DSP Final Project - Fall 2019

Obadiah Boda, Katherine Latta, Mashroor Rashid, Zvezda Spring

Department of Electrical and Computer Engineering

Missouri University of Science and Technology

Table of Contents

[**Introduction**](#_i4gm7t3skbw) **3**

[**Description of Music**](#_8cq4mwgzqi97) **4**

[**Method: FIR Filter**](#_cbgyxsgsvav5) **5**

[**Method: Pre-programmed Filter**](#_pl2k6l9t9h5w) **6**

[**Conclusion**](#_cj59io6afprt) **7**

[**Appendix**](#_hurvcmn8d60f) **8**

# 

# 

# 

# 

# Introduction

Filtering has become a common tool when utilizing digital signals. For example, say we’re filming a video for a school project, and upon reviewing the footage, we notice a low drone from the A/C unit. Through our video editing software, we can apply a “hum” filter, to take out the A/C drone, and leave the actor’s voice intact. This type of audio filtering is just one of many applications of digital signal filtering.

At the most basic level, a filter will have a fourier transform applied to it. When we apply this transform to an audio signal in the time domain, we will receive a signal in the frequency domain. Once in the frequency domain, a power spectrum plot should be used to see the power levels at each frequency component of the signal. From this visualization, we can decide which frequencies to include or exclude. These values are then used when designing a Finite Impulse Response (FIR) or Infinite Impulse Response (IIR).

This experiment will be used to verify that an FIR built in MATLAB will be comparable to MATLAB’s built-in filtering functions.

# Description of Music

When trying to find audio for the following filter applications, we tried to find clips that contained a very wide range of frequencies. With a wide range of frequencies, we can easily attenuate a specific range of frequencies; in our case, we decided to filter out higher frequencies.

After listening to many clips, we decided to look for percussion-based music, as the wide array of frequencies made it very easy to filter out hi-hats and snare drum sounds. The clip we chose consists of only percussion instruments; namely a piano, bass drum, snare drum, and hi-hat symbol.

# Method: FIR Filter

The first step we did was read in the audio file that we had provided. Then we took the FFT and the Power Spectrum of the signal (See Appendix B and C). We then declared filter order M as 140. Later we also declared the normalized passband and stopband frequencies as constants set to particular values. Which together calculated the cutoff frequency, by averaging the two frequencies. Following that we calculated the filter length through adding one to the filter order M. Furthermore, we created the lowpass impulse filter response through the usage of the formula provided (See Appendix N). We multiplied the Hamming Window by the low pass filter response to each other. After that we took the fft of that product with the original music signal.

In addition, we looked at the produced signals in decibel and in normalized frequency; we plotted these representations. We changed the orientation of the windowed fft from the FIR filter. We got the output through multiplying the changed windowed fft and the fft of the original signal. After this we used the ifft to go back to the original data set and domain. Then we played the audio clip and created an audio file.

Following this, we calculated the length of the signal. Also, we took the fft of the signal and set the frequency variable range. Later we calculated the power of the signal with these calculations. Finally, we plotted the Power Spectrum and Normalized Frequency graphs.

# Method: Pre-programmed Filter

We had the option of using the Chebyshev, it might’ve had more benefits such as having a steeper roll off that would give us more distinctive frequencies through the usage of the filter. Alas, the audio file that we used didn’t necessarily call for such features that Chebyshev provided. So, instead, we used the Butterworth preprogrammed filter in MatLab.

Initially we used the Butterworth default values of alpha and beta. So alpha would be initially set to the value of one and beta would be set to 30; alpha and beta affect the range of the magnitude. Then, we used a stopband frequency (normalized fs) and a passband frequency (normalized fp) that would be appropriate for dealing with low frequency inputs. The stopband frequency and passband frequency were used to give a general location of where the filter would operate. We knew that later on we would have to adjust these values to get a better output of signals. We gave initial normalized values of the passband and stopband. We then fed these values into the Passband and Stopband formulas (See Appendix L) to find the non-normalized frequencies. They are then reused to calculate the order of the filter also known as n, using a given a formula (See Appendix K).

Then we calculated the corner frequency through the formula provided (See Appendix M), through the usage of the previously calculated non-normalized frequency stopband. Afterwards, we calculated the normalized cutoff frequency through implementing the corner frequency into the formula (See Appendix N). We made adjustments to the original four variables: alpha, beta, normalized fs, and normalized fp. We changed these values to manipulate the corner frequency. We printed out the FFT and the Power Spectrum graph representations of the Butterworth Filter.

# Conclusion

In conclusion, we were able to create a low pass filter using MATLAB. However, our low pass filter was not as effective as the built-in Butterworth filter in MATLAB.

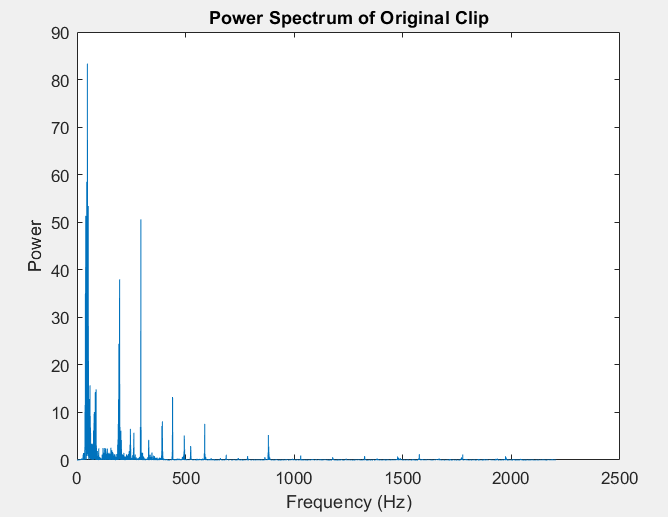
We were able to filter out our targeted frequencies (hi-hat, Snare drum), but the Butterworth filter in MATLAB filtered out all instruments but the bass drum, which produced the lowest frequencies in the audio clip. Additionally, our filter also affected the clarity of the music; our filtered audio file was muffled compared to the original clip. When implementing the Butterworth filter, it should be noted that the benefits of this filter come from it’s low ripple; in our application, it was not important, since our pass band frequency shifted our filter magnitude far enough to avoid capturing a ripple.

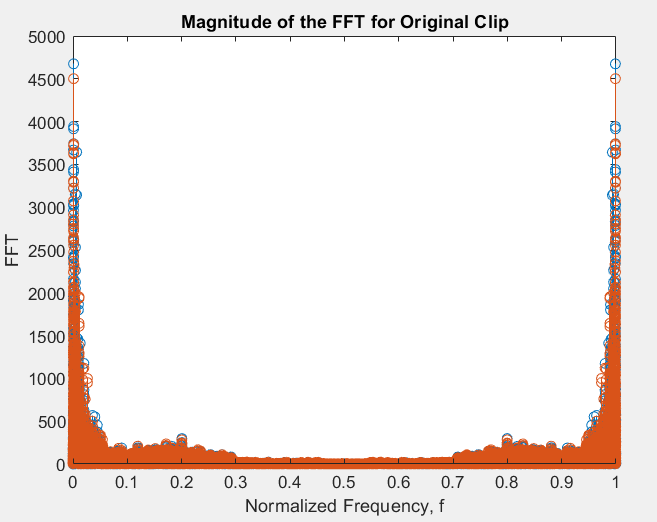
In future replications of this experiment, we would recommend choosing a Chebyshev filter as opposed to a Butterworth filter. We came to this conclusion after examining the properties of each filter. The steeper roll-off of the Chebyshev filter yields a cleaner filter, compared to the Butterworth filter.

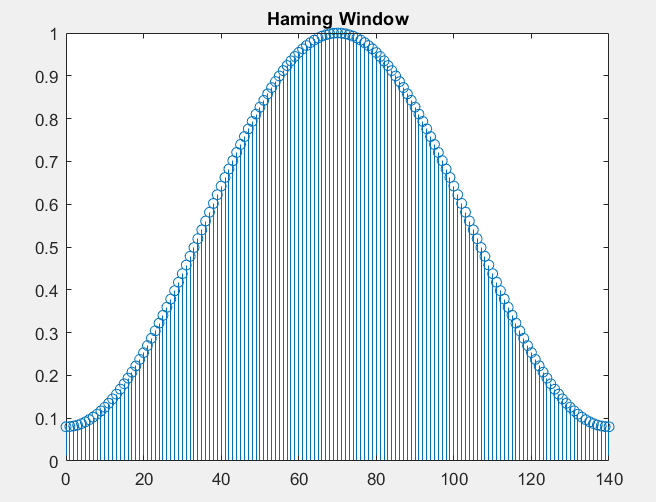
# Appendix

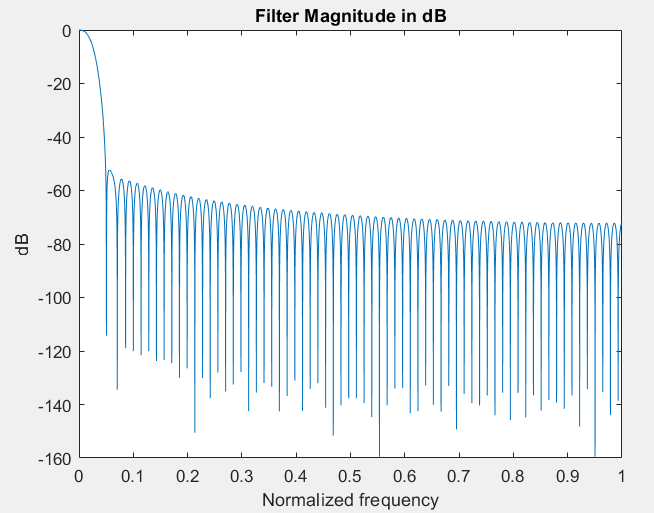
Appendix A: Matlab code

|  |
| --- |
| clear all% this clears all your variables [x,Fs]=audioread('clip.wav'); % reads in the file sound(x,Fs); %play original file pause(3);  m=length(x); %calculates the length of the signal m1=pow2(nextpow2(m)); % chooses the next higher power of 2 X=fft(x,m1); % takes the fft of signal f=(0:m1-1)\*(Fs/m1); % sets your frequency variable range power=abs(X).^2/m1; % calculates the power of the signal figure(1); % creates a new plot window plot(f(1:floor(m1/20)),power(1:floor(m1/20))) % plots the power xlabel('Frequency (Hz)'); % labels the horizontal axis ylabel('Power'); % labels the vertical axis title('Power Spectrum of Original Clip'); % graph title figure(2); % creates a new plot window nf=0:1:m1-1; % normalized frequency variable for plotting fft stem(nf./m1,abs(X)); % plot of fft magnitude xlabel('Normalized Frequency, f'); %labels horizontal axis ylabel('FFT'); % labels vertical axis title('Magnitude of the FFT for Original Clip') %graph title  M = 140 % filter order n = 0:1:M; % time index fs = .005; %220hz  fp = .02; %882hz fc = (fs+fp)/2; % cutoff frequency L = M +1; %filter length  hd= 2\*fc\*sinc(2\*fc\*(n-(M/2))); %shifted low pass filter impulse response w = .54 -.46\*cos(2\*pi\*n/(L-1)); %hamming window  figure(3); stem(n,w); title('Haming Window');  hw = hd.\*w; %windowed impulse response Hw = fft(hw,m1); %windowed FFT  Hw2 = Hw(1:(m1/2)); HWdB = 20\*log10(abs(Hw)); %convert to decibel HWdB2 = HWdB(1:(m1/2)); %shorten to only plot to f = 0 to .5 n2 =0:1:(m1/2)-1; %new plot index figure(6); plot(n2/(m1/2),HWdB2);  title('Filter Magnitude in dB'); xlabel('Normalized frequency'); ylabel('dB');   HWc = Hw.'; %changed orientation of windowed fft Y = HWc.\*X; %output y = ifft(Y); %y[n] sound(y,Fs); %play filtered clip pause(4); audiowrite('filtered.wav',y,Fs);   m=length(y); %calculates the length of the signal m1=pow2(nextpow2(m)); % chooses the next higher power of 2 X=fft(y,m1); % takes the fft of signal f=(0:m1-1)\*(Fs/m1); % sets your frequency variable range power=abs(X).^2/m1; % calculates the power of the signal figure(4); % creates a new plot window plot(f(1:floor(m1/20)),power(1:floor(m1/20))) % plots the power xlabel('Frequency (Hz)'); % labels the horizontal axis ylabel('Power'); % labels the vertical axis title('Power Spectrum Components For filtered'); % graph title figure(5); % creates a new plot window nf=0:1:m1-1; % normalized frequency variable for plotting fft stem(nf./m1,abs(X)); % plot of fft magnitude xlabel('Normalized Frequency, f'); %labels horizontal axis ylabel('FFT'); % labels vertical axis title('fft filtered') %graph title    %butterworth filter alpha = 1; beta = 50; fs1 = .01; %normalized fs fp1 = .001; %normalized fp fs = (Fs/pi)\*tan(pi\*fs1); %stop frequency formula fp = (Fs/pi)\*tan(pi\*fp1); %pass frequency formula  FO = ceil((log10(10^(0.1\*alpha)-1)-log10(10^(0.1\*beta)-1))/(2\*(log10(fp)-log10(fs)))); %sets filter order   Fc = fs/((10^(0.1\*beta)-1)^(1/(2\*FO))); %determines corner frequency Wn = Fc/(Fs/2); %normalized cutoff frequency [b, a] = butter(FO, Wn); %creates low pass filter via butterworth function Filtered\_S = filter(b,a,x); %calculates filtered signal audiowrite('ButterworthSignal.wav', Filtered\_S, Fs); %writes filtered audio clip sound(Filtered\_S, Fs); pause(4);    m = length(Filtered\_S); %calculates the length of the signal m1 = pow2(nextpow2(m)); % chooses the next higher power of 2 FFT = fft(Filtered\_S, m1); % takes the fft of signal f = (0:m1-1) \* (Fs/m1); % sets your frequency variable range Filtered\_P = abs(FFT).^2/m1; % calculates power of the signal figure(7); plot(f(1:floor(m1/20)), Filtered\_P(1:floor(m1/20))); xlabel('Frequency'); ylabel('Power'); title('Power spectrum for butterworth filtered music');  figure(8); % creates a new plot window  nf=0:1:m1-1; % normalized frequency variable for plotting fft  stem(nf./m1,abs(FFT)); % plot of fft magnitude  xlabel('Normalized Frequency, f'); %labels horizontal axis  ylabel('FFT'); % labels vertical axis  title('Magnitude of the FFT for Butterworth Filter') %graph title |

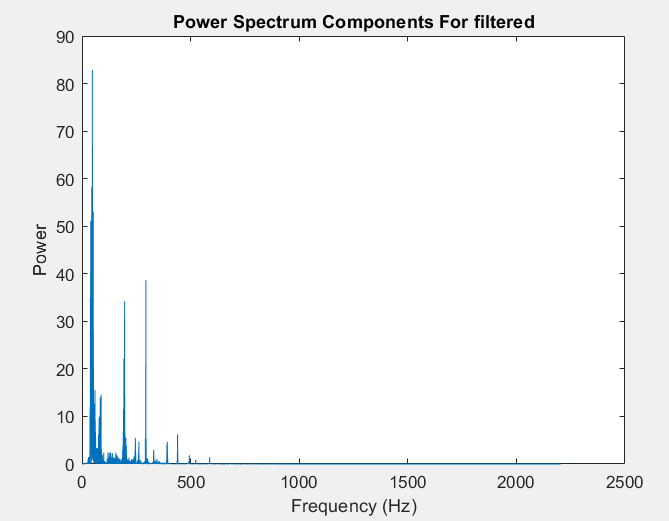
**Appendix B:** Power Spectrum of Original Clip 

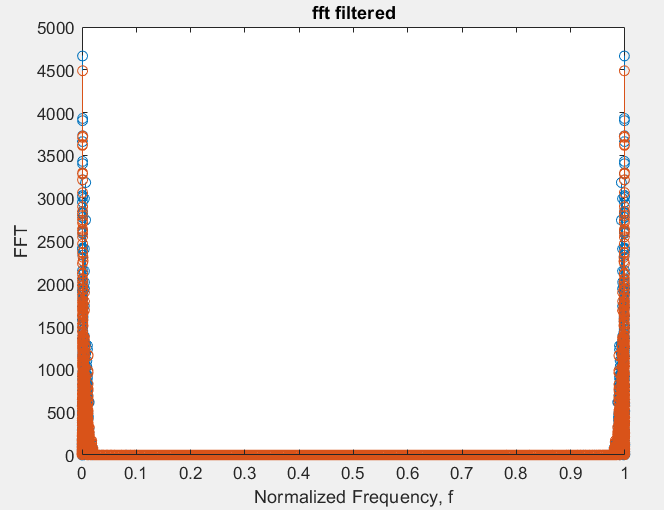
**Appendix C:** FFT Original 

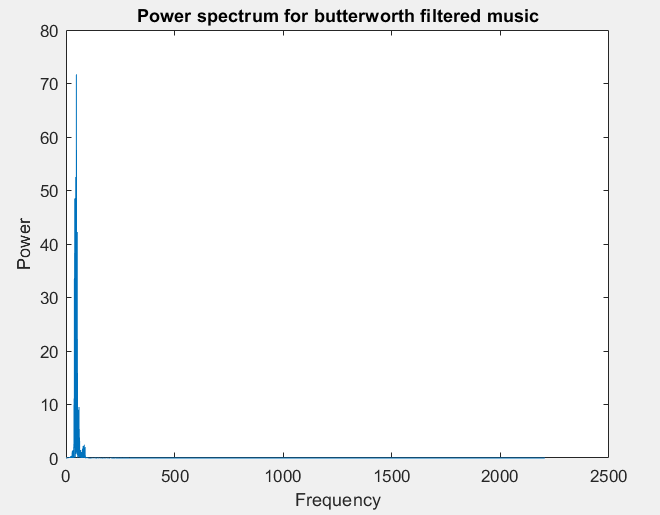
**Appendix D:** Hamming Window

**Appendix E:** Filter Magnitude in dB 

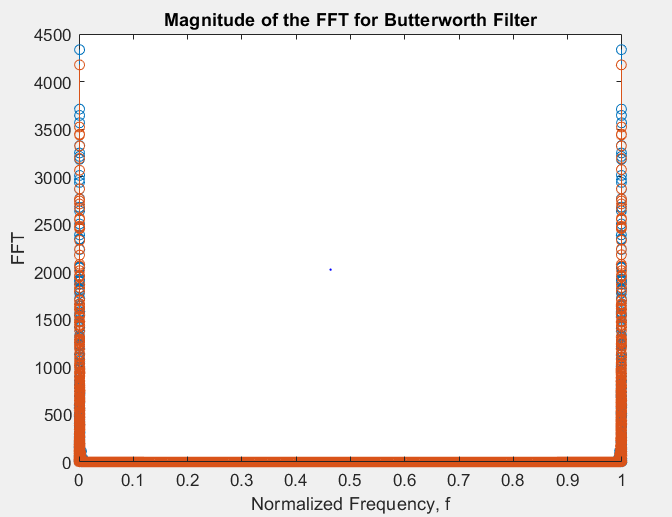
**Appendix F:** Power Spectrum of Filtered Clip



**Appendix G:** FFT Filtered****

**Appendix H:** Power Spectrum Butterworth 

**Appendix I:** Power Spectrum Butterworth

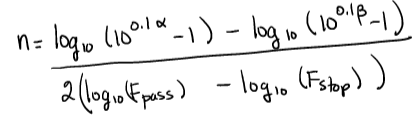


**Appendix J:** Audio Clip

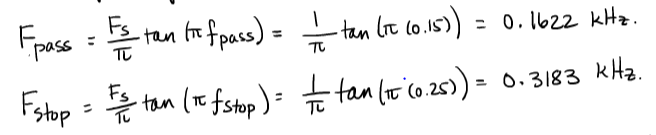
<https://www.dl-sounds.com/royalty-free/category/holiday-season/> Jingle app beat: clip credit

Citation: "Free Royalty Free Holiday - Season Christmas Music." *DL Sounds*, www.dl-sounds.com/royalty-free/category/holiday-season/.

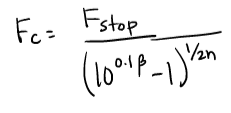
**Appendix K:** Filter Order Calculation



**Appendix L:** Normalized Frequencies for Butterworth Filters



**Appendix M:** Cutoff Frequency for Butterworth Filters

****

**Appendix N:** Low Pass Filter Impulse Response (FIR)

