Cory Pisano, Minh Pham

DSP Equalizer Project

The purpose of this project was to further our understanding of FIR filtering by implementing a multiband equalizer. Our program is specifically split into 3 MATLAB files: main_program_gui.m, main_program_gui.fig, and equalizer.m.

As the name implies, main_program_gui.m and its accompanying .fig file contains the code for the program's GUI, whereas equalizer.m contains the signal processing algorithm. The program GUI is designed to be extra functional, and is partitioned into 3 functions: Amplification, File, and Status. For the most part, the program is designed such that only the button callbacks do the work. There are 3 global variables which store the file name and path, processed audio, and the player.

The Status section contains a file name and path listing, File Status, and Save Status. File Status allows one to see if processing has been applied to the indicated file, or whether the currently shown gains are applied to the file. The Save Status indicates if the file has been saved yet. All of these elements are static text elements that are modified by various button and slider callbacks. The File Status and Save Status indicators are designed to be invisible until a file is loaded.

The Amplification section contains 8 sliders that allows one to vary the gain for 8 frequency bands (150Hz, 300Hz, 600Hz, 1.2KHz, 2.4Khz, 4.8KHz, 9.2KHz, and 18.4KHz), from -10dB to 10 dB. This is the only place in which a non-button callback does anything. Each slider callback modifies the corresponding static text element below it to indicate the current gain level, in addition to updating the File Status to warn the user that the currently shown gains have not yet been applied to the sound.

Contained within the Amplification section is the Gain Reset subsection, which allows one to reset either individual gains or all gains back to 0dB. The ResetButton callback checks all the checkboxes for their value. For each checkbox with a nonzero value, the ResetButton callback will set the slider value to 0, update the text label, and uncheck the checkbox. The ResetAll button callback does the same thing, except for all of the checkboxes and and sliders. Both will update the File Status to warn the user that the new gains have not yet been applied to the sound.

The file section contains 5 buttons: Load, Process, Play, Stop, and Save. The program uses the audioplayer object to play sounds, instead of the sound command because audioplayer allows one to stop the music, which is rather convenient for longer song files. The load_button callback gets the file and path using uigetfile, and concatenates the two into a global loadedfile variable. It also makes the statuses under the Status section visible once a file is loaded.

The process_button callback gets gains from each slider, creates the input signal using wavread, and passes both the input signal and a vector of gains to equalizer.m. Once the equalizer is done processing, the File and Save indicators are updated. Play and Stop use the play and stop methods to control playback of the sound. It should be noted that Play will only play what is returned from equalizer.m. Therefore, it is important to apply 0 gain before playing the original loaded song.

The final button is the Save button. The save_button callback uses uiputfile to get a file path and name, which is used by wavwrite to write the processed audio. The callback will then update the Save status in the Status section of the program.

The true workhorse of the entire program is equalizer.m, which processes the audio. It is a

MATLAB function that takes the audio and gains as vector arguments, and returns a vector that contains the processed audio. The filter had an order M=512, which means the filter had a length of M+1 = 513. The first portion of the program converts the inputted gain vector from decibels to a gain factor. Given that gain in dB is 20log(A), A is given as $10^{(gain/20)}$. A 0 is also added as the 9th element of the gain vector, the reason for which will be explained when the actual filter itself is examined. The final part of the setup is to create the vector of frequency cutoffs, which are all scaled by the sampling frequency.

$$h_{\mathrm{mb}}[n] = \sum_{k=1}^{N_{\mathrm{mb}}} (G_k - G_{k+1}) \frac{\sin \omega_k (n - M/2)}{\pi (n - M/2)}, \qquad (7.81)$$

The above equation[1] is used to implement the multiband filter. This allows one to use one set of Kaiser window parameters with the same behavior at all discontinuities (band edges). G represents all of the gains. It should be noted that G_{k+1} exceeds the number of bands, which is why 0 was added as the 9th element to the gain vector. It should also be noted that it is possible to use a sinc function to implement the portion of the equation being multiplied by G_k-G_{k+1}. This is important, as using a sine function would result in division by zero.

The filter implemented in our program was split into 2 parts: the gain part, and the sinc (band) part. The band portion was implemented first, which was then multiplied according by the gain difference portion. The result is the impulse response of the filter, h_d.

The next task at hand was to window this new filter. It was discovered that a beta of 0 gave the best filter performance. A smaller beta, given all else is fixed, typically means a smaller main lobe. This means that the transition period between the pass and stop bands is tighter. We found this most important for this application, due to the smaller, closer frequencies used in the first 3 or so bands. Recall that all of the discontinuities share the same Kaiser window characteristics. If beta were too high, the transition periods would be too large, causing the bands to "blend" into each other, greatly reducing the effectiveness of the equalizer. With the window created, completing the filter was a simple matter of multiplying h_d by the Kaiser window.

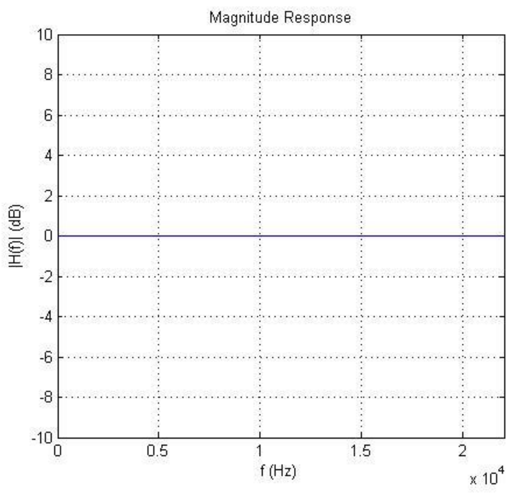
To understand this filter, its frequency and impulse response are plotted. To plot the magnitude response, the discrete Fourier transform of the filter was taken. The negative values of the filter were discarded, and the values were converted to decibels then plotted against frequency. For the impulse response, the filter was plotted against n (which ranges from 0 to M).

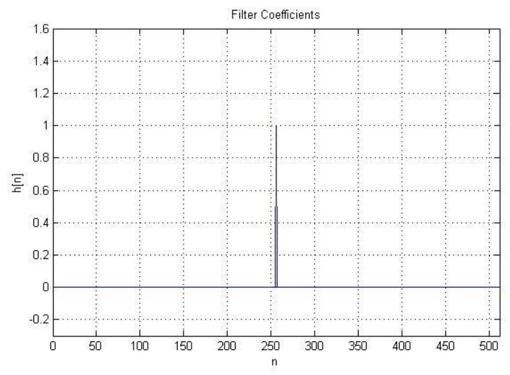
Finally, the input was filtered. This is a simple matter of convolving the filter with the input. The output was then passed back to the calling function.

The program was tested in several manners. First, the magnitude and impulse response were found for several sets of gains:

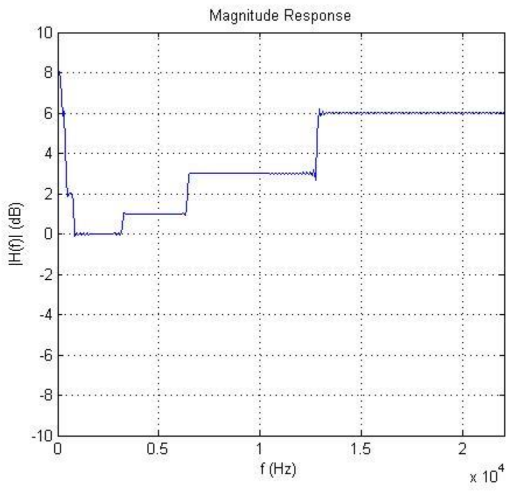
1 *Discrete-Time Signal Processing,* Oppenheim, p550

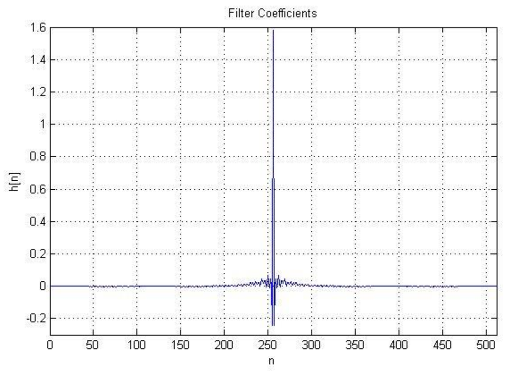
(a) g = [0 0 0 0 0 0 0 0]



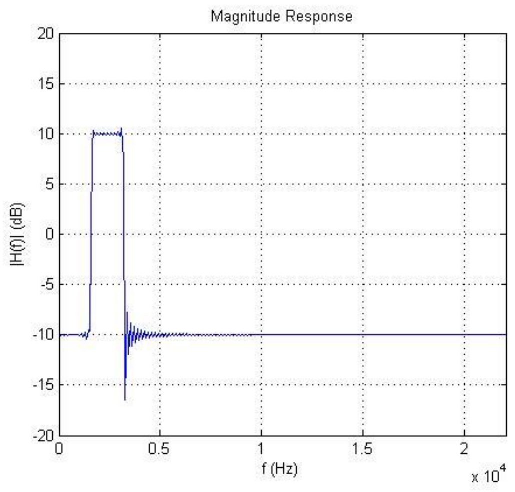


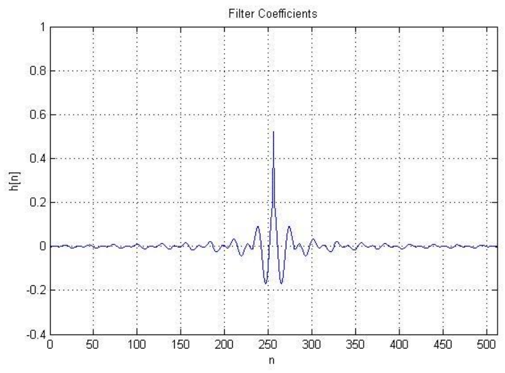
(b) g = [8 6 2 0 0 1 3 6]



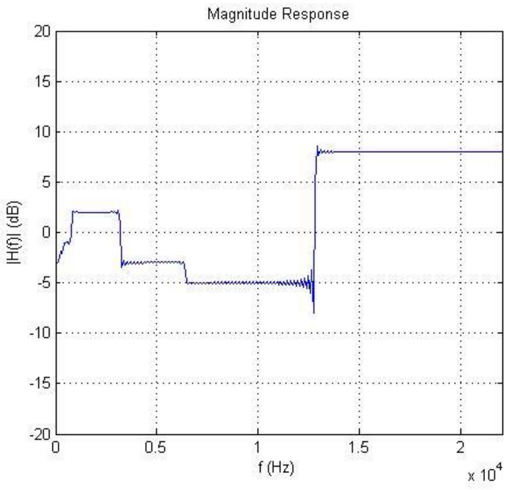


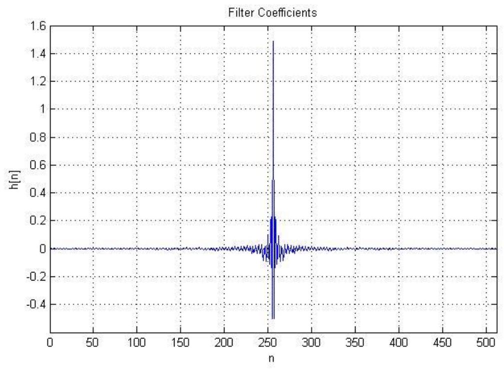
(c) g = [-10 -10 -10 -10 10 -10 -10 -10]





(d) g = [-3 -2 -1 2 2 -3 -5 8]



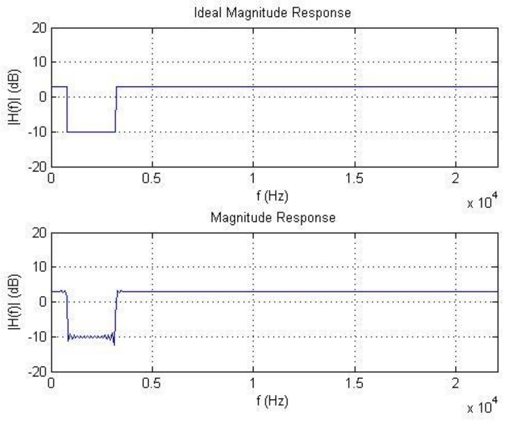


As shown in the above magnitude responses, the transition periods are very small, short, and sharp, as desired. However, there are occasionally ripples between the bands, such as that between bands 7 and 8 in part d, or between bands 5 and 6 in part c.

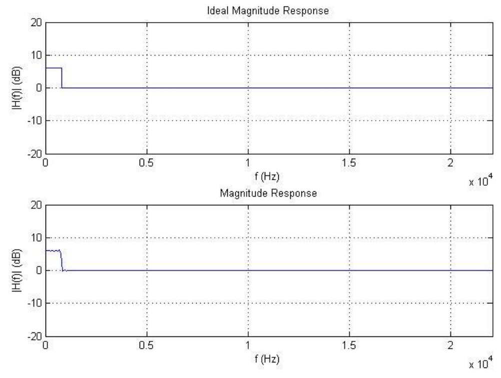
The final part of testing involved using the equalizer on songs. We found 2 songs high suitable for testing the equalizer: *Madness* and *Panic Station*, both songs by the group Muse. *Madness* is a particularly bass heavy song, while *Panic Station* is a bit more balanced, with a good amount of mid to high frequency sound. Both files are 44.1KHz PCM monaural WAV files obtained by ripping directly from a CD. This is to ensure maximum sound quality and compatibility, as MATLAB's wavread function can be particular about the wav file being inputted.

The magnitude response of the filter for each test sample was plotted, along with the ideal magnitude response (on top).

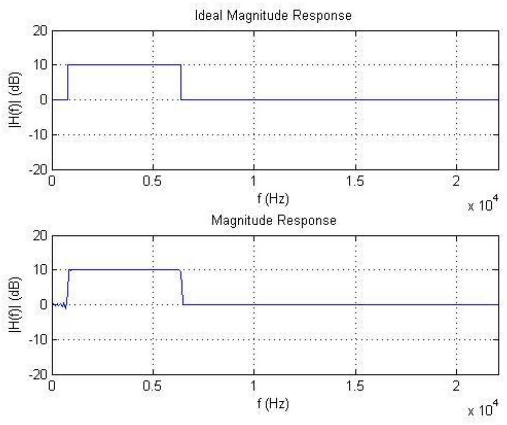
Madness, with low mids:



Panic Station, with high bass:



Panic Station, with high mids:



As shown above, the filter does an excellent job, providing a magnitude response that is close to ideal. The worst of the above appears to be *Madness* with low mids, which has ripples between the stop and pass bands. Nevertheless, its performance is admirable.

The audio of the processed songs tended to be fine, although over increasing the bass on certain songs would result in clipping. Saving the processed audio file allowed one to play the song after it was processed. While MATLAB warns about clipping when saving the file, there is no discernible difference between playing the saved file, and playing the processed sound in the program itself, nor are there any noticeable quality issues.

The necessary project files are available directly on EasyChair, but due to the size restrictions a separate zip file is hosted at the following link:

https://docs.google.com/file/d/0B9K_xZDaYdbNcm1BY3N3WWJvTnM/edit?usp=sharing

This zip file contains all the implementation files, instructions, as well as the test input and output files used. The zip file on EasyChair contains only the implementation files and instructions.