NORWICH UNIVERSITY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

EE 487 DIGITAL SIGNAL PROCESSING

Spring 2014

Laboratory Report

Laboratory Experiment 7

Design of a 10-Band Digital Equalizer

Submitted by:

Daniel Alexander Rodriguez

Email:

drodrigu@stu.norwich.edu

Date:

9 April, 2014

Abstract

This laboratory exercise involves the creation of a 10-channel equalizer primarily by using the butterworth filter approximation method. The mathematics modeling program MATLAB was used to model the equalizer. A stereo sound file was input into the program, where 1 low-pass filter, 1 high-pass filter, and 9 band-pass filters divided the stereo sound signal into 10 different frequency channels. Each filtered channel was then adjustable including a dB gain variable to change the gain for each respective channel, and then the results were recombined into two stereo channels. The stereo channels were then saved to a new stereo sound file. The results were plotted for the dB plot and time signal plot for before and after the dB gain values were adjusted.

Introduction

The 10-channel equalizer filter for this laboratory exercise can be modeled using basic principles of low-pass, band-pass, and high-pass filters. The butterworth approximation method is used to filter these 10 individual channels, and was accomplished using the mathematics modeling program MATLAB.

Apparatus and Procedure

The mathematics modeling program MATLAB was used to design the stereo audio equalizer in this laboratory exercise.

The equalizer was modelled by first separating the input stereo sound file into 10 frequency channels using a low-pass filter for the first channel, a band-pass filter for up to the ninth channel, and a high-pass filter for the tenth channel. Each filter was modelled using the MATLAB scripts **`butter.m`** and **`filter.m`**.

The stereo sound file was read into the program using the following code:

```
[stereochirp, Fs, nbits] = wavread('stereo.WAV');
xt = stereochirp;
```

Note that the script **`wavread.m`** stored the amplitude of the signal, as well as the sampling rate and number of bits.

The script **butter.m** was used to generate a transfer function for one of the filters using the butterworth approximation technique, given the order and cutoff frequency parameters, and **filter.m** used the generated transfer function and applied it to the stereo sound file. The following MATLAB code shown shows these scripts in the final program.

```
[Hznum_32, Hzden_32] = butter(3,64/(Fs/2), 'low');
bp_32 = filter(Hznum_32,Hzden_32,xt);
```

Notice that this butterworth script uses order 3 and a cutoff frequency of 64, which is scaled between 0 and 1 by the folding frequency of sampling rate. Scaling the cutoff frequency for this function is necessary for the scripts operation. The filter script then takes the numerator `Hznum_32` and denominator `Hzden_32` of the butterworth script's output array and applies it to the input stereo sound file `xt`. For a high-pass butterworth filter, the procedure for the low-pass butterworth filter from before are exactly the same; however, **butter.m** must contain `'high'` instead of `'low'` when specifying the type of filter in the script.

For a band-pass butterworth filter, the parameters of the **butter.m** must be adjusted for the starting and stopping frequencies of the passband, as shown:

```
[Hznum_64, Hzden_64] = butter(2,[64 125]/(Fs/2), 'bandpass');
bp_64 = filter(Hznum_64,Hzden_64,xt);
```

This code is used to pass the signal between 64 Hz and 125 Hz. Similar to the low-pass filter script, the starting and stopping frequencies of this band-pass filter must be scaled by the folding frequency of the sampling rate to scale both frequencies between 0 and 1.

The gain for each channel was then calculated by applying the fast Fourier transform (FFT) on all 10 filtered channels individually, and then converting the FFTs' amplitudes to dB gain values as shown (for the tenth channel):

```
bp_16000_fft = abs(fft(bp_16000));
bp_16000_dB = 20*log10(bp_16000_fft);
```

Following this, each channel was separated as shown:

```
bp_32_l = bp_32(:,1);
bp_32_r = bp_32(:,2);
```

This code splits the left and right channels of the 32 Hz to 64 Hz filtered channel. This process was repeated for every other filtered channel.

The gain control was done by setting a variable equal to the desired gain adjustment, where the default is 0 dB for no gain change. Then, the dB value is converted back into the original amplitude unit for the stereo sounds signal so it can be scaled correctly. The following code shows this unit conversion and scaling method.

```
a_32_dB_l = 0;
a_32_l = 10^(a_32_dB_l/20);
```

Note that this code is used for only the left channel of the stereo sound signal. This process was repeated for every other filtered channel for both the left and right audio channels of the sound file.

The signal was restored by adding all 10 of the filtered channels together after adjusting the gain to the desired level. The variable calculated by converting the dB value back into the unit used by the stereo sound signal was used to scale each filtered channel by multiplying each respective one together. See the code below for the channel addition:

```
bp_l =   a_32_l*bp_32_l + a_64_l*bp_64_l + a_125_l*bp_125_l ...
+ a_125_l*bp_125_l + a_250_l*bp_250_l + a_500_l*bp_500_l ...
+ a_1000_l*bp_1000_l + a_2000_l*bp_2000_l + a_4000_l*bp_4000_l ...
+ a_8000_l*bp_8000_l + a_16000_l*bp_16000_l;
```

Note that the above code is used for the left channel of the stereo sound signal only. This process was repeated for the right channel as well.

Both the left and right channels of the stereo sound signal were combined into an array to recreate the stereo data file:

```
bp_stereo = [bp_l bp_r];
```

Finally, the resulting stereo signal was written as a WAV file using **`wavread.m`**:

```
wavwrite(bp_stereo, Fs, 'bp_stereo.WAV');
```

Analysis and Results

See **Figure 1** for the plot of the 10-channel equalizer designed using MATLAB in this laboratory exercise.

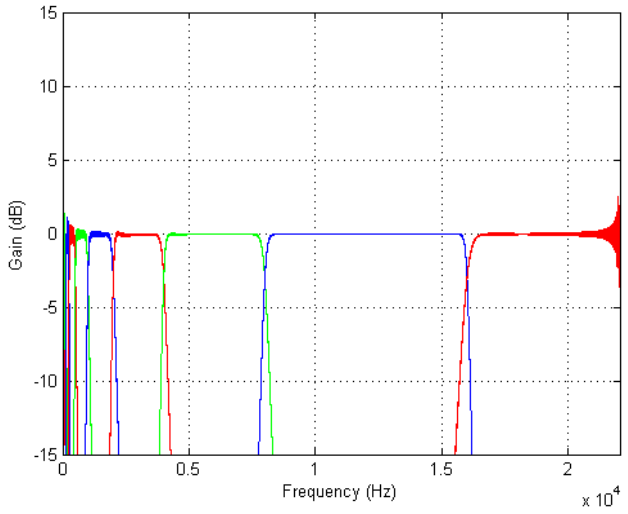

Figure 1. This is the plot of the 10-channel equalizer design in this laboratory exercise. The gain is unchanged at a gain of 0 dB. Notice each channel is represented by a different color in consequent order. Notice the gaps between the wide channels toward the high-frequency end and distortion of the narrower channels toward the low-frequency end.

The gaps between each passband channel are caused by the placement of the cutoff, starting, and stopping frequencies. The noise at the low frequency end of the plot is caused by the butterworth filter's approximation tendency to crossover signals within a small passband area. See **Figure 2** for a better view of the low frequency region of this plot.

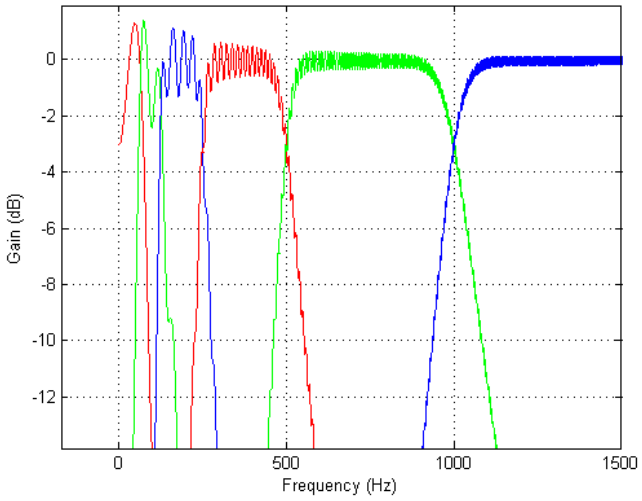

Figure 2. This plot provides a better look at the lower frequencies of the 10-band equalizer plot in **Figure 1**. Notice the noise at the low frequency channels. These sinusoid spikes are caused by the nature of the butterworth approximation function. The shorter the passband of the filter channel, the more sinusoid noise there is for the passband.

The effects from the noise and gaps between the channels of the filter on the output time signal can be seen in **Figure 3**.

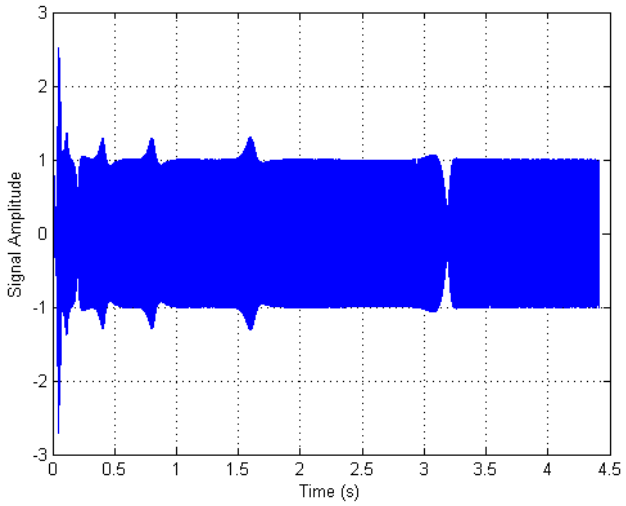

Figure 3. This is the time plot of the stereo sound signal after being filtered by the 10-channel equalizer from **Figure 1**. Notice the spikes and gaps. These are caused by the nature of the butterworth filter and placement of the cutoff frequencies.

These are caused by the noise of the low-frequency butterworth and the gaps or overlaps that occur from channel to channel. However, the signal is still pretty consistent throughout playback. Note that the parameters can be adjusted to fix these signal spikes, however, the cutoff frequencies were selected in accordance for the desired passband length for consistency.
The dB gain of the filter was adjusted using the MATLAB code for the program. The following code was used to adjust the dB gain of the 8,000 Hz to 16,000 Hz left channel passband to 10 dB:

```
a_8000_dB_l = 10;
a_8000_l = 10^(a_8000_dB_l/20);
```

Notice in **Figure 4** that the 8,000 Hz to 16,000 Hz channel passband raises from 0 dB (as shown in **Figure 1**) to 10 dB as expected. Note that the 4,000 Hz to 8,000 Hz and 16,000 Hz to 20,000 Hz channel passbands were also adjusted to 5 dB and 13 dB of gain, respectively.

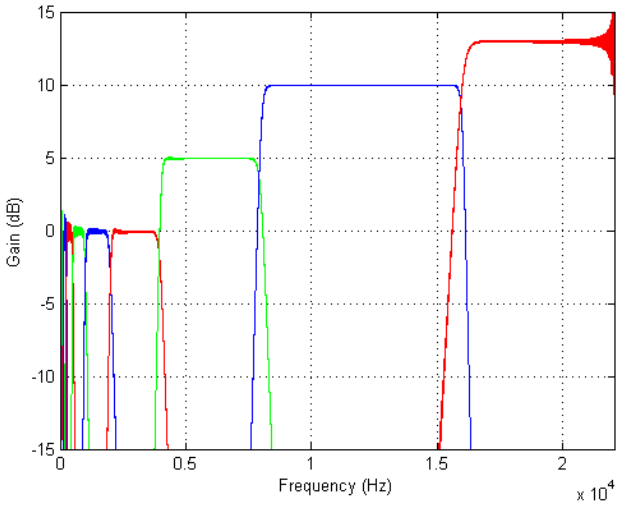

Figure 4. This is the plot of the 10-channel equalizer for the stereo sound signal, with the dB gains for the last three channels adjusted to 5 dB, 10 dB, and 13 dB, respectively in consequent order from left to right.

Figure 5 shows the effect of adjusting the dB gain of the 10-channel equalizer shown in **Figure 4**.

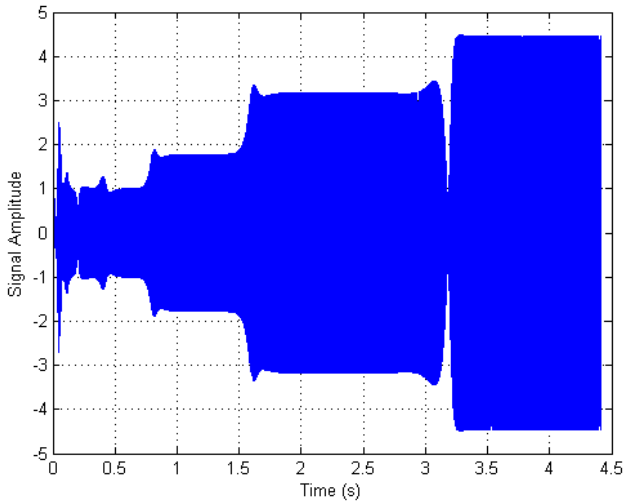

Figure 5. This is the time plot of the stereo sound signal after being filtered by the 10-channel equalizer from **Figure 4**. The effect of adjusting the dB gain at the high frequency end of the equalizer had boosted the high frequency noise from the original stereo sound signal as expected. By design, the input sound file increases in frequency as it progresses during playback, therefore providing evidence for the dramatic boost on the right-side of the time plot. The step response for the right end of the plot also matches the proportionality of the adjusted 10-channel equalizer plot in **Figure 4**.

Conclusion

This filter was designed using basic principles of low-pass, band-pass, and high-pass filters. The butterworth approximation method was used to find a transfer function that would appropriately filter 10 individual channels of the input stereo sound signal, and MATLAB was used to execute this function. The dB gain was adjusted after separating each channel into separate stereo components, and then the channels were recombined per left and right stereo components and the new filtered stereo sound signal was written. However, the output was not perfect because there were signs of noise and gaps in the playback. The signal was plotted and showed cases of sinusoidal noise caused by the butterworth approximation for short passband. Also evident were the spikes and gaps in the playback, which were caused by the placement of the channel passband edges. Regardless of these errors, the output of the filter still successfully adjusted the dB gain of 10 separate frequency range channels.

Appendices

APPENDIX A

The following code was used to model this entire 10-channel equalizer. The filter types are separated and organized according to the passband range. The channel splitting follows the process of separating the input signal into 10 channels. The gain control then succeeds the channels splitting for each channel for both stereo components. Finally, all of the components are recombined into a stereo signal, plotted, and then written into a stereo sound file for future audio playback.

```
clc;
clear all;
close all;

[stereochirp, Fs, nbits] = wavread('stereo.WAV');
xt = stereochirp;

Ts = 1/Fs;
N = length(stereochirp);
df = Fs/N;
f = df*[0:1:N-1];

%%%%---- LOWPASS ----%%%%

% 32 - 64 Hertz

[Hznum_32, Hzden_32] = butter(3,64/(Fs/2), 'low');
bp_32 = filter(Hznum_32,Hzden_32,xt);
bp_32_fft = abs(fft(bp_32));
bp_32_dB = 20*log10(bp_32_fft);

%%%%---- BANDPASS ----%%%%

% 64 - 125 Hertz
[Hznum_64, Hzden_64] = butter(2,[64 125]/(Fs/2), 'bandpass');
bp_64 = filter(Hznum_64,Hzden_64,xt);
bp_64_fft = abs(fft(bp_64));
bp_64_dB = 20*log10(bp_64_fft);

% 125 - 250 Hertz
[Hznum_125, Hzden_125] = butter(4,[125 250]/(Fs/2), 'bandpass');
bp_125 = filter(Hznum_125,Hzden_125,xt);
bp_125_fft = abs(fft(bp_125));
bp_125_dB = 20*log10(bp_125_fft);

% 250 - 500 Hertz
[Hznum_250, Hzden_250] = butter(4,[250 500]/(Fs/2), 'bandpass');
bp_250 = filter(Hznum_250,Hzden_250,xt);
bp_250_fft = abs(fft(bp_250));
bp_250_dB = 20*log10(bp_250_fft);
```

```
% 500 - 1000 Hertz
[Hznum_500, Hzden_500] = butter(5,[500 1000]/(Fs/2), 'bandpass');
bp_500 = filter(Hznum_500,Hzden_500,xt);
bp_500_fft = abs(fft(bp_500));
bp_500_dB = 20*log10(bp_500_fft);

% 1000 - 2000 Hertz
[Hznum_1000, Hzden_1000] = butter(6,[1000 2000]/(Fs/2), 'bandpass');
bp_1000 = filter(Hznum_1000,Hzden_1000,xt);
bp_1000_fft = abs(fft(bp_1000));
bp_1000_dB = 20*log10(bp_1000_fft);

% 2000 - 4000 Hertz
[Hznum_2000, Hzden_2000] = butter(9,[2000 4000]/(Fs/2), 'bandpass');
bp_2000 = filter(Hznum_2000,Hzden_2000,xt);
bp_2000_fft = abs(fft(bp_2000));
bp_2000_dB = 20*log10(bp_2000_fft);

% 4000 - 8000 Hertz
[Hznum_4000, Hzden_4000] = butter(14,[4000 8000]/(Fs/2), 'bandpass');
bp_4000 = filter(Hznum_4000,Hzden_4000,xt);
bp_4000_fft = abs(fft(bp_4000));
bp_4000_dB = 20*log10(bp_4000_fft);

% 8000 - 16000 Hertz
[Hznum_8000, Hzden_8000] = butter(25,[8000 16000]/(Fs/2), 'bandpass');
bp_8000 = filter(Hznum_8000,Hzden_8000,xt);
bp_8000_fft = abs(fft(bp_8000));
bp_8000_dB = 20*log10(bp_8000_fft);

%%%%---- HIGHPASS ----%%%%

% 16000 - 20000 Hertz
[Hznum_16000, Hzden_16000] = butter(20,16000/(Fs/2), 'high');
bp_16000 = filter(Hznum_16000,Hzden_16000,xt);
bp_16000_fft = abs(fft(bp_16000));

bp_16000_dB = 20*log10(bp_16000_fft);

%%%%---- CHANNEL SPLITTING ----%%%%

bp_32_l = bp_32(:,1);
bp_32_r = bp_32(:,2);

bp_64_l = bp_64(:,1);
bp_64_r = bp_64(:,2);

bp_125_l = bp_125(:,1);
bp_125_r = bp_125(:,2);

bp_250_l = bp_250(:,1);
bp_250_r = bp_250(:,2);

bp_500_l = bp_500(:,1);
```

```
bp_500_r = bp_500(:,2);

bp_1000_l = bp_1000(:,1);
bp_1000_r = bp_1000(:,2);

bp_2000_l = bp_2000(:,1);
bp_2000_r = bp_2000(:,2);

bp_4000_l = bp_4000(:,1);
bp_4000_r = bp_4000(:,2);

bp_8000_l = bp_8000(:,1);
bp_8000_r = bp_8000(:,2);

bp_16000_l = bp_16000(:,1);
bp_16000_r = bp_16000(:,2);


%%%%---- GAIN CONTROL ----%%%%

%Left Channel Gain
a_32_dB_l = 0;
a_32_l = 10^(a_32_dB_l/20);

a_64_dB_l = 0;
a_64_l = 10^(a_64_dB_l/20);

a_125_dB_l = 0;
a_125_l = 10^(a_125_dB_l/20);

a_250_dB_l = 0;
a_250_l = 10^(a_250_dB_l/20);

a_500_dB_l = 0;
a_500_l = 10^(a_500_dB_l/20);

a_1000_dB_l = 0;
a_1000_l = 10^(a_1000_dB_l/20);

a_2000_dB_l = 0;
a_2000_l = 10^(a_2000_dB_l/20);

a_4000_dB_l = 0;
a_4000_l = 10^(a_4000_dB_l/20);

a_8000_dB_l = 0;
a_8000_l = 10^(a_8000_dB_l/20);

a_16000_dB_l = 0;
a_16000_l = 10^(a_16000_dB_l/20);

% Right Channel Gain
a_32_dB_r = 0;
```

```
a_32_r = 10^(a_32_dB_r/20);

a_64_dB_r = 0;
a_64_r = 10^(a_64_dB_r/20);

a_125_dB_r = 0;
a_125_r = 10^(a_125_dB_r/20);

a_250_dB_r = 0;
a_250_r = 10^(a_250_dB_r/20);

a_500_dB_r = 0;
a_500_r = 10^(a_500_dB_r/20);

a_1000_dB_r = 0;
a_1000_r = 10^(a_1000_dB_r/20);

a_2000_dB_r = 0;
a_2000_r = 10^(a_2000_dB_r/20);

a_4000_dB_r = 0;
a_4000_r = 10^(a_4000_dB_r/20);

a_8000_dB_r = 0;
a_8000_r = 10^(a_8000_dB_r/20);

a_16000_dB_r = 0;
a_16000_r = 10^(a_16000_dB_r/20);


%%%%---- SIGNAL ADDITION ----%%%%

bp_l =   a_32_l*bp_32_l + a_64_l*bp_64_l + a_125_l*bp_125_l ...
       + a_125_l*bp_125_l + a_250_l*bp_250_l + a_500_l*bp_500_l ...
       + a_1000_l*bp_1000_l + a_2000_l*bp_2000_l + a_4000_l*bp_4000_l ...
       + a_8000_l*bp_8000_l + a_16000_l*bp_16000_l;

bp_r =   a_32_r*bp_32_r + a_64_r*bp_64_r + a_125_r*bp_125_r ...
       + a_125_r*bp_125_r + a_250_r*bp_250_r + a_500_r*bp_500_r ...
       + a_1000_r*bp_1000_r + a_2000_r*bp_2000_r + a_4000_r*bp_4000_r ...
       + a_8000_r*bp_8000_r + a_16000_r*bp_16000_r;

plot(f,a_32_dB_l + bp_32_dB - 49.5,'r')
hold on
plot(f,a_64_dB_l + bp_64_dB - 49.5,'g')
plot(f,a_125_dB_l + bp_125_dB - 49.5,'b')
plot(f,a_250_dB_l + bp_250_dB - 49.5,'r')
plot(f,a_500_dB_l + bp_500_dB - 49.5,'g')
plot(f,a_1000_dB_l + bp_1000_dB - 49.5,'b')
plot(f,a_2000_dB_l + bp_2000_dB - 49.5,'r')
plot(f,a_4000_dB_l + bp_4000_dB - 49.5,'g')
plot(f,a_8000_dB_l + bp_8000_dB - 49.5,'b')
plot(f,a_16000_dB_l + bp_16000_dB - 49.5,'r')
grid on
```

```
ylim([-15 15])
xlim([0 Fs/2])
xlabel('Frequency (Hz)')
ylabel('Gain (dB)')

bp_stereo = [bp_l bp_r];

t = f/(10^4);

figure(2)
plot(t, bp_l)
grid on
xlabel('Time (s)')
ylabel('Signal Amplitude')

wavwrite(bp_stereo, Fs, 'bp_stereo.WAV');
```