{filtered}$$

Equation 2 - Noise for discrete signals

Overall, any noise can be approximated as sine functions of various periods and amplitude [3]. However by analyzing each signal individually, the specific type of noise is identified.

2.1.1 ClayColoredRobin.wav

The noise present in this sound clip can be defined as pink noise. This should not be a surprise as pink noise is ubiquitous and occurs in many physical, biological and economic systems [2]. In this sample the signal of interest is the Clay Colored Robin chirping sound, however listening closely it is possible to hear the constant surrounding environmental noise of other birds, wind against the trees, animal and insects' communication which can be seen on Figure 1 as the constant amplitude middle section around the time axis.

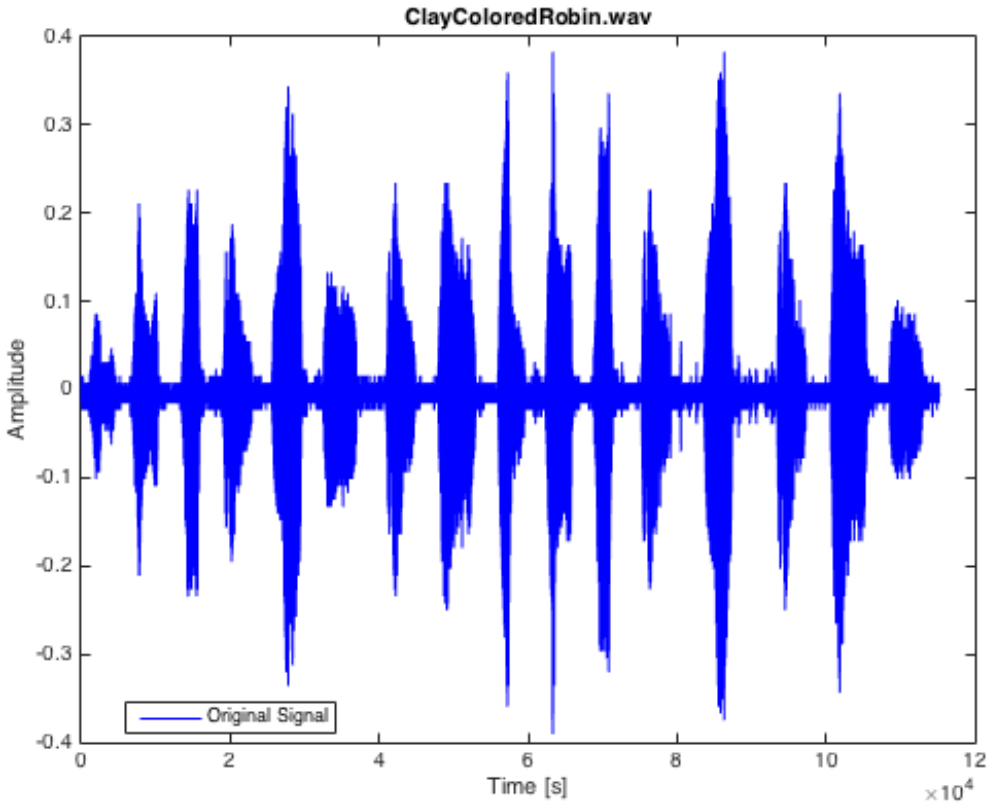

Figure 1 - Original time domain plot of ClayColoredRobin signal outlining pink noise

Modelling pink noise of this audio sample can be achieved through small period sine waves of equal energy, thus creating the environmental ambient noise that can be heard by a human ear with combination of the signal. The reason that the noise is of equal energy is because the noise stays at constant amplitude through out the sample. Due to its small period sine waves, the noise fluctuates rapidly over time and can therefore be attenuated by a low pass filter.

2.1.2 tapestry.wav

The noise present in this sample is not as easily identified since the overall signal capture is of low quality. Figure 2 it shows random noise as spikes in amplitude at ~1.5sec and ~3.3sec. This type of noise can be described as shot noise, which occurs due to random electron behavior in which electrons contributing to current are largely suppressed due to charge build up [4]. The occurrence in Figure 2 can also be described as transient noise which is defined by a short pulse followed by decaying oscillations [5]. The initial spike is caused by impulse interference and the following oscillations are due to resonance on the channel of incoming signal.

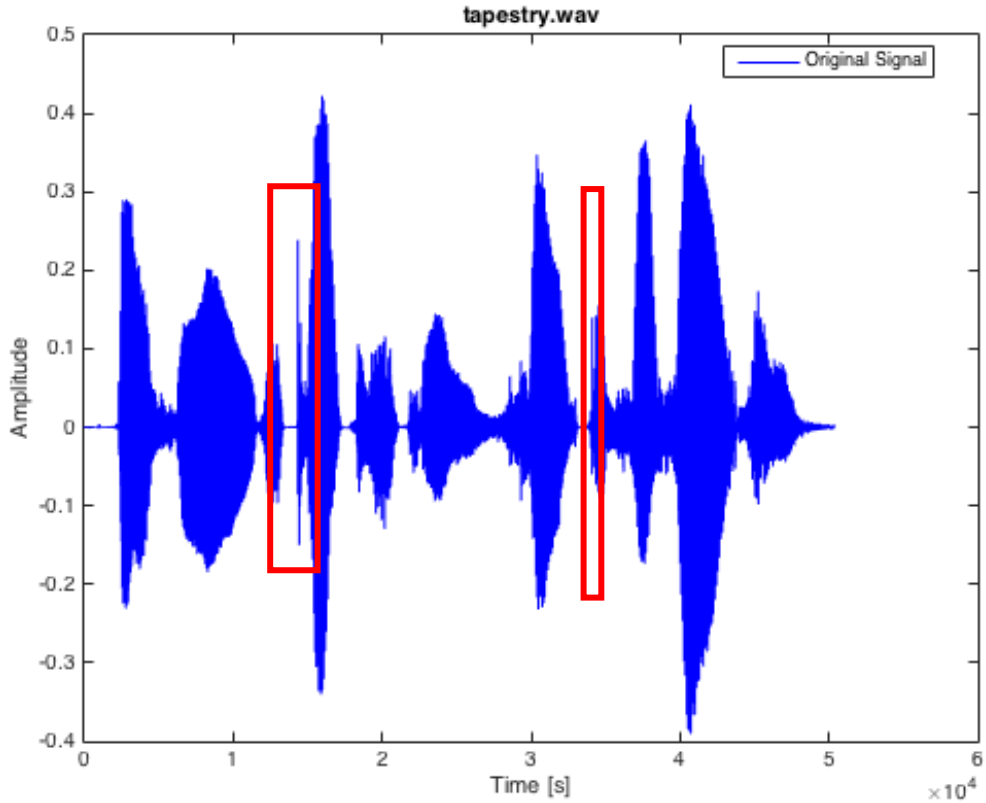

Figure 2 - Original time domain plot of tapestry.wav signal outlining shot and transient noise

To model this type of noise, it should be a short impulse of a sine wave with very short period that occurs randomly during signal capture.

2.1.3 drumloop1.wav

The noise in this sample is nearly non-existent. The sample almost sounds like an artificial synthesis of a drum loop. However, since percussion instruments are present in the sample the is minimal amount of white noise occurring in the system. The snare and the base drums are typically recreated using white noise for audio synthesis [3], which can be seen in Figure 3 where the beginning of the drum loop starts with a base drum, and the occasional snare drums have fluctuating amplitude.

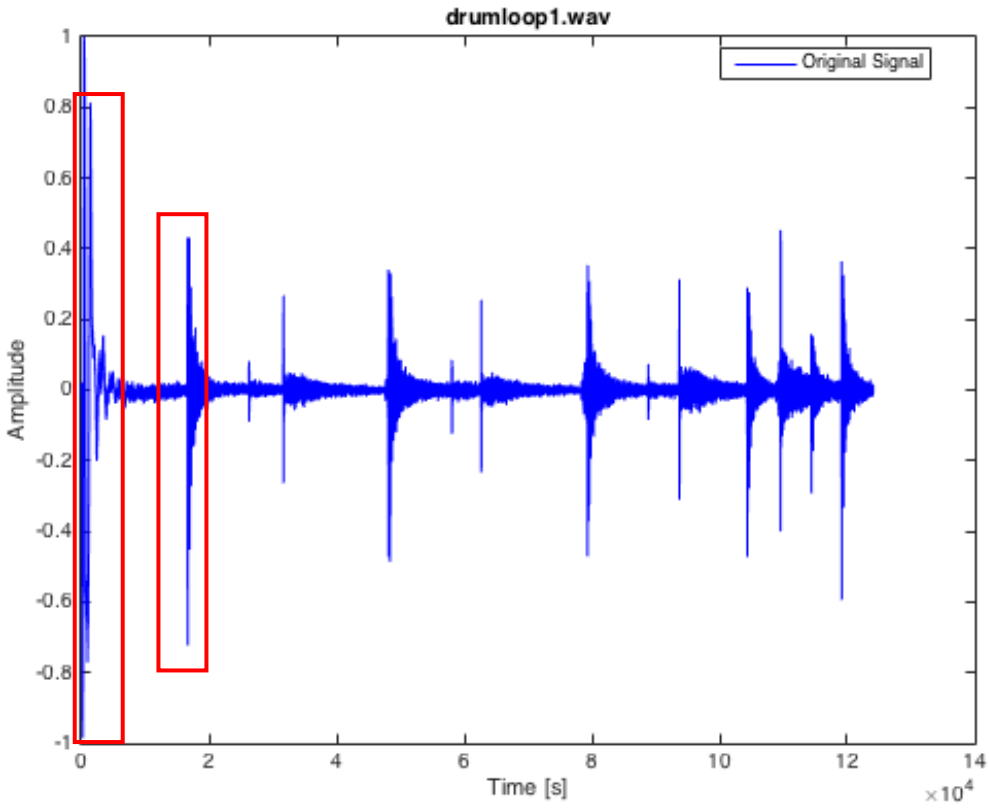

Figure 3 - Original time domain plot of drumloop1.wav displaying white noise

To model this type of noise, it also should be short period of sine wave which creates high signal distortion for very small period of time.

2.2 Low Pass Filters

Noise reduction, or in other words the recovery of the original signal is the design objective for signal processing which can be done with the use of a filter. Each filter used in MATLAB is represented mathematically as impulse response functions, and further as difference and differential equations.

2.2.1 Averaging Filter

The averaging filter smoothens out dramatic changes in the amplitude by outputting an average of surrounding surrounding signal values. It takes an average of an odd number of values, centered on the current index value. The function is defined as follows:

```
avgFilter(signal, sample_rate, window)
```

Parameters:

1) signal – the signal from the sound clip
2) sample_rate – the sample rate used to get the signal from the sound clip

3) window – range of values to average, centered on the current value. This should be an odd number since we are dealing in the discrete domain.

This function returns a vector representing a signal.
The difference function for this signal can be represented in Equation 3:

$$y[n] = \frac{1}{window}\left(\sum_{i=-(window-1)/2}^{(window-1)/2} x[n-i]\right)$$

Equation 3 - Average difference function

The summation sums all of the values within the inputted window. Since the summation begins at –(window-1)/2 and goes to (window-1)/2, the center value will be 0. Since the expression within the summation is x[n-i], this means it is centered on x[n]. The sum is then divided by the number of values in the window to obtain an average signal.

The differential equation for this signal can be represented in Equation 4:

$$y(t) = \frac{1}{window}\int_{-window/2}^{window/2} x(t-\tau)d\tau$$

Equation 4 - Average differential equation

The rationale for this equation is very similar to the difference equation above with two minor changes. The summation of values over an interval in the continuous domain is performed through an integral. The window no longer has to be an integer value the limits of the integral change to –(window/2) and (window/2).

The impulse response functions in both continuous and discrete domains are defined by modifying the differential and difference equations. In the difference/differential equations, the input, x, becomes the unit impulse function δ. The output y, becomes the impulse response, h.

Continuous:

$$h(t) = \frac{1}{window}\int_{-window/2}^{window/2} \delta(t-\tau)d\tau$$

Equation 5 - Average filter impulse response function for continious domain

Discrete:

$$h[n] = \frac{1}{window}\left(\sum_{i=-(window-1)/2}^{(window-1)/2} \delta[n-i]\right)$$

Equation 6 - Average filter impulse response function for discrete domain

2.2.2 Gaussian Filter

Gaussian function known as the bell shaped curve or normal distribution function is represented mathematically in Equation 3, which is centered about the y axis.

$$G(t) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(\frac{-(t-\mu)^2}{2\sigma^2}\right)$$

Equation 7 - Gaussian function

where σ is the standard deviation or window, μ is the expected value of normal distribution curve, in this case 0.

The Gaussian filter smoothens out dramatic changes in the amplitude by outputting a weighted average of surrounding signal values, where the weighting is based on the normal distribution curve values. The function is defined as follows:

```
gausswin(N, alpha)
```

Parameters:

1) `N` - number of points from the normal distribution curve which defines the window size
2) `alpha` – defines the weighting factor for the window size

In other words, alpha controls the width of the normal distribution curve. The higher the alpha value the narrower the width of the Gaussian Function, thus alpha is changing the weighting priority to the center of the window in an exponential effect. In our application, a higher value of alpha will put a larger emphasis on the closest signal values. The function that is called to complete the entire filtering processed with Gaussian function is shown below:

```
gaussianFilter(alpha, window, signal)
```

Parameters:

1) `alpha` - defines the weighting factor for the window size
2) `window` - number of points from the normal distribution curve which defines the window size
3) signal - the audio sample

This function returns a vector representing a signal.
The difference function for this signal can be represented in Equation 8:

$$y[n] = \frac{\sum_{-(window-1)/2}^{(window-1)/2} G[\tau] * x[n-\tau]d\tau}{\sum_{-(window-1)/2}^{(window-1)/2} G[\tau]d\tau}$$

Equation 8 - Gaussian difference function

The summation sums all of the values within the inputted window. Since the summation begins at –(window-1)/2 and goes to (window-1)/2, the center value will be 0 due to the nature of the normal distribution function that is centered at 0. The x(n-τ) is convoluted

with the Gaussian function inside the summation in order to assign weighted values for each index from n–(window-1)/2 to n+(window-1)/2 centered on x(n). Then the sum of the convolution is divided by the sum of the Gaussian function to normalize the result in order to achieve magnitude of range -1< y < 1.

The differential equation for this signal can be represented in Equation 9:

$$y(t) = \frac{\int_{-window/2}^{window/2} G(\tau) * x(t-\tau)d\tau}{\int_{-window/2}^{window/2} G(\tau)d\tau}$$

Equation 9 - Gaussian differential function

The rationale for this equation is very similar to the difference Equation 8, with a couple of minor changes. First, the summation of values over a continuous domain interval is accomplished using an integral. Second, the upper and lower limits have changed since they can be fractions in the continuous domain.

The impulse response functions in both continuous and discrete domains are defined by modifying the differential and difference equations. From the difference/differential equations, the input, x, becomes the unit impulse function δ. The output y, becomes the impulse response, h.

Continuous:

$$h(t) = \frac{\int_{-window/2}^{window/2} G(\tau) * \delta(t-\tau)d\tau}{\int_{-window/2}^{window/2} G(\tau)d\tau}$$

Equation 10 - Gaussian filter impulse response function for continious domain

Discrete:

$$h[n] = \frac{\sum_{-(window-1)/2}^{(window-1)/2} G[\tau] * \delta[n-\tau]d\tau}{\sum_{-(window-1)/2}^{(window-1)/2} G[\tau]d\tau}$$

Equation 11 - Gaussian filter impulse response function for discrete domain

2.2.3 Median Filter

Median filter cannot be represented mathematically because the process of the median filter is to take all the values in the window, then sort those value in ascending order to determine the median of the list and set that as your current y(t). Although the sorting algorithm can be done in MATLAB code, it requires logical evaluation of Boolean operations, such as greater than and less than, which cannot be represented in a mathematical formula since it requires a loop for sorting the list.

The function in MATLAB uses the built in median to find the center value of the sorted window.

```
medianFilter(signal, window)
```

Parameters:

1) signal – the signal from the sound clip
2) window – range of values to sort and determine a median, centered on the current value. This should be an odd number since we are dealing in the discrete domain.

The median filter cannot be represented as difference/differential equations for the reason stated above. In addition, the input response function cannot be represented as well for both the continuous and discrete domains.

2.3 Application of Low Pass Filters

This section demonstrates the low pass filters applied onto each of the input signals. In order to determine which window size works the best for each signal an iterative experimental approach was used which involves choosing various values for the parameters to pass into the filter function and documenting each step in a table. The table involves the feed back in terms of the audibility of the signal compared to the original signal for the first line and better or worse to the previous combination trial. This will ultimately determine which filter works the best for each signal.

2.3.1 ClayColoredRobin.wav

The following parameter values are used for the average filter:

avgFilter (… , window = 10)

Table 1- Average filter trials for ClayColoredRobin signal

noise present?	**Yes**	**Yes worse**	**Yes worse**	**Yes worse**	**Yes worse**	**Yes better**	**Yes better**	**Yes worse**	**Yes better**
window	38	30	16	20	68	26	**10**	4	6

The experiment revealed that at any window size the background noise of other animals and insects was still present. Figure 4 demonstrates the averaged signal of the input. However, there is very little difference audibly between filtered and unfiltered signal with use of the averaging filter, however window size of 10 sounded slightly better than the listed trials.

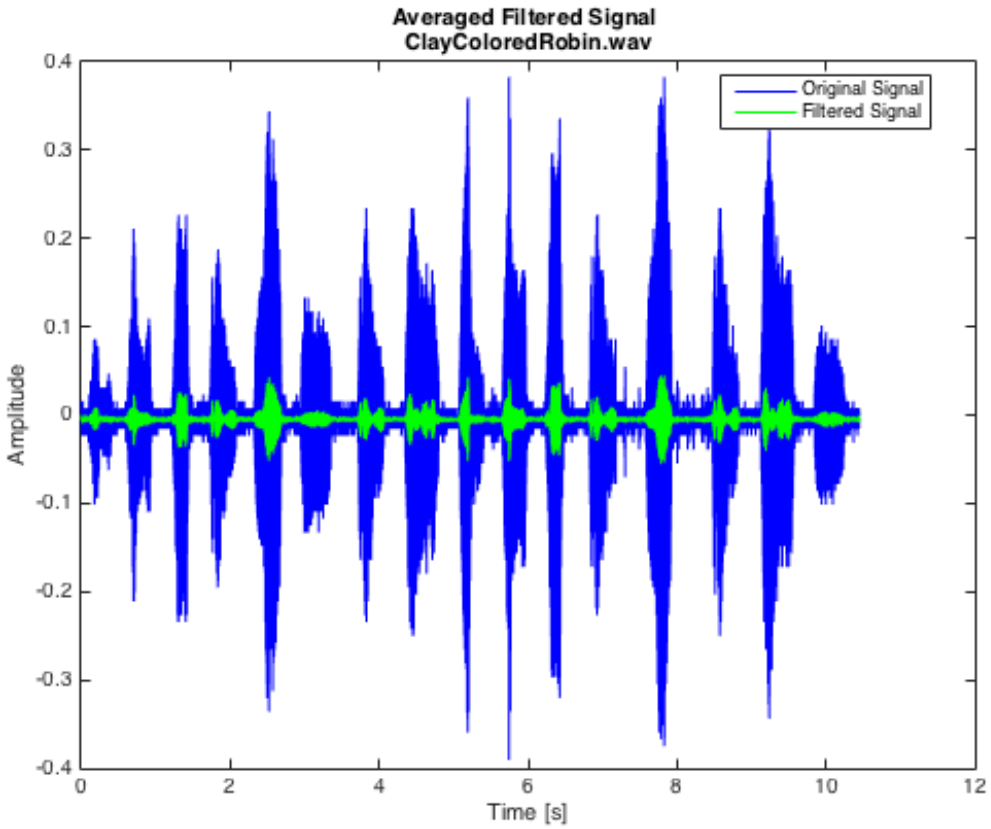

Figure 4 - Average filter effect on the ClayColoredRobin.wav audio file

The following parameter values are used for the Gaussian filter:

gaussianFilter (… ,
N = 25,
alpha = 7)

Table 2- Gaussian filter trials for ClayColoredRobin signal

noise present?	**Yes**	**Yes better**	**Yes worse**	**Yes worse**	**Yes worse**	**Pink noise better**	**Pink noise better**	**Yes worse**	**Less signal**
N	5	10	10	10	3	30	**25**	25	25
alpha	2	4	5	7	7	7	**7**	9	5

Through experimental approach the combination for size 25 window and 7 as the width of the normal distribution function, resulted in a signal without other animal or insect sounds however the pink noise remained in the signal with any combination of the above. The filter made the signal more defined but is not able to remove the short period pink noise waves out of the system. Figure 5 shows the original signal with the filtered signal to demonstrate the filtering of animal and insect noises in the background of the whistling Robin.

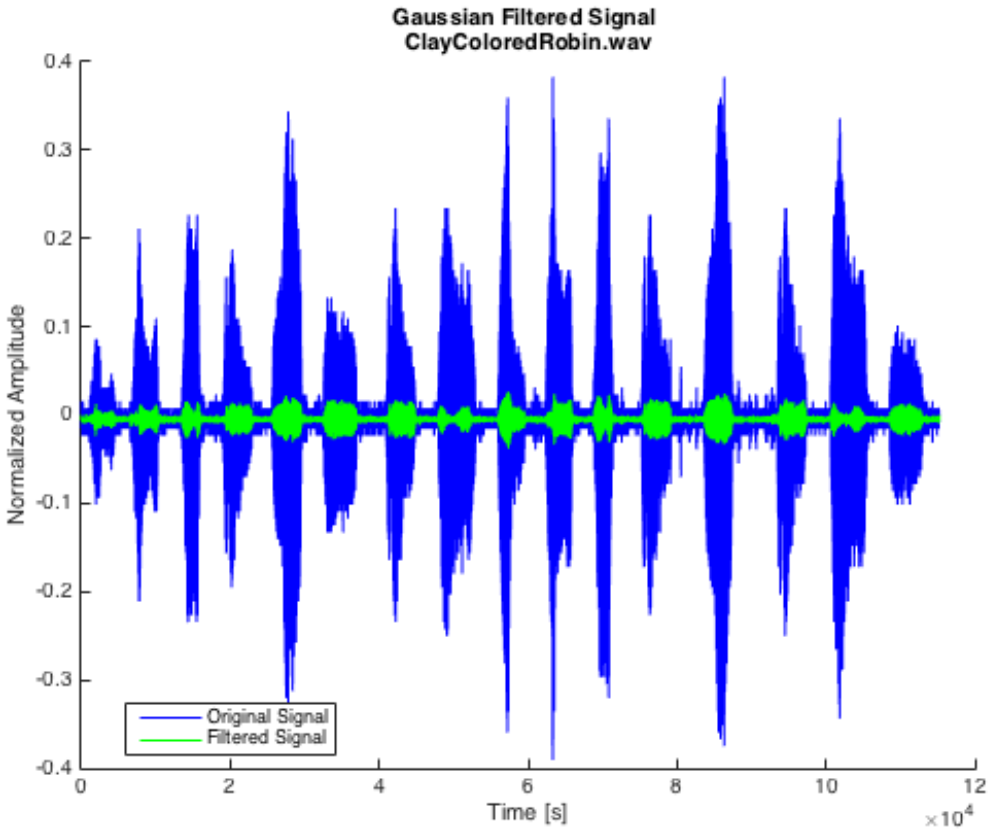

Figure 5 - Gaussian filter effect on the ClayColoredRobin.wav audio file

The following parameter values are used for the median filter:

medianFilter (… , window = 13)

Table 3 - Median filter trials for ClayColoredRobin signal

noise present?	New noise	New noise worse	New noise worse	Static worse	Static worse	Yes better	Yes better	**Static better**	Yes worse
window	11	5	3	15	17	9	7	13	1

Through experimental approach the combination for size 13 resulted in the best signal out of all the trials, however, the filter created static noise and introduced greater interference with the signal creating an water drop sound, which can be seen on Figure 6. Not applying the filter to the system results in better signal than with the filter.

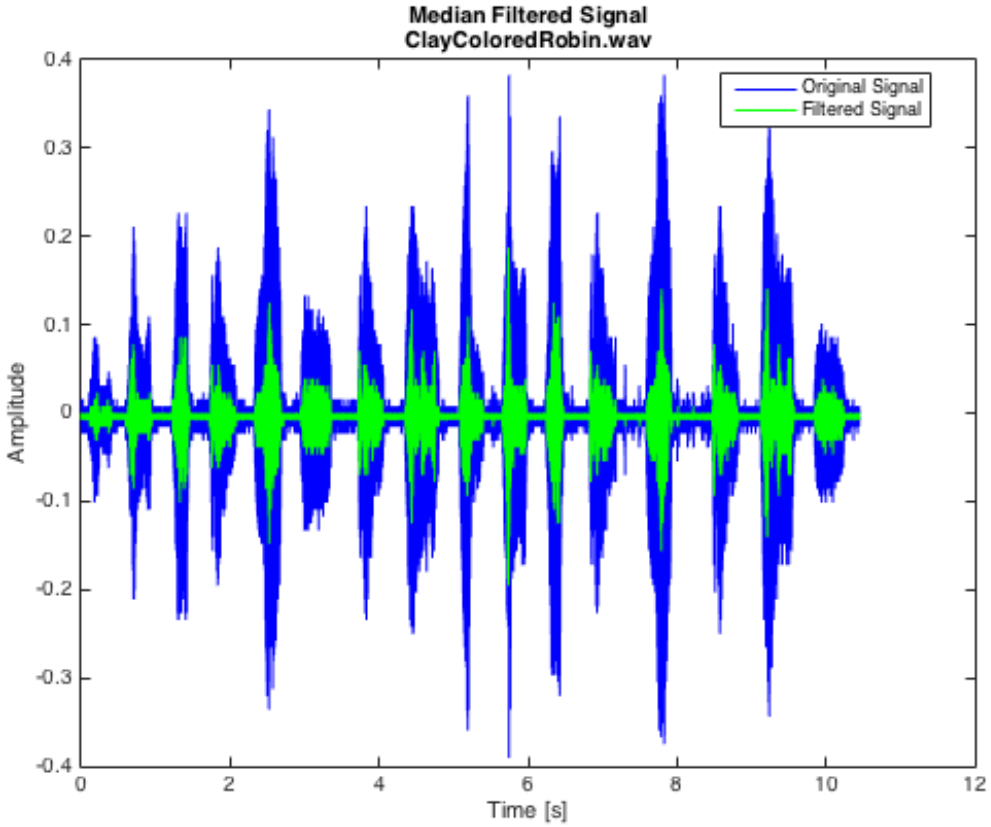

Figure 6 - Median filter effect on the ClayColoredRobin.wav audio file

2.3.2 tapestry.wav

The following parameter values are used for the average filter:

avgFilter (… , window = 2)

Table 4 - Average filter trials for tapestry signal

noise present?	Yes	No better	No worse	No better	No worse	No better	No better	No better	No better
window	38	30	16	20	68	26	**10**	4	2

The experiment revealed that window size of 2 does the least damage to the signal, thus the filtered signal sounds worse than the unfiltered, as the filter is cutting out important signal waves that complete the woman's speech. Figure 7 demonstrates the averaged signal of the input. Window size of 2 is chosen as it does minimal damage to the signal, meaning deletes the least amount of useful signal compared to the other trials.

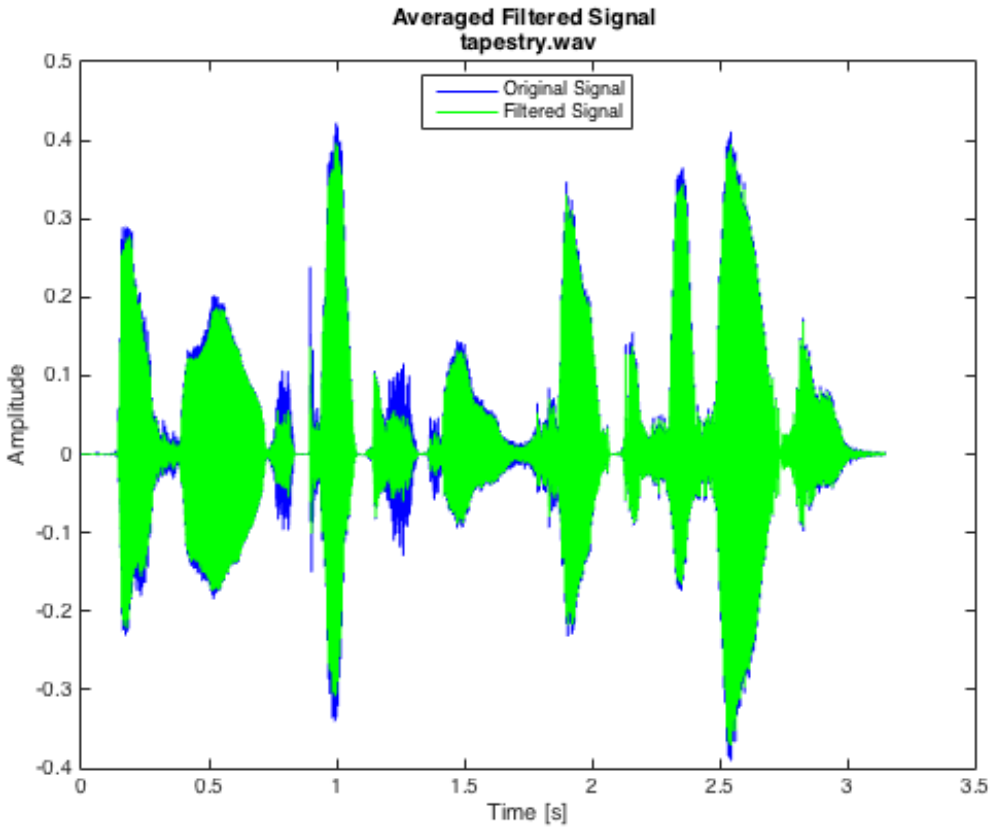

Figure 7- Average filter effect on the tapestry.wav audio file

The following parameter values are used for the Gaussian filter:

gaussianFilter (… ,
N = 25,
alpha = 25)

Table 5 - Gaussian filter trials for tapestry signal

noise present?	**No**	**No quieter**	**No better**	**No better**	**No filter**	**No quieter**	**No better**	**No better**	**No better**
N	5	10	10	10	3	30	**25**	25	25
alpha	2	4	5	7	7	7	**7**	9	25

Through iterative approach the combination for size 25 window and 25 as the width of the normal distribution function, resulted in a signal that is very similar to the unfiltered tapestry.wav sample with minimal loss of the significant signal waves, which is determined by clarity of sound. Figure 8 shows the applied Gaussian filter, which notice it very similar to the average filter result.

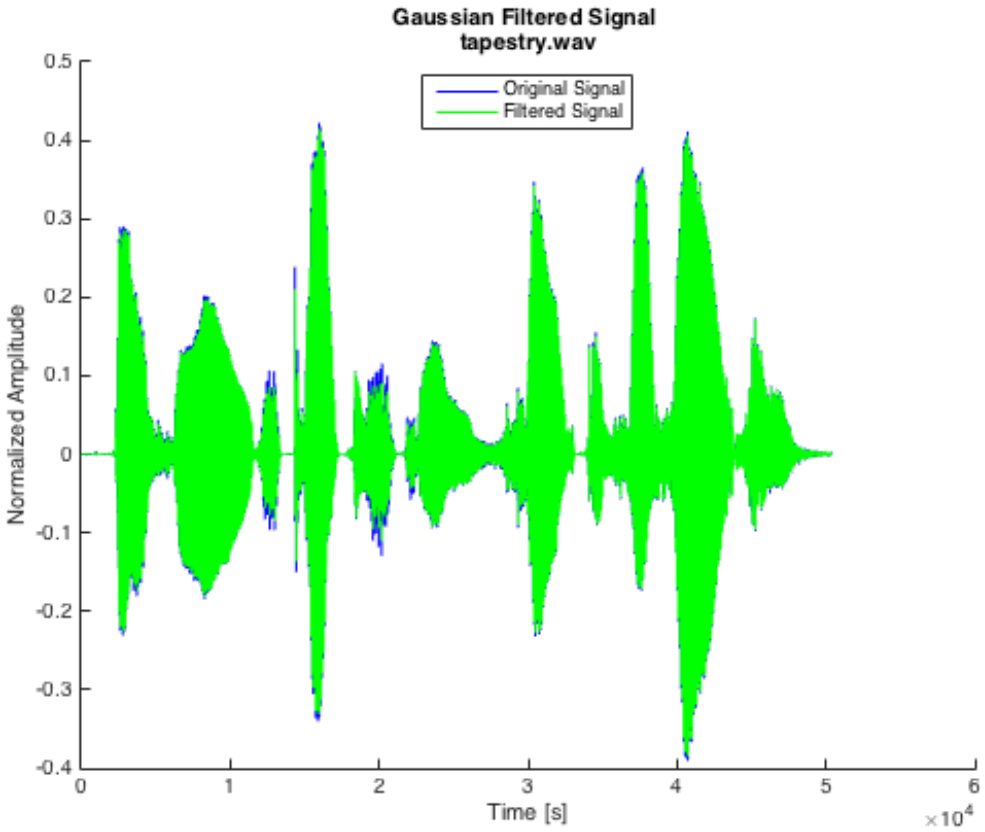

Figure 8 - Gaussian filter effect on the tapestry.wav audio file

The following parameter values are used for the median filter:

medianFilter (… , window = 1)

Table 6 - Median filter trials for tapestry signal

noise present?	**static**	**Static better**	**Static better**	**Static worse**	**Static worse**	**Static better**	**Static better**	**Static worse**	**None better**
window	11	5	3	15	17	9	7	13	1

Through experimental approach the combination for size 1 resulted in the clearest audio compared to the trials performed, however, the filter created static noise and introduced greater interference with the signal, which can be seen on Figure 9. Due to the parameter set for the filter input, the unfiltered signal and filtered sounds almost the exact same with close to no loss in the important signal waves.

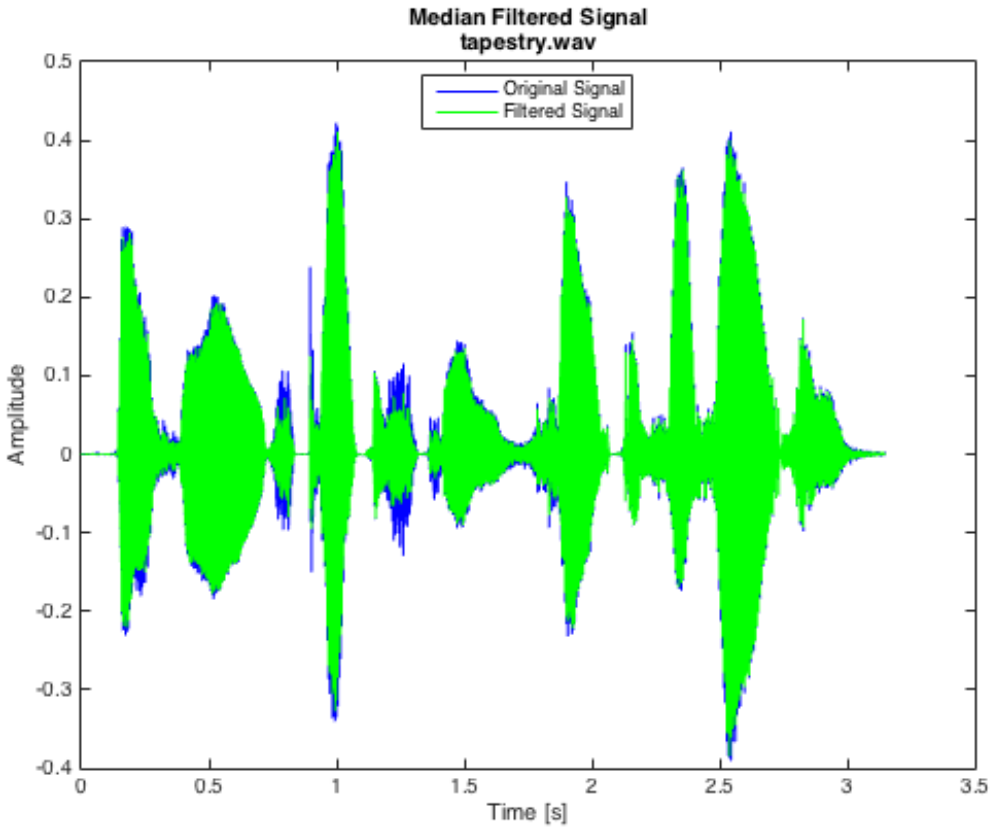

Figure 9 - Median filter effect on the tapestry.wav audio file

2.3.3 drumloop1.wav

The following parameter values are used for the average filter:

avgFilter (… , window = 2)

Table 7 - Average filter trials for drumloop1 signal

Noise?/ signal damage?	Less signal	Less signal better	Less signal better	Less signal worse	~ 0 signal worse	Less signal better	Less signal better	Less signal better	Less signal better
window	38	30	16	20	68	14	**10**	4	2

The experiment revealed that window size of 2 sounds the best for this average filtered signal. Figure 10 demonstrates the averaged signal of the input. Window size of 2 is chosen as it does minimal damage to the signal, since the original signal contains almost no noise which in means that the filter deletes useful signal waves. However, window size of 2 retains most of the original audio thus sounding better than the trails in Table 7.

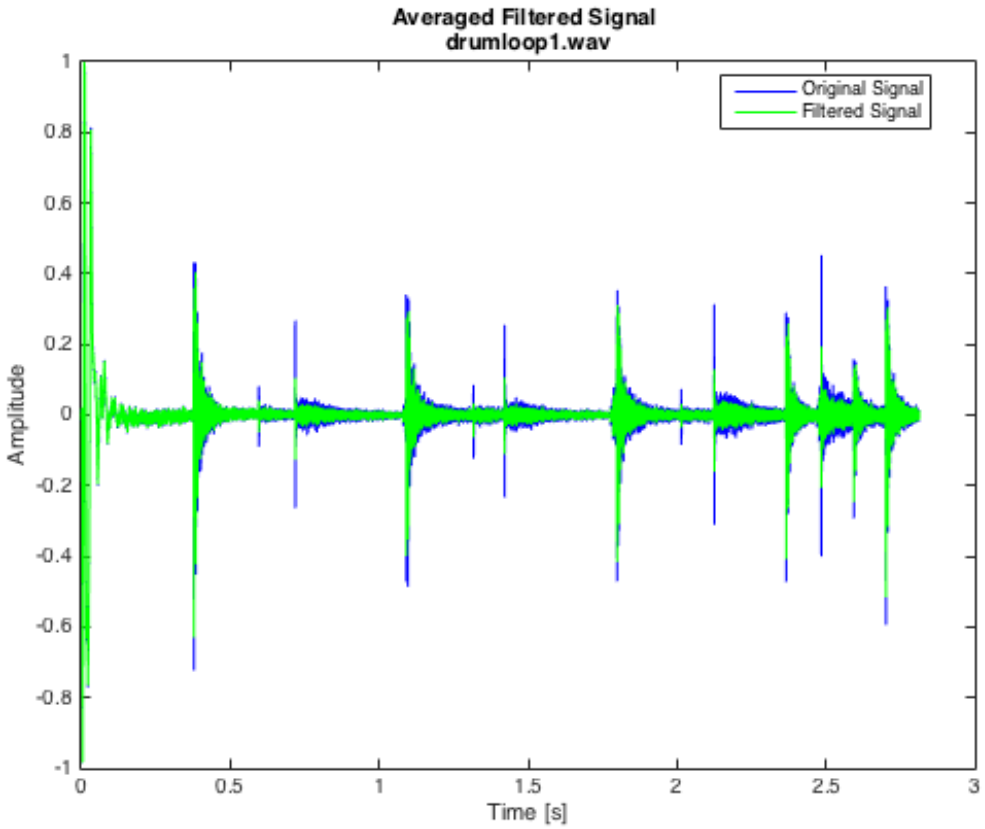

Figure 10 - Average filter effect on the drumloop1.wav audio file

The following parameter values are used for the Gaussian filter:

gaussianFilter (… ,
N = 10,
alpha = 7)

Table 8 - Gaussian filter trials for drumloop1 signal

noise present?	**No**	**No same**	**No better**	**No better**	**No filter**	**Yes worse**	**No better**	**No better**	**No better**
N	5	10	10	10	3	30	**25**	25	25
alpha	2	4	5	7	7	7	**7**	9	25

Through iterative approach the combination for size 10 window and 7 as the width of the normal distribution function, resulted in a signal that is very similar to the unfiltered drumloop1 sample with minimal loss of the important signal waves, which is determined by clarity of sound. Figure 11 shows the applied Gaussian filter, which notice it very similar to the original signal in blue.

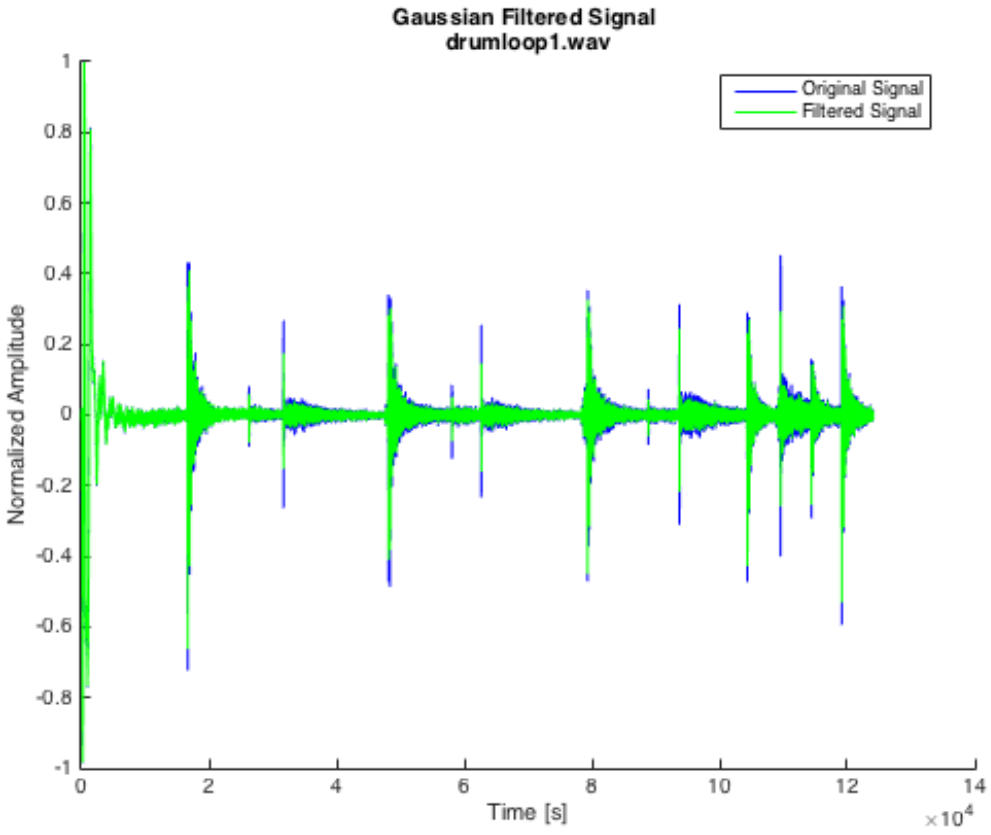

Figure 11- Gaussian filter effect on the drumloop1.wav audio file

The following parameter values are used for the median filter:

medianFilter (… , window = 3)

Table 9 - Median filter trials for drumloop1 signal

noise present?	**Yes**	**No better**	**No better**	**No worse**	**Static worse**	**No better**	**No better**	**Static worse**	**No better**
window	11	5	3	15	17	9	7	13	1

Through experimental approach the combination for size 3 resulted in the clearest filtered audio (see Figure 12) compared to the trials performed. However, as mentioned in section 2.1.3, the drumloop1 sample contains minimal amount of noise, thus the application of a low pass filter is damaging the signal by removing important signal waves that contribute to the clarity of the sound. the filter created static noise and introduced greater interference with the signal, which can be seen on. Due to the parameter set for the filter input, the unfiltered signal and filtered sounds almost the exact same with close to no loss in the important signal waves.

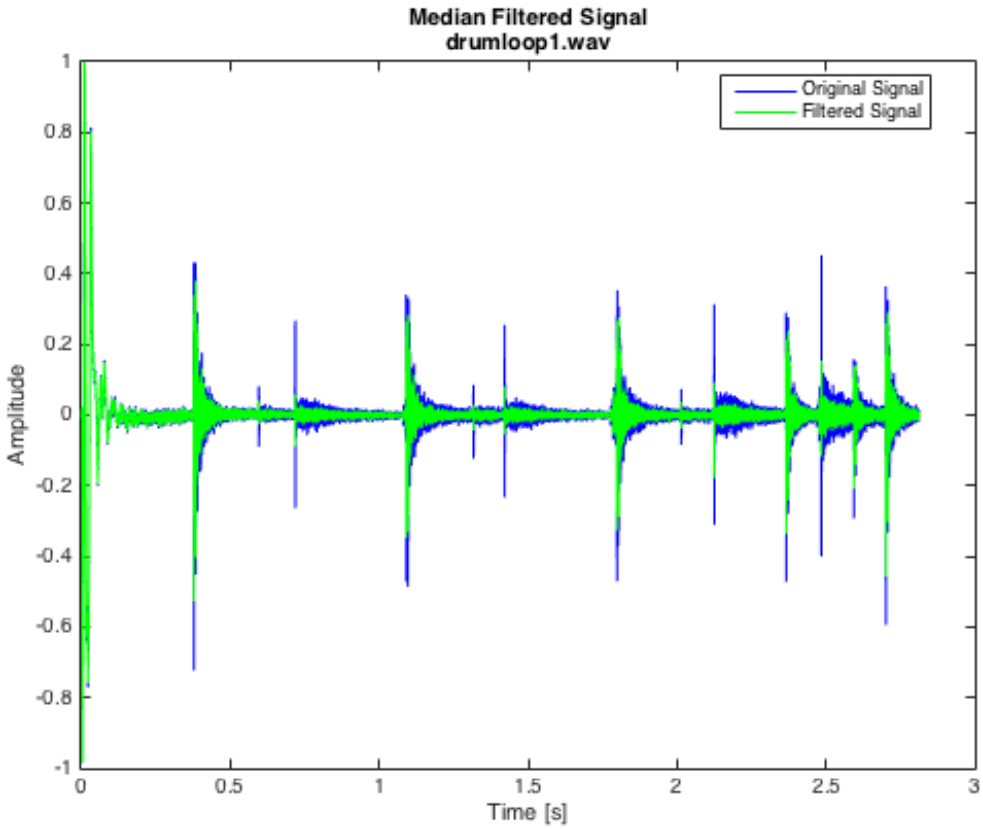

Figure 12 - Median filter effect on the drumloop1.wav audio file

3.0 TASK TWO – USING MATLAB FOR SIGNAL ANALYSIS

While looking over question 2, it became apparent that the three parts could be easily solved with a function that could count the number of peaks over a threshold. Thus, the following function is defined:

```
get_peaks(signal, sample_rate, window, threshold, num_find_peaks)
```

Parameters:

1) signal – the signal from the sound clip
2) sample_rate – the sample rate used to get the signal from the sound clip
3) window – window for the averaging filter applied within get_peaks
4) threshold – peaks will only be counted if they pass above this threshold
5) num_find_peaks – the number of times to run find_peaks on the signal, this will be expanded on further below.

This function returns an integer representing the number of peaks.

First, get_peaks initializes variables and runs an averaging filter on the signal using the inputted window. The built-in function find_peaks is then run on the signal num_find_peaks number of times. This function returns a list of amplitude values which are peaks in the input signal. Running it more than once further reduces oscillation and noise, creating a peak riding function. Figure 4, a visualization of the peak riding function for the drum loop sound after find_peaks was run 5 times, can be found in the appendix.

If we run find_peaks too many times on the signal, we end up losing too much information. If we run find_peaks too few times, we end up with too much noise and oscillation left in the signal.

Once we have a proper peak riding function, like the one above, we can count the number of times a peak occurs by counting the number of times the function goes above the threshold. Due to remaining noise in the signal, the function might oscillate about the threshold a few times at points of crossing. To prevent multiple readings, get_peaks uses a hysteresis with a high of threshold and a low of threshold * 0.85.

The following subsections will explain how each question was solved by showing experimentally determined get_peaks parameter values and a description of the extra logic specific to that question. A visualization of the average filtered signal and the PR function for each can be found in the Appendix. The signal and sample_rate parameters will be omitted since they are always determined as follows:

```
[signal, sample_rate] = audioread(sound_file_name, 'double');

num_peaks = get_peaks(signal, sample_rate ...);
```

3.1 Syllables in Tapestry Clip

The following parameter values are used to obtain the number of peaks:

```
get_peaks ( … ,
        window = 6,
        threshold = 0.07,
        num_find_peaks = 3 )
```

Each time a syllable is spoken, a peak occurs. By the definition of a syllable, the speaker then pronounces the next syllable after a very short pause. This pause causes a dip in the amplitude of the signal. Therefore, by counting the peaks above the background noise, the number of syllables can be accurately determined. In this case, the averaging filter helps to get rid of a random noise spike as well as accenting the syllables to make them easier to detect. The analysis resulted in calculating 10 syllables.

3.2 Beats per Minute in Drum Loop

The following parameter values are used to obtain the number of peaks:

```
get_peaks ( … ,
        window = 6,
        threshold = 0.02,
        num_find_peaks = 5 )
```

The number of beats per minute are calculated as the number of times the drum is hit during the sound clip, extrapolated to a number of hits per minute. Mathematically, this was done by dividing the number of samples by the sample rate to get the duration of the sound clip in seconds. The number of times the drum was hit was first calculated by average filtering the signal then counting the peaks above the given threshold. Using the average filter helps to smooth out the sound of the symbols so it is easier to isolate the drum. Finally, the BPM can be calculated using the following equation:

$$BPM = \left(\frac{60}{duration}\right) * hits$$

The analysis resulted in value for bpm = 106.6239

3.3 Number of Chirps by Clay Coloured Robin

The following parameter values are used to obtain the number of peaks:

```
get_peaks ( … ,
        window = 6,
        threshold = 0.03,
        num_find_peaks = 4 )
```

The chirps in this sound file clearly stand out when compared to the background noise. By smoothing the values using an averaging filter as well as the PR function, the chirps are easy to pick out using get_peaks. The number of peaks is equal to the number of chirps so no extra calculations are required. The analysis resulted in calculating 16 whistle chirps.

4.0 CONCLUSIONS

This report provides a clear mathematical definition of noise in a generic signal. Furthermore, a detailed explanation of noise present in each of the audio samples is provided in section 2.1. The next section, provides a solution to removing the noise from given samples using low pass filters. Three different low pass filters are designed: average, Gaussian and median. In addition, each filter is represented mathematically as difference and differential equations for discrete and continuous domains, respectively as well as impulse response functions. However as previously deducted, the median filter cannot be represented mathematically as the function requires a sorting algorithm which is a logical method, rather than a mathematical equation.

The Section 2.3, provides a detailed analysis of application of low pass filters and demonstrates the results of filtered signals for each audio sample. Furthermore, the method for determining the window size for each filter is purely experimental and iterative to achieve the purest signal.

After filtering ClayColoredRobin.wav audio sample through all three design filters, analysis shows that the most appropriate filter to remove noise from the signal is the Gaussian filter. The reasoning of analysis is that the median filter created interferences in the signal which introduced more noise into the system, hence the filtered signal contains random oscillations in amplitude (see Figure 6), and after average filtering the audio signal still contains the environmental background noise, where on Figure 4 it can be seen that the filtered signals is jittery, to while the Gaussian filter removes nearly all pink noise which is evident in Figure 5.

Filtering the tapestry.wav audio sample, it is determined that the average filter works best for this particular sample. The reasoning of this analysis is that the median filter creates static noise in the signal since it filters out useful signals as evident from Figure 9. The Gaussian filter basically does not change the signal from the original to filtered stages which can be seen in Figure 8 where the filtered (green) covers mostly all of original signal (blue). However, the average filter smoothens the overall signal, as well as removing the noise at ~0.7s and ~1.3s to create a clearer sound, see Figure 7.

Finally, the filtering the drumloop1.wav audio sample determined that no filters were optimal for the particular sample as the sample did not contain any significant noise to be filtered. Hence Figure 10 through 12 look nearly identical to each other. The filtered and original audio samples sound very similar since no small period noise is present in the sample to begin with.

5.0 RECOMMENDATIONS

Based on the conclusions drawn from filtering analysis of audio samples, it is evident that the ClayColoredRobin.wav is the sample that contained dominant low period noise which was removed by the low pass filters. However, for tapesty.wav audio sample the low pass filter was not as effective, for the reason that it may require a band pass filter to remove both the short period signal waves and very large period noise or a better signal capturing method. The reason why a better signal recording method may be required is because the microphone that was used to record the particular audio sample may be changing its gain based on the volume changes in the woman's voice or due to its internal logic thereby making the low pass filter ineffective. In addition, the drumloop1.wav audio sample appears to be auto synthesized which results in no noise of the signal, thus illuminating the need for a low pass filter, or filter of any kind.

Further recommendations for method of acquiring the window size for specific audio samples is to create spectrograms to determine the signal beginning and signal frequency end, allowing for visual representation of noise in the system. This allows for more accurate window size determination, as well as faster then experimentally comparing the sounds of different filtered signals.

Task two can also be solved using the envelope function available only in MATLAB 2015b which was unavailable for the timeline of the current project. Which allows for faster peak detection method as well as more accurate than experimentally determining the amount of times to loop findpeaks() function in MATLAB 2015a.

6.0 REFERENCES

[1] Vyacheslav Tuzlukov (2010), Signal Processing Noise, Electrical Engineering and Applied Signal Processing Series, CRC Press. 688 pages. ISBN 9781420041118 noise in general

[2] Downey, Allen (2012). *Think Complexity*. O'Reilly Media. p. 79. ISBN 978-1-4493-1463-7. "Visible light with this power spectrum looks pink, hence the name." pink noise

[3] Carter,Mancini, Bruce,Ron (2009). Op Amps for Everyone. Texas Instruments. pp. 10–11. ISBN 0080949487. – white noise

[4] "Shot Noise." *Wikipedia*. Wikimedia Foundation, 15 June 2015. Web. 23 Oct. 2015.

[5] "Transient Noise." *Wikipedia*. Wikimedia Foundation, 20 Oct. 2013. Web. 23 Oct. 2015.

7.0 APPENDIX

4.1 Figures

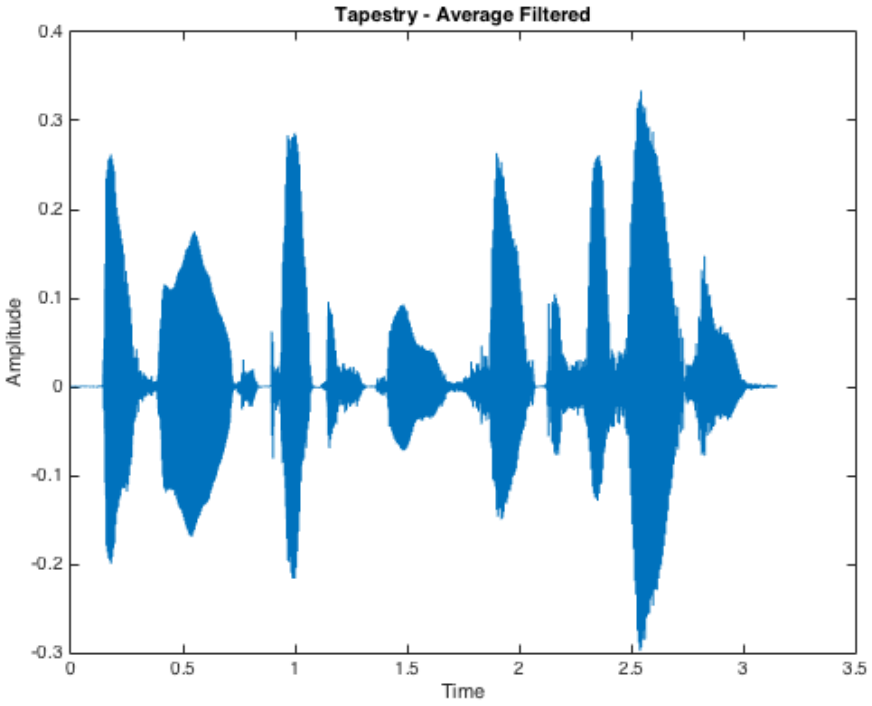

Figure 13: Average Filtered Signal of the Tapestry Clip

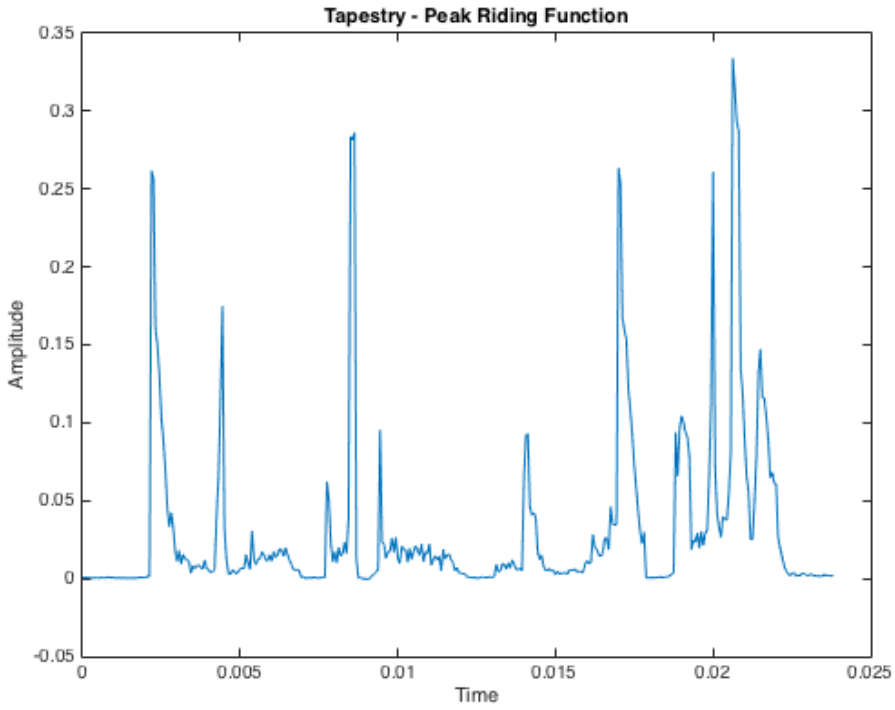

Figure 14: PR Function of the Tapestry Clip

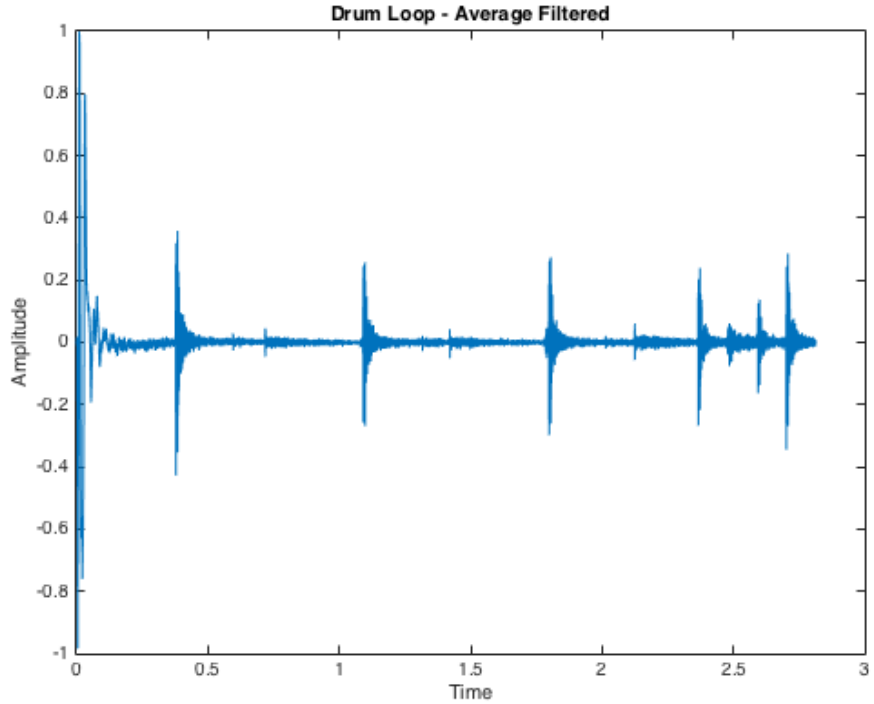

Figure 15: Average Filtered Signal of the Drum Loop

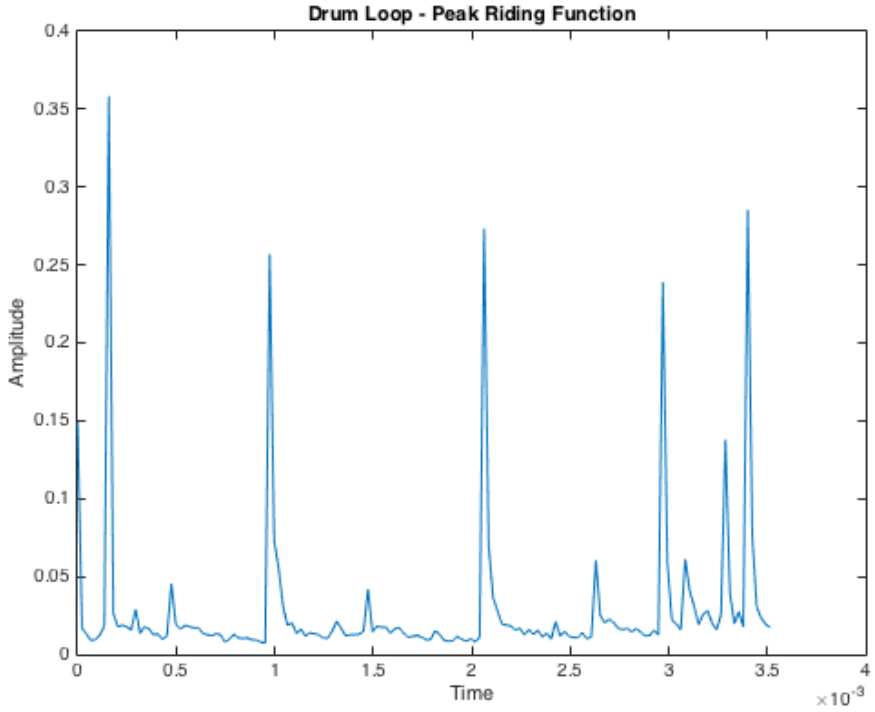

Figure 16: PR Function of the Drum Loop

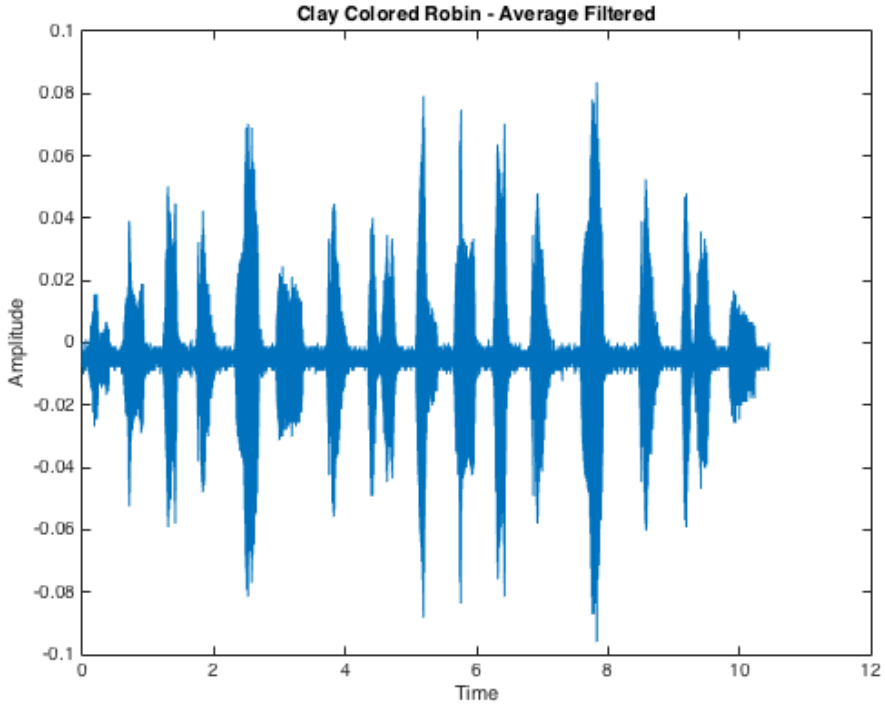

Figure 17: Average Filtered Signal of the Clay Coloured Robin

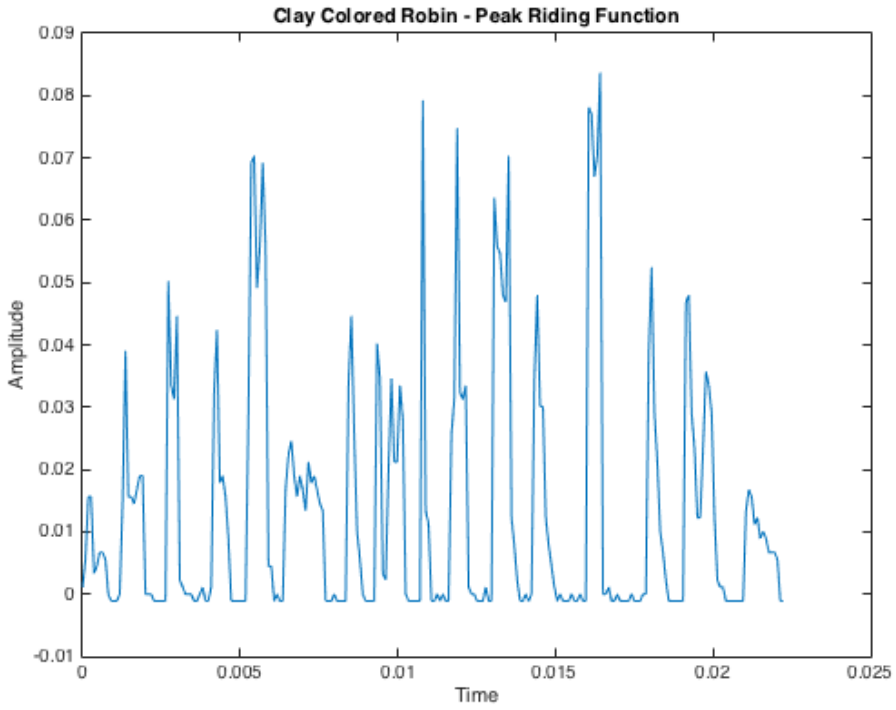

Figure 18: PR Function of the Clay Coloured Robin

4.2 Matlab Code

```
function y = avgFilter(signal, sample_rate, window)
%    sampling frequency
    t=0:1/sample_rate:(length(signal)-1)/sample_rate;

    y = zeros(length(signal), 1);
    % % For loop doing averaging
    for j = window/2 + 1 : length(signal) - window/2
        y(j) = mean(signal(j - window/2 : j + window/2));
    end

    y = double(y);
end
```

```
function [ smoothedAudio] = gaussianFilter( alpha, window, signal)
    % creating the gaussian smoothing function
    gaussFilter = gausswin(window,alpha);
    % normalize the guassian function
    gaussFilter = gaussFilter / sum(gaussFilter);
    % convolute the audio file with the guassianFilter
    [n,m] = size(signal);
    smoothedAudio = zeros(n,m);

%    to account for multiple channels in the signal, take the first one
%    smoothedAudioData = [];
%    for i = 1 : m
```

```
        smoothedAudio = conv(signal(:,1), gaussFilter);
%       smoothedAudioData = [smoothedAudioData; smoothedAudio];
%   end
end
```

```
function [y] = medianFilter(signal, window)
    y = zeros(length(signal), 1);
    % % For loop finding median
    for j = window/2 + 1.5 : length(signal) - window/2 - 0.5
        y(j) = median(signal(j - window/2 - 0.5 : j + window/2 + 0.5));
    end

    y = double(y);

end
```

```
%% Question One Code
% to replicate results of task 1 of the report simply press run
% uncomment sound(x, Fx) to hear original
% uncomment sound(y(:,1),Fx) to hear the filtered signal
clear all;
close all;
clc;

% list of sample files
fileName = {'ClayColoredRobin.wav', 'tapestry.wav', 'drumloop1.wav'};

% fileParam is a matrix of window parameters for each filter for each
audio
% sample
%     row one correspods to audio file one in the list
(ClayColoredRobin)
%               Avg window   gaussF N     gaussF alpha       median
window
% fileParam = [     10,          25,             7,             13   ;
...]

% N - the number of points for the gaussian function, aka window
% alpha - proportional to reciprocal of standard deviation, aka width
of normal distribution
fileParam = [10, 25,  7, 13;
              2, 25, 25,  1;
              2, 10,  7,  3;];


% plotting noise
for i = 1 : length(fileName)
    [x,Fx] = audioread(fileName{i},'double');
    figure;
    plot(x(:,1), 'b');

    xlabel('Time [s]')
    ylabel('Amplitude')
```

```
    title([fileName{i}]);
    legend('Original Signal','Location','best')
end

% plotting filtering

for i = 1 : length(fileName)
    %% Plot filtered graphs for an audio sample in the fileName array
    % Average Filter
    [x,Fx] = audioread(fileName{i},'double');

    y = avgFilter(x(:,1), Fx, fileParam(i,1));

    t=0:1/Fx:(length(y)-1)/Fx;

    % plot original signal
    figure;
    plot(t,x(:,1), 'b');
    hold on;
    plot(t,y(:,1), 'g');

    xlabel('Time [s]')
    ylabel('Amplitude')
    title(['Averaged Filtered Signal ', fileName(i)]);
    legend('Original Signal','Filtered Signal','Location','best')
    % sound(x,Fx);
    % sound(y(:,1),Fx);


    %Gaussian Filter
    % plot original signal
    figure;
    hold on;
    plot(x(:,1), 'b');

    smoothedAudio= gaussianFilter(fileParam(i, 3), fileParam(i,2), x);
    % plot filtered signal
    plot(smoothedAudio(:,1), 'g');
    xlabel('Time [s]');
    ylabel('Normalized Amplitude');
    title(['Gaussian Filtered Signal ', fileName(i)]);
    legend('Original Signal','Filtered Signal','Location','best')
    % sound(smoothedAudio, Fx);


    %Median Filter
    % plot original signal
    figure;
    plot(t,x(:,1), 'b');
    hold on;

    h = medianFilter(x, fileParam(i,4));
    t=0:1/Fx:(length(h)-1)/Fx;

    plot(t,h(:,1), 'g');
    xlabel('Time [s]')
```

```
    ylabel('Amplitude')
    title(['Median Filtered Signal', fileName(i)]);
    legend('Original Signal','Filtered Signal','Location','best')
    % sound(y,Fx);
end
```

```
% % % get_peaks function definition
function num_peaks = get_peaks(signal, sample_rate, window, threshold,
num_find_peaks)

    % Initialises num_peaks to 0
    num_peaks = 0;
    % Initialises to a "low" state, this menas that the peak riding
    % function value is not currently in a peak.
    state = 'low';

    % Apply the averaging filter
    signal = avgFilter(signal, sample_rate, window);

    %     % Plotting the function for a visual representation of what's
going on.
    %     t=0:1/sample_rate:(length(signal)-1)/sample_rate;
    %     plot(t, signal);
    %     xlabel('Time')
    %     ylabel('Amplitude')
    %     title('Drum Loop - Average Filtered')

    % Initialize signal_peaks. This is used to better identify the
purpose of
    % the variable form here on out.
    signal_peaks = signal;

    % Finds peaks of the signal three times. This effectively creates a
list of
    % values that rides the peaks of the sine waves. Doing it three
times
    % reduces most of the noise in this function and clearly isolates
the
    % function riding the peaks of the signal.
    for iteration = 1 : num_find_peaks
        signal_peaks = findpeaks(signal_peaks, sample_rate);
    end

    %     % Plotting the function for a visual representation of what's
going on.
    %     figure;
    %     t=0:1/sample_rate:(length(signal_peaks)-1)/sample_rate; %
sampling frequency
    %     plot(t, signal_peaks);
    %     xlabel('Time')
    %     ylabel('Amplitude')
    %     title('Drum Loop - Peak Riding Function')

    % Iterate through the peak riding function. The if statements and
```

```
states
    % are used to essentially count the number of "bumps" rising pas
the
    % threshold cutoff in the sound clip.
    for value = 1 : length(signal_peaks)
        if (strcmp(state,'low') && signal_peaks(value) > threshold)
            num_peaks = num_peaks + 1;
            state = 'high';
        % By using a value of threshold*0.85 to go back to low state, a
        % hystersis effect can be simulated with the code. This is
helpful
        % because a peak riding function may still have some noise and
it is
        % undesireable to overcount the number of threshold crossings.
        elseif (strcmp(state,'high') && signal_peaks(value) <
(threshold * 0.85))
            state = 'low';
        end
    end
end
```

```
% % % Question 2 i)
clc
close all

% Load the tapestry sound clip
[signal,sample_rate] = audioread('tapestry.wav','double');

% Obtain the number of peaks (syllables)
syllables = get_peaks(signal, sample_rate, 6, 0.07, 3);

% Displaying syllables to the user
display(syllables);
```

```
% % % Question 2 ii)
clc
close all

% Load the drumloop1.wav sound clip
[signal,sample_rate] = audioread('drumloop1.wav','double');

% Get the signal duration in seconds
duration = length(signal) / sample_rate;

num_beats = get_peaks(signal, sample_rate, 6, 0.2, 5);

% The BPM is equal to the (# of beats / minute) or (1 minute/ seconds
% played) * (# of beats) for sound files less than a minute in length.
% Since the drumloop is just under 3 seconds in length, the second
formula
% will be applied.
bpm = (60 / duration) * num_beats;
```

```
% Displaying syllables to the user
display(bpm);


% % % Question 2 iii)
clc
close all

% Load the bird chipring sound clip
[signal,sample_rate] = audioread('ClayColoredRobin.wav','double');

% Get the number of peaks (chirps)
num_chirps = get_peaks(signal, sample_rate, 6, 0.03, 4);

% Displaying syllables to the user
display(num_chirps);
```