ENGG*3390- Signal Processing

Lab 5 Report

Bilal Ayyache- 0988616

Palak Sood - 0986802

Kathlene Titus – 0954584

November 29th, 2018

Introduction

The purpose of this lab is to apply a few frequency-based approaches to digital filter design in MATLAB. When sampling an audio signal, several information can be collected as data. The Fast Fourier Transform (FFT) function in MATLAB calculates the Discrete-Time Fourier Series (DTFS) of a signal. The DTFS is defined only for periodic signals. It is known that it is faster to perform the filtering in the frequency domain than in the time domain due to the computational advantages for the FFT algorithm. In this lab, the original signal is transformed using the FFT function and is analyzed in the frequency domain to identity a pattern in their periodicity. As a result, collecting the extracted data (k indices), the creation of a suitable filter (impulse of such indices), and the filtering of noise of the original signal in time domain (convolution of the signal with the filter) is done. Moreover, it was found that the truncation of the filter and the segmentation of each frequency was a critical part of the experiment affecting the ability to obtain the desired results. The limitation of the filters impulse coefficients correlated with its ability to filter noise and pass through tones, which showed the importance of selecting an appropriate filtering window. Therefore, the proper implementation of a well-designed filter was the most important part during this experiment.

Materials

During the lab, the following tools and equipment we used:

1. Function generator and oscilloscope
2. Input audio source, e.g. PC workstation, MP3 player, headphones, etc.
3. TI TMS320C5505 eZdsp USB Stick with cable, audio patch cords
4. Lab 5 project files as posted on CourseLink
5. TI Code Composer Studio (CCS) SDK

Procedure

Lab 5 consisted of three different parts; the frequency analysis, the frequency domain filtering, and the time domain filtering. All filtering and analysis were completed using MATLAB. The DSP implementation was implemented using one of the time domain filters. The SETI_Signal_A.wav audio was used as the main signal for the three different parts of the lab.

The frequency analysis of signal A involved reading the audio file into MATLAB and getting the Discrete-time Fourier series of the signal. After calculating the Fourier series, the 5+5 peak indexes that corresponds to the 5 tones embedded in the signal was found. The audio file

was read into MATLAB using the audioread with the given file name. The magnitude of the discrete-time Fourier series of the signal was found using MATLAB. Code can be seen below:

```
[y,Fs] = wavread('SETI_Signal_A.wav');
fft_SigA = fft(y);
fft_SigA_abs = abs(fft_SigA);
plot(fft_SigA_abs);
A = find(fft_SigA_abs>200);
```

The ifft function gave the DTFS in real and imaginary values so the absolute of the DTFS was plotted instead. The indexes of the 5+5 peaks that corresponded to the 5 tones hidden in the signal were found by using the function “findpeaks" which found the -frequencies of the signal with the highest values with their index and arranged them from highest to lowest. The first 10 peaks selected were the 5+5 peaks needed.

The frequency domain filtering of Signal involved two sections: using a filter length of N = 60000 and using a filler length of N = 12000. With a filter length of N = 60000, the original signal was used since it had a filter length of N = 60000. The filter H[k] was found by making all the values of the frequencies in which the 5+5 peaks didn't occur to be zero and the 5+5 peaks to be 1.

Once the filter H[k] was found, the filter's impulse response, h[n], was calculated using the function “ifft” with the filter h[k]. The filter H[k] was applied to the absolute DTFS of signal since the DTFS meant that the signal was already in the frequency domain. The result was then convened back to time domain by using the ifft function. The code for these processes is given below.

```
[y,Fs] = wavread('SETI_Signal_A.wav');
fft_SigA = fft(y);
N = 60000;
filt = zeros(N,1);
freq_loc = [1308, 1961, 2618, 2936,
3298,
56704, 57066, 57384, 58041, 58694];
        filt(freq_loc) = 1;
output = filt.*fft_SigA;
outtime = ifft(output)
sound(outtime);
```

For the filter length of N = 12000, the signal was divided into 5 segments of 1 second since the signal was originally 5 seconds long with a filter length of N = 60000. The ifft function was used to find the DTFS for each of the five segments and the DTFS for each of the segments were given in real and imaginary values so the absolute of the DTFS was used instead. The code below was used in dividing the signal into 5 segments and finding the DTFS for each of the segments:

```
audioread('SETI_Signal_A.wav');
yone = y(1:12000);
ytwo = y(12001:24000);
ythree = y(24001:36000);
yfour = y(36001:48000);
yfive = y(48001:60000);
y1f = fft(yone);
y2f = fft(ytwo);
y3f = fft(ythree);
y4f = fft(yfour);
y5f = fft(yfive);
```

Each of the five segments contained a 1+1 peak in it. The indexes of the 5+5 peaks that corresponded to the 5 tones hidden in the signal were found by using the function “findpeaks" which found the frequencies of the signal with the highest values with their index and arranged them from highest to lowest. The first 2 peaks for each of the five segments were selected and those were the 5+5 peaks needed to make the filter H [kl. The filter h[k] was found by making all the values of the frequencies in which the 5+5 peaks didn't occur to be zero and the 5+5 peaks to be 1.

Once the filter H[k] was found, the filter's impulse response, h[k], was calculated using the function "ifft" with the filter H [K]. The filter H [K] was applied to each of the absolute DTFS of signal since the DTFS meant that the signal was already in frequency domain. The code used in calculating the impulse response and applying the filter H[K] to each of the five segments in frequency domain can be found in the appendix

The results gotten from applying the filter H[k] to each of the five segments in the frequency domain were converted back to time domain using the RI function. The One domain for the segments were joined back together using the vercat function since each segments time domain was given in one column.

The time domain filtering of Signal A was split into three parts which include: using the first impulse response, using the second impulse response and using the second impulse response truncated. The impulse response of the filler length of N = 60000 was used to filter the original signal (signal A) with the use of the filter function with B = impulse1, A = 1 and y = Signal A. The impulse response of the filter length of N = 12000 was used to filter the original signal by using the filter function with B = impulse2, A = 1 and y = Signal A.

The impulse response of the filter length of N = 12000 was truncated to a filter length of N = 256. The normalization was done by finding the square of the sum of bin by using the dot function. The truncated impulse response was then divided by the square root of the dot product of h [nl in order to get the square of the sum of h[n] to be 1. After the truncated impulse response was normalized to have a unit energy of 1, it was used to filter the original signal.

The truncated impulse response coefficients calculated using MATLAB were used for the DSP implementation. The coefficients needed to be scaled in order to be used for the DSP. The scaling of the coefficients done using MATLAB is given below

newImpulse = ceil (neuvimpulse•2"15);

The scaled filter coefficients used for the program are given below:

Int16 coeffs[FILTER] = {6549,6322,5660,4622,3296,1797,251, ….. }

The variable “coeffs” in configuration.c was set to the scaled filter coefficient values. The Lab 5 project was then compiled and ran with the SETI_Signal_A.wav audio file playing as the input audio source.

Results/ Discussion

2.1 Frequency Analysis

For the frequency analysis of the audio signal 'SETI_Signal_A.wav', the function audioread was used to input the audio into MATLAB and we also used the function sound to listen to it. The audio signal was distorted and had a lot of noise. However, there were 5 tones that were audible behind the noise. As seen in Figure 1, the magnitude of DTFS of the periodic signal was plotted using the MATLAB function fft.

The indexes of the 5 + 5 peaks corresponding to the 5 tones that appear embedded in the signal were using the MATLAB find function. As seen in Table 1, the indexes were converted to their Fourier coefficient indexes, k. Since MATLAB indexing reference is k+1, the actual k was calculated by subtracting 1 from the indexes. The discrete-time frequency, ω, was found by calculating $2\pi k/N$, where N = 60000. The continuous-time frequency, f, was calculated using the equation kf_s/N, where f_s = 12 kHz.

Table 1: Fourier coefficients indexes, k, and their corresponding discrete and continuous frequencies

k+1	Fourier coefficient indexes (k)	Discrete-time Frequency (ω)	Continuous-time Frequency (f)
1308	1307	0.136869	261.4
1961	1960	0.205251	392
2618	2617	0.274052	523.4
2936	2935	0.307352	587
3298	3297	0.345261	659.4
56704	56703	5.937924	11340.6
57066	57065	5.975833	11413
57384	57383	6.009134	11476.6
58041	58040	6.077935	11608
58694	58693	6.146317	11738.6

2.2 Frequency Domain Filtering

A signal was filtered in the frequency domain with filters of lengths N=60000 and N=12000. The filter was designed by creating a vector of the length N and setting all the values to zero except for the indexes. For N = 60000, filter's impulse response, h1[n] was plotted by using the ifft function as seen in Figure 2.

The filter was then applied to the signal in the frequency domain. When the filter length of N=60000 is applied, the audio seemed to be loud, noisy, and the tones were diverted. If the tones of the signal were to change over period N, the change would not be heard since the rest of the values in the filter have been zeroed out therefore not allowing for the change to be heard. Whereas, at N=12000 the tones were found to be clear and were distinguished from each other. Figures 5 and 6 shows the impulse response of filter with length N=60000 and N=12000, respectively.

2.3 Time Domain Filtering

The second impulse response, h2[n] was used to filter the original signal, by truncating the filter and normalizing the coefficients. Figure 3 shows filter the impulse of h2[n] in time

domain. The resulting audio sounded a little distorted and there was some noise present, however the tones could be distinguished from each other despite some noise and distortion.

The effects of the truncated filter resulted in a low performance since the length of the filter was reduced. This happens because every single coefficient has an effect when it is convolved with a signal. When the filter was being created, any index that was not corresponding to a peak was zeroed. This process of zeroing in frequency domain corresponds to a coefficient in the time domain. Therefore, every filter length that is removed reduces the zeroing effect on the noise by the filter. Similarly, indexes corresponding to a peak were allowed through, resulting in impulse response coefficients in the time domain. So, when coefficients are truncated, the filter's ability to pass through the desired tones is also reduced. Figure 4 shows the impulse response of truncated time domain filter h3[n]. This implies that the COLA method does affect performance if the filter is greatly truncated to apply it.

Conclusion

In conclusion, this lab enabled us to filter in frequency and time domain by using the Fast Fourier Transform as well as the Windowed Frequency Domain Filtering. It is evident that by using the Windowed Frequency Domain Filtering, the tones of the signal are audible and distinguished much more clearly than when using a filter with the length of the signal. Finally, when truncating the filter, the quality of the signal worsens.

Appendix

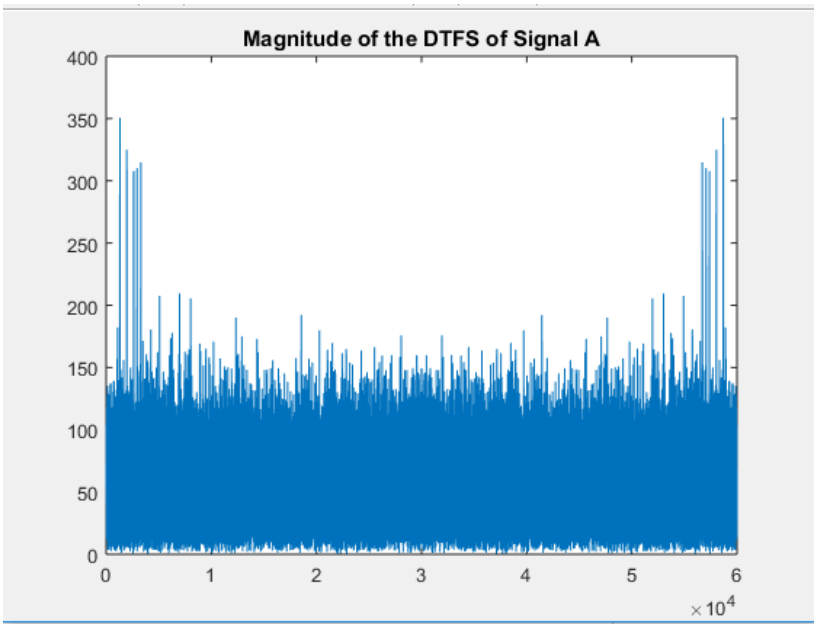

Figure 1: Magnitude of the DTFS of Signal A

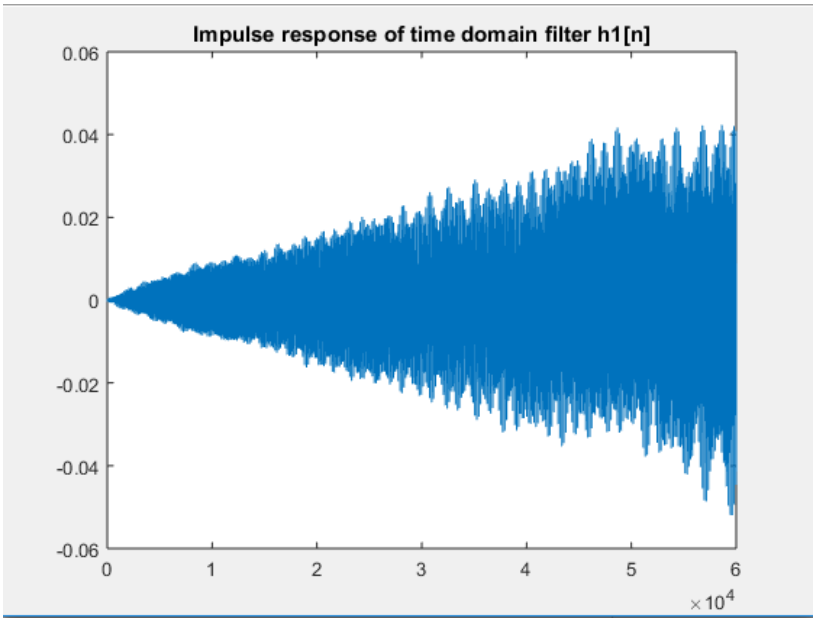

Figure 2: Impulse response of time domain filter h1[n]

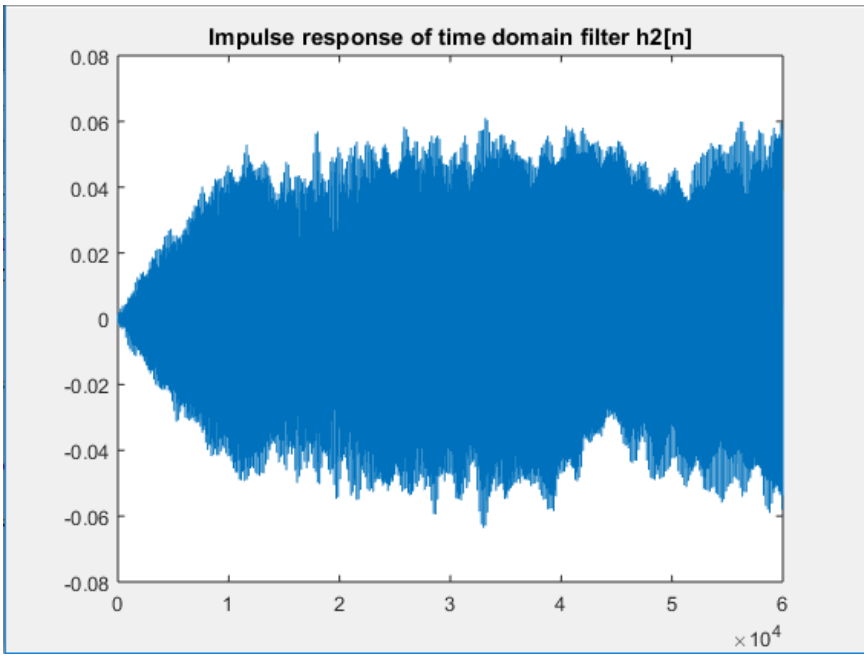

Figure 3: Impulse response of time domain filter h2[n]

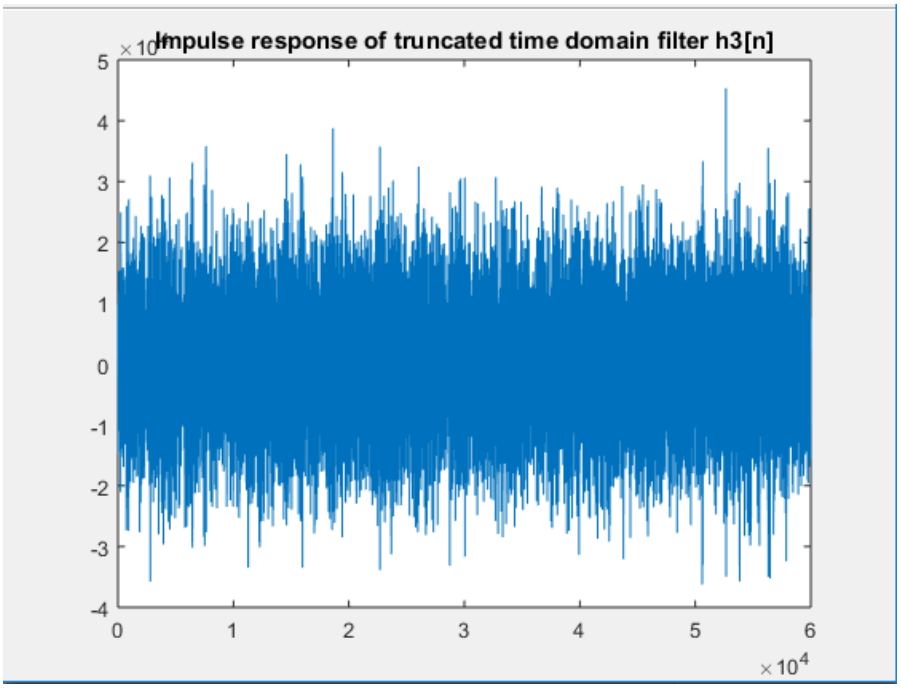

Figure 4: Impulse response of truncated time domain filter h3[n]

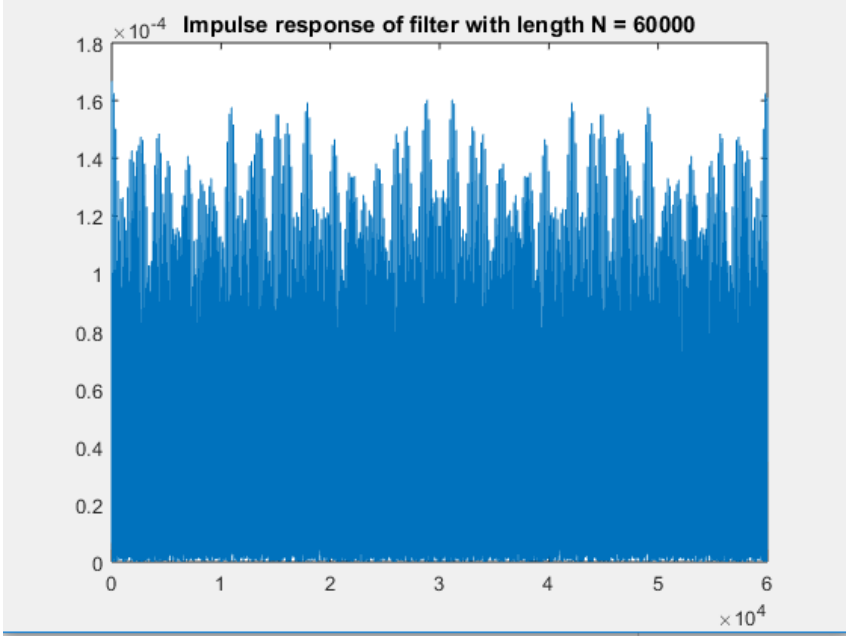

Figure 5: Impulse response of filter with length N=60000

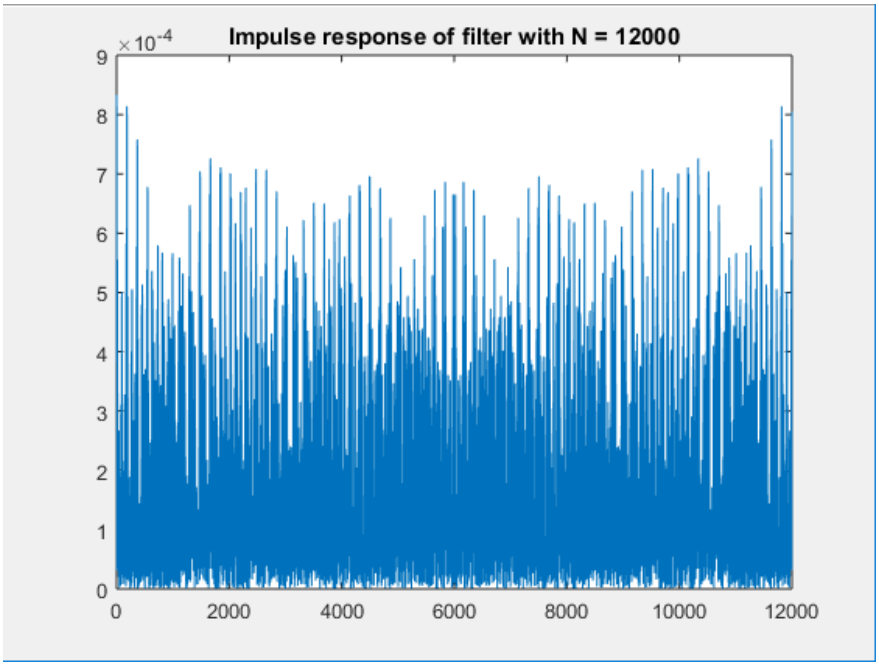

Figure 6: Impulse response of filter with N=12000

6549	6322	5661	4622	3297	1798	252	-1216	-2489	-3473

Figure 7: Coefficients of h3[n] with N=256