105062206 蔡哲維

Multi Media HW2

1.

第一題我們首先要來實做三個 FIR filter。

進入 myfilter.m 的 framework 之中觀察看看要怎麼做。

```
function [outputSignal, outputFilter] = myFilter(inputSignal, fsample, N, windowName, filterName, fcutoff)
%%% Input
% inputSignal: input signal
% fsample: sampling frequency
% N : size of FIR filter(odd)
% windowName: 'Blackmann'
% filterName: 'low-pass', 'high-pass', 'bandpass', 'bandstop'
% fcutoff: cut-off frequency or band frequencies
%       if type is 'low-pass' or 'high-pass', para has only one element
%       if type is 'bandpass' or 'bandstop', para is a vector of 2 elements

%%% Ouput
% outputSignal: output (filtered) signal
% outputFilter: output filter
```

可以看到傳進來的參數有要處理的訊號、sampling frequency、window 的大小、 window 的名字、filter 的名字、還有要在哪個頻率 cut 掉。 首先先做 normalization，讓 sampling frequency 落在 2pi 的位置，並讓 fcutoff 跑 到其對應的值上。同時取 middle 值為 window 的大小除以 2 取 floor，方便之後計算。

```
%% 1. Normalization
fcutoff = fcutoff/fsample;
middle = floor(N/2);
```

接著根據不同的 filter 名稱進行處理。算出 ideal filter 的值。

```
if strcmp(filterName,"low-pass") == 1
    for n = (-middle):(middle)
        if n==0
            outputFilter(middle+n+1) = 2*fcutoff;
        else
            outputFilter(middle+n+1) = sin(2*pi*fcutoff*n)/(pi*n);
        end
    end

elseif strcmp(filterName,"high-pass") == 1
    for n = (-middle):(middle)
        if n==0
            outputFilter(middle+n+1) = 1-2*fcutoff;
        else
            outputFilter(middle+n+1) = -sin(2*pi*fcutoff*n)/(pi*n);
        end
    end
```

```
elseif strcmp(filterName,"bandpass") == 1
    for n = (-middle):(middle)
        if n==0
            outputFilter(middle+n+1) = 2*(fcutoff(2)-fcutoff(1));
        else
            outputFilter(middle+n+1) = (sin(2*pi*fcutoff(2)*double(n))-sin(2*pi*fcutoff(1)*double(n)))/(pi*double(n));
        end
    end
```

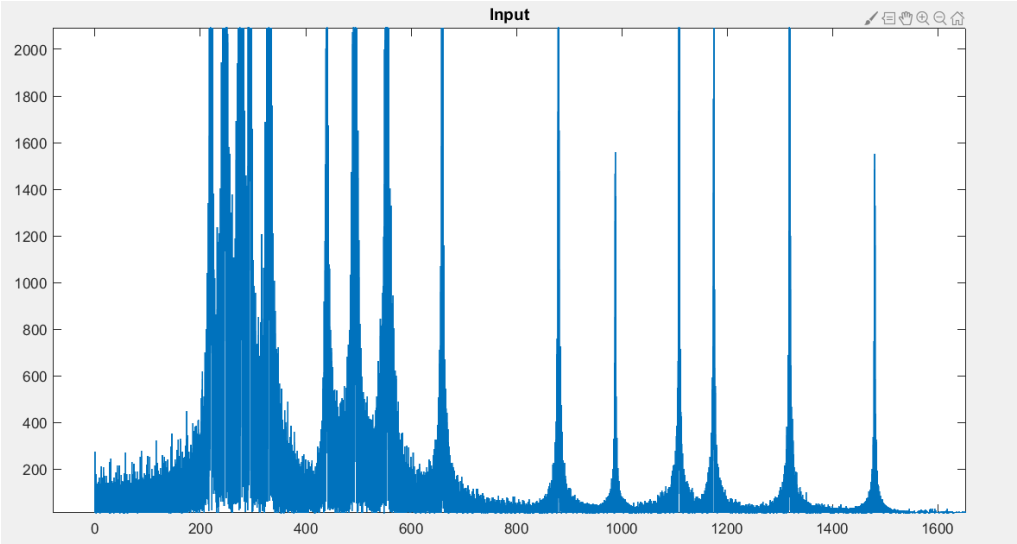
接著將 ideal 的 filter 去乘上 window function，在這題中我們只會用 Blackman window function。我做的範圍是 0~N-1，如助教在討論區所說的，由於相位的關 係，將 Blackman window function 的方程式改成 w(n) = 0.42 - 0.5*cos(2*pi*n / (N1)) + 0.08*cos(4*pi*n / (N-1))。

```
if strcmp(windowName, 'Blackman') == 1
    for n = 0:N-1
        outputFilter(n+1) = outputFilter(n+1)*(0.42-0.5*cos((2*pi*n)/(N-1)) + 0.08*cos((4*pi*n)/(N-1)));
    end
else
    disp("Not Blackman Window");
end
```

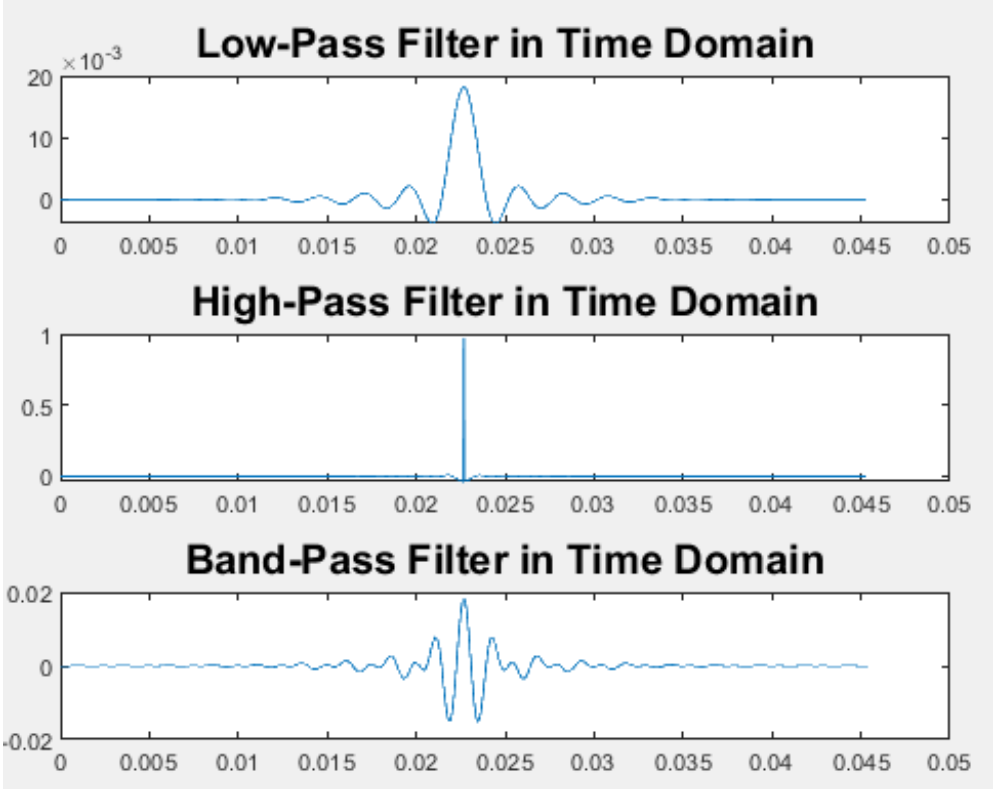
最後對於傳進來要處理的音訊，將其和 filter 做 convolution。
若 n-k 小於 0，則加上 0，若 n-k 大於 0，則去加上對應的 sample 點的值(index 為 n-k)乘上相應的 filter 值（因為 Matlab 沒有 index 0，故 index 的部分都要 +1）。每一個原始訊號的 sample 點，都去做 N 次，k 從 0~N-1。

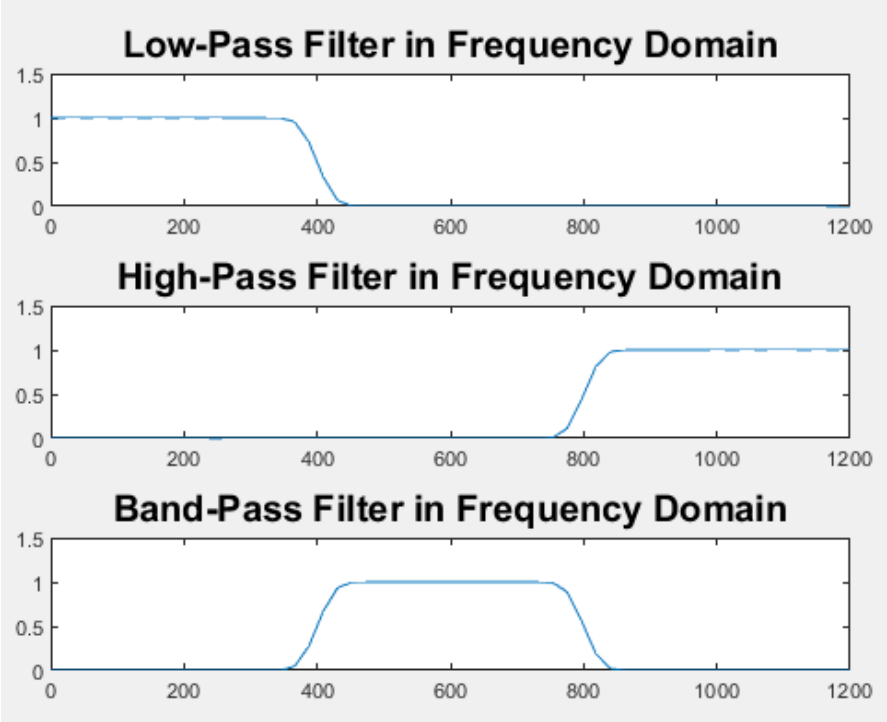
```
%% 4. Filter the input signal in time domain. Do not use matlab function 'conv'
outputSignal = zeros( length(inputSignal) , 1);
for n = 0:length(inputSignal)-1
    for k = 0:N-1
        if (n-k) < 0
            outputSignal(n+1) = outputSignal(n+1) + 0;
        else
            outputSignal(n+1) = outputSignal(n+1) + outputFilter(k+1)*inputSignal(n-k+1);
        end
    end
end
```

寫好 myfilter.m 之後，只需要在主 script 中針對不同要取的訊號呼叫該 function 即可。 因為原始音檔是 mix，有三個音訊混在一起。在呼叫 myfilter.m 之前，先觀察原始音檔的頻譜。



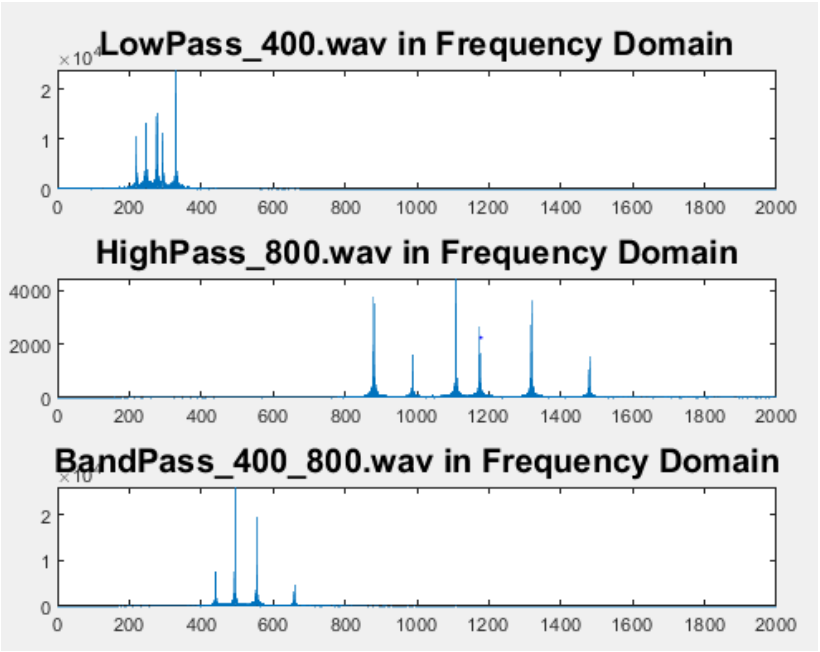
可以觀察到主要可以用 400HZ 和 800HZ 的區間將三個音訊分開。故在主 Script 中去呼叫 myfilter.m。(用了 lowpass、highpass、bandpass 三種 filter)。





可以觀察到，當我們將 ideal impulse response 乘上 window function，使其變成 finite 的時候，在 frequency domain 可以看到，因為 window 的大小有限，原本 cut off frequency 的邊界應該是垂直，現在變成斜的，有 ripple 的情況發生。

接著觀察經過這三種 filter 跑出來的音訊，可以看到原本三種音訊混雜的 mix 檔，經過不同 filter 處理後，分出了三種音訊，有了三種 frequency spectrum。且是依據 lowpass、highpass、bandpass，以及 cut off frequency 來區分的。



做完 filtering 之後，題目接著要求我們去讓這三個音訊的 sampling rate 降為 2000HZ。這裡使用 MATLAB 內建的函式去做 resampling。

```
%% 4, Reduce the sample rate of the three separated songs to 2kHz.
[P,Q] = rat(2000/fs);
resample_outputSignal_lowpass = resample(outputSignal_lowpass, P, Q);
resample_outputSignal_highpass = resample(outputSignal_highpass, P, Q);
resample_outputSignal_bandpass = resample(outputSignal_bandpass, P, Q);
```

比較了 resampling 前後的音訊。在 lowpass(400)、bandpass(400~800)的時候，其 實具體聽不出甚麼差距。但在聽 highpass(800)的時候，差距就很明顯了，resampling 過後的音訊有些音直接消失或者頻率改變。我想這是因為高於 1000HZ 的音，無法在 sample rate 是 2000HZ 的情況下還原出來（根據奈奎斯定 理），有失真的情況發生，才會造成這個結果。

最後要針對通過 lowpass filter 的音訊去做 one fold 和 multiple fold echo 的處理。 這裡純粹就帶入講義 comb filter 的公式去做。

```
%% 5. one-fold echo and multiple-fold echo (slide #69)
one_fold_echo_outputSignal_lowpass = zeros(length(outputSignal_lowpass),1);
for n = 0:length(outputSignal_lowpass)-1
    if n - 3200 < 0
        one_fold_echo_outputSignal_lowpass(n+1) = outputSignal_lowpass(n+1) + 0;
    else
        one_fold_echo_outputSignal_lowpass(n+1) = outputSignal_lowpass(n+1) + 0.8* outputSignal_lowpass(n-3200+1);
    end
end
```

```
mutiple_fold_echo_outputSignal_lowpass = zeros(length(outputSignal_lowpass), 1);
for n = 0:length(outputSignal_lowpass)-1
    if n - 3200 < 0
        mutiple_fold_echo_outputSignal_lowpass(n+1) = outputSignal_lowpass(n+1) + 0;
    else
        mutiple_fold_echo_outputSignal_lowpass(n+1) = outputSignal_lowpass(n+1) + 0.8* mutiple_fold_echo_outputSignal_lowpass(n-3200+1);
    end
end
```

值得注意的是在輸出 multiple fold echo 的時候，音訊有些音的強度超過 audiowrite 許可的範圍，導致用 audiowrite 做會有 clipping 的情況發生，造成聲 音變形，故在輸出之前先去做 normalization。

```
%% 5. Save the echo audios 'Echo_one.wav' and 'Echo_multiple.wav'
audiowrite("Echo_one.wav", one_fold_echo_outputSignal_lowpass, fs);

%normalize the "mutiple_fold_echo_outputSignal_lowpass" for preventing
%clipping
mutiple_fold_echo_outputSignal_lowpass = mutiple_fold_echo_outputSignal_lowpass / max(abs(mutiple_fold_echo_outputSignal_lowpass));
audiowrite("Echo_multiple.wav", mutiple_fold_echo_outputSignal_lowpass, fs);
```

2.

第二題針對一個原始音檔為 16bit 表示的訊號，用 bit reduction 將其改成 8bit 表 示。

```
%% 2. Bit reduction
% (Hint) The input audio signal is double (-1 ~ 1)

bit_reduction_output = zeros(length(input), 2);
for j = 1:2
    for i = 1:length(input)
        bit_reduction_output(i,j) = round(128*input(i,j))*(1/128);
        if bit_reduction_output(i,j) == 1
            bit_reduction_output(i,j) = floor(128*input(i,j))*(1/128);
        end
        if bit_reduction_output(i,j) == 1
            bit_reduction_output(i,j) = floor(128*input(i,j)-0.1)*(1/128);
        end
    end
end
```

於是這裡先將原始音訊的每個 sample 點的值做 scale，乘上 128，去做 round， 再除以 128，取得用 8bit 可以表示的值。因為我這裡用 8bit 表示，scale 過後只 能表示-128~127，故 round 完除以 128 為 1 的數值都要改成 127/128。第一個 if 是去讓原始訊號的某個點的值乘上 128 做完 round 為 1 的值，改成用 floor 去算，會變成 127，之後再去除以 128。最後一個 if 是避免原始訊號某點的值有 1，如此乘上 128 無論去做 round 或是 floor 都還是 128，所以先讓她減掉 0.1，再去做 floor，會得到 127，最後再去做除以 128 的動作。

接著將這個音訊輸出，這裡用 audiowrite。因為 audiowrite 預設的 BitsPerSample 為 16bit，故在寫音檔的時候，需要去告訴她 BitsPerSample 為 8bit（雖然前面已經做好量化了，用 16bit 表示也會有相同結果，但為了統一性，這裡還是如此宣告）。

```
%%% Save audio (audiowrite) Tempest_8bit.wav
% (Hint) remember to save the file with bit = 8
audiowrite("Tempest_8bit.wav", bit_reduction_output, fs, "BitsPerSample", 8);
```

做完 16bit-8bit 的 reduction 之後，為了補償隨之而來的噪音，以及音樂有些時候會有一階一階的狀況，聽起來不連續，故要做 dithering。這裡用 rand 隨機產生 0~1 之間的值然後乘以 2 加-1，得到-1~1 之間 的值，再除以 128，然後加在做完 bit reduction 的訊號上。

```
%% 3. Audio dithering
% (Hint) add random noise
dithered_bit_reduction_output = zeros(length(input), 2);

for j = 1:2
    for i = 1:length(input)
        dithered_bit_reduction_output(i,j) = bit_reduction_output(i,j) +  (-1 + (1+1)*rand)/128;
    end
end
```

接著再拿這個訊號去做 noise shaping。

```
%% 4. First-order feedback loop for Noise shaping
% (Hint) Check the signal value. How do I quantize the dithered signal? maybe scale up first?
f_in = dithered_bit_reduction_output;
noise_shape_output = zeros(length(input),2);

c = 5;
for j = 1:2
    for i = 1:length(input)
        if i == 1
            e = 0;
        else
            e = f_in(i-1,j) - noise_shape_output(i-1,j);
        end
        f_in(i,j) = f_in(i,j) + c * e;
```

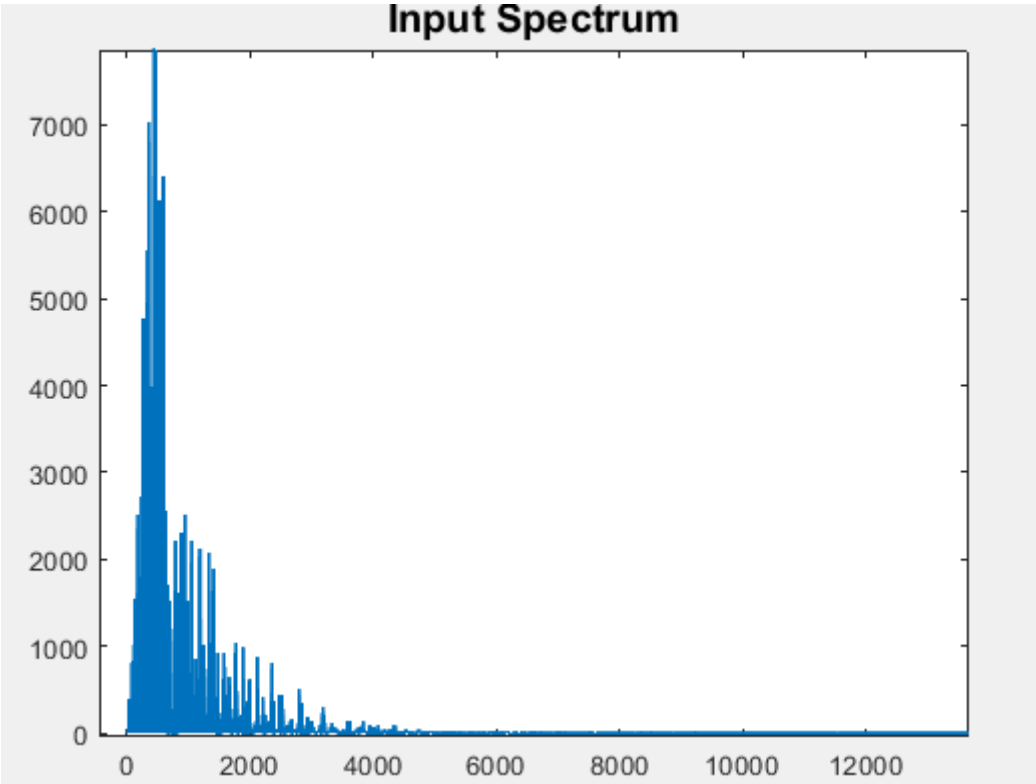
```
        if f_in(i,j)*128 > 127
            noise_shape_output(i,j) = 127;
        elseif f_in(i,j)*128 < -128
            noise_shape_output(i,j) = -128;
        else
            noise_shape_output(i,j) = f_in(i,j)*128;
        end
        noise_shape_output(i,j) = round(noise_shape_output(i,j));
        noise_shape_output(i,j) = noise_shape_output(i,j)/128;
    end
end
```

首先我宣告一個 f_in 的變數讓他等於 dithering 完的訊號，並初設 noise_shape_output 的值為 0。接著進入迴圈之中，去做 F_in(i) = F_in(i) + D(i) + cE(i－1)，由於這裡的訊號 f_in 已經都做完 dithering 了，故只需要做 F_in(i) =

F_in(i) + cE(i－1)的部分。接著，為了較符合理論，我先去量化了算出來的 F_in(i)值，得到 F_out(i)值，區間在-128~127 之間，這裡的做法是先讓 f_in(i)乘上 128，如果此值大於 127，則讓 noise_shape_output（noise shaping 最後的輸出）等於 127；若 f_in(i)乘上 128，此值小於-128，則讓 noise_shape_output（noise shaping 最後的輸出）等於-128。-128~127 之間的值則直接 assign 給 noise_shape_output。接著去做 round，做完之後得到-128~127 之間的整數值，再去除以 128。量化完成。
算 Error 的時候用 f_in(未量化的訊號) 去減 noise_shape_output，得出偏差值。如此重複去做。 將整個訊息做完。

做完 noise shaping 之後，要把高頻的雜訊弄掉，這裡使用第一題寫的 filter，cut off frequency 定在 4500。 因為做完 noise shaping 後可以將一些雜訊集中到高頻的地方，故用 lowpass filter 濾掉，又，觀察原始的 input 訊號，得知音樂的頻率主要落在 4500HZ 之前。

```
%% 5. Implement Low-pass filter
[noise_shape_output(:,1), ~] = myFilter(noise_shape_output(:,1), fs, 2001, "Blackman", "low-pass", 4500);
[noise_shape_output(:,2), ~] = myFilter(noise_shape_output(:,2), fs, 2001, "Blackman", "low-pass", 4500);
```



接著為了要讓音量在一定的區間，讓 normalization 的結果更好，所以先做 limiting。這裡取的 threshold 是+-0.9，用的方法是 hard clipping，將超出的值直接拉成 0.9 或-0.9。

```
%% 6. Audio limiting

for j = 1:2
    for i = 1:length(input)
        if noise_shape_output(i,j)<0
            if noise_shape_output(i,j)<-0.9
                noise_shape_output(i,j) = -0.9;
            end
        else
            if noise_shape_output(i,j)>0.9
                noise_shape_output(i,j) = 0.9;
            end
        end
    end
end
```

最後，做 normalization，也將其他的音量拉相同的比例。

```
%% 7. Normalization
MAX = max(abs(noise_shape_output));
peak = max(MAX);
normalization = 1 / peak;

recover_output = noise_shape_output * normalization;
```

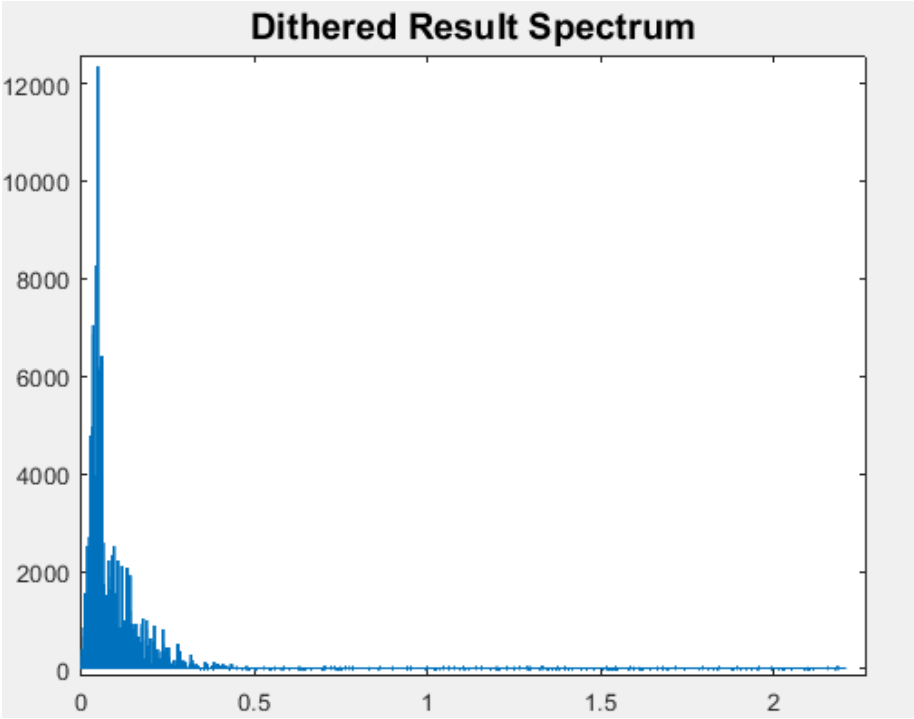
由於做完 noise shaping 之後，我們又去做了 filter、limiting、normalization。雖然在 noise shaping 的過程中我去將訊號量化成 8bit，但做了這些處理之後，sample 點的值可能又跑掉了，不在 8bit 表示的 scal 上，所以輸出音檔的時候一樣，告訴 audiowrite 我的 BitsPerSample 為 8，讓 audiowrite 輸出的音檔做量化仍用 8bit 表示。

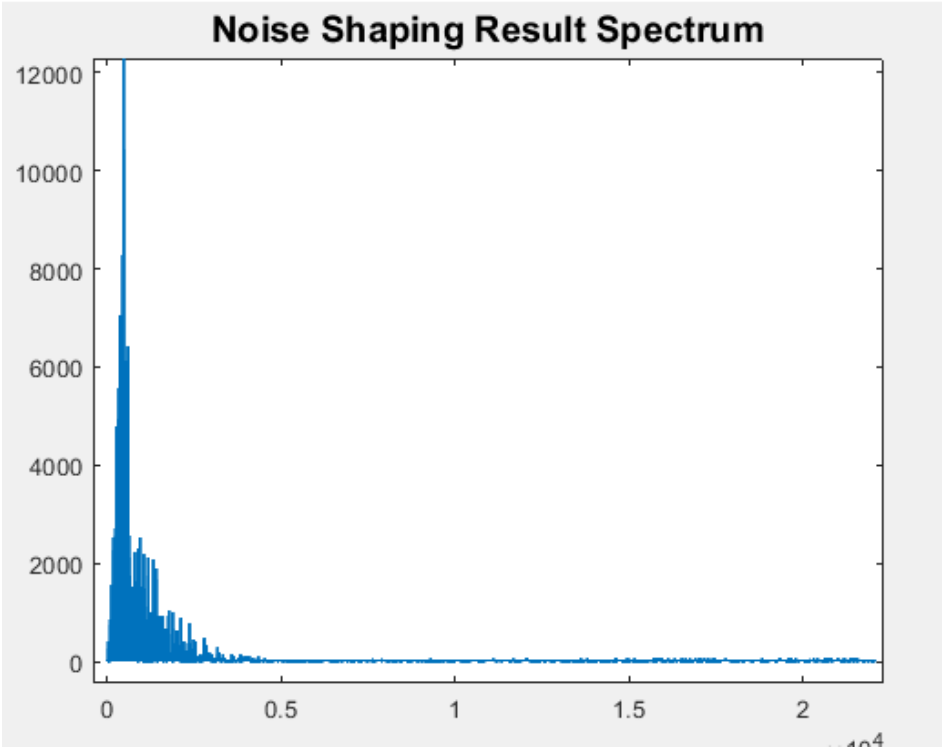
```
%% 6. Save audio (audiowrite) Tempest_Recover.wav

audiowrite("Tempest_Recover.wav", recover_output, fs, "BitsPerSample", 8);
```

以下是 dithering 和 noise shaping 做出來的圖，雖然說做出來圖顯示的差距沒

有 非常明顯（主要原因有因為其實一開始做完 bit reduction 的訊號是已經量化過的，而 dithering 和 noise shaping 理論上來說應該要在還沒量化前去做；還有 就是加上去的 noise 強度我調的較小，因為想要讓 Tempest_Recover 增加的 noise 不要太強）。但仍能夠從圖中觀察到，在做完 noise shaping 之後，noise 有往高頻的部分集中，而只做 dithering 的 noise 頻率分布較平均分散。





且我有試著將只做 dithering 的音檔和 Tempest_Recover 輸出，用聽的來做辨別，發現只做 dithering 的雜訊明顯比較大聲。又，將 Tempest_Recover 和

Tempest_8bit 做比較，會發現聽的時候，Recover 的版本可以明顯的將 noise 和音樂區分出來，而 8bit 版本較無法，且 Recover 版本的音樂不再像 8bit 版本的音樂，會有一階一階，有時候不太連續的現象發生，音樂基本上是連續平滑的。雖然說 Tempest_Recover 的雜訊可能大聲一點點（我認為是因為實作和理論的差距：我先做 bit reduction，再用 reduction 完的結果去做 dithering 加上 noise 的關係，且 noise shaping 無法完全將 noise 移到高頻濾掉，故雜訊可能變多。理論上要做量化之前，要先去做 dithering 和 noise shaping 的，但實做中並非如此。），但我認為做出來的結果仍算符合預期：音樂不再有一階一階的感覺，且音樂和 noise 可以明顯的區分開來。

（我發現用 audiowrite 輸出音檔，若沒有指定 BitsPerSample 為 8，audiowrite 預設為 16bit，則我的 Tempest_Recover 的效果會很理想，noise 變很小，且音樂一階一階的問題也消失了，但那是用 16bit 表示。在這次作業中我仍在輸出 Tempest_Recover 的時候，指定 BitsPerSample 為 8，為了符合助教的規範，故會有雜訊較大聲的情況發生。因為做完 filter 和 limiting 和 normalization 之後又需要再用 8bit 量化一次，又會有資訊喪失一次，這是我的觀察。）