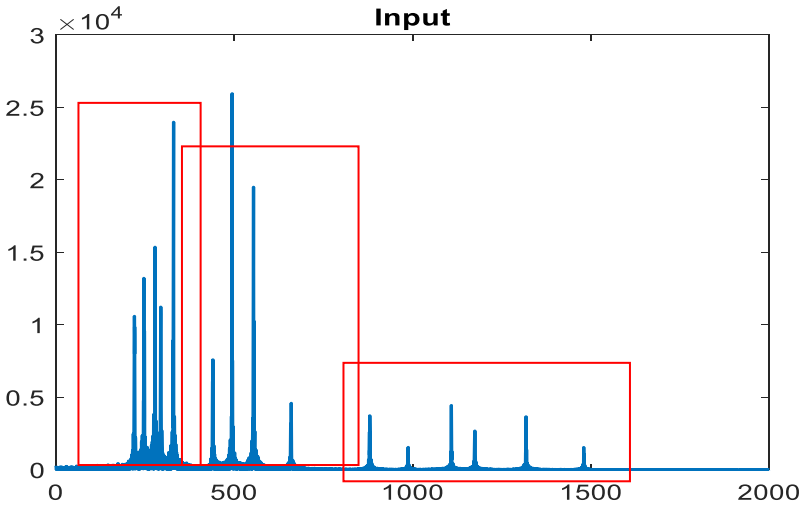
Multimedia Assignment2

Q1. Create my own FIR filters to filter audio signal

※Implementation

To separate three songs form one audio file, by observing the input spectrum , I found out that there's special frequency distribution.
I can use exactly low-pass , bandpass and highpass filter accordingly.




```
%low-pass
[outputSignal, outputFilter] = myFilter(y_input, fs, 1501,'Blackmann', 'low-pass', 380);
%bandpass
[outputSignal_pass, outputFilter_pass] = myFilter(y_input, fs, 1501,'Blackmann', 'bandpass', [380 701]);
%high-pass
[outputSignal_high, outputFilter_high] = myFilter(y_input, fs, 1501,'Blackmann', 'high-pass', 701);
```

And since there very few frequency above 1500, so I chose 1501 for the fsample , and chose cut frequency (fcutoff)due to the spectrum(below 380 is low-pass, between 380 and 701 id bandpass , above 701 is high-pass).
As for the filters implementation and blackmann window function, just referring the formulas on slides and they were done easily.
However ,filter the input signal to time domain, I referred files online and copy it.
(The link is on last page.)
For one/multiple fold echo, I also referred to slide

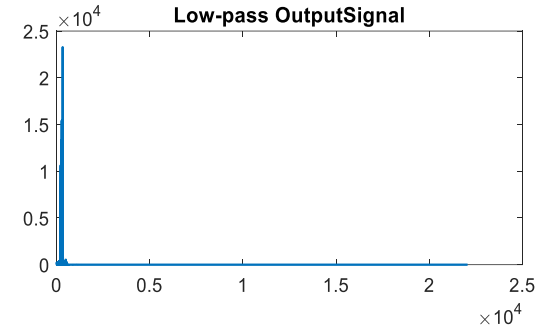
One-fold echo: a_k = [1, 0, 0, 0, ..., 0, 0.8] , b_k = [1]
(That is, 3199 zeros between 1 and 0.8.)
The output of the filter is: **y[n] = x[n] + 0.8*x[n-3200]**

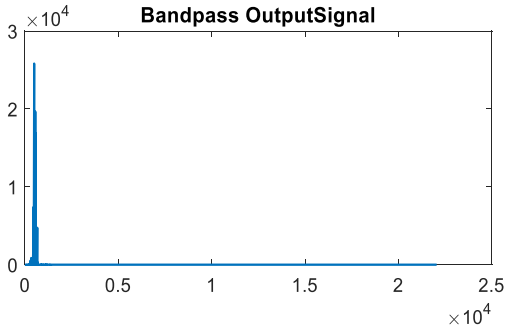
Multiple-fold echo: a_k = [1], b_k = [1, 0, 0, 0, ..., 0, -0.8]
(That is, 3199 zeros between 1 and -0.8.)
The output of the filter is: **y[n] = x[n] + 0.8*y[n-3200]**

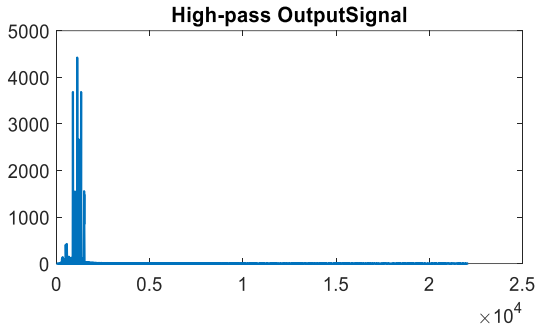
But there's something worth notice, which's that since n-3200 ,so the for loop of doing echo, we have to shift index to prevent data clipping(I'll discuss later).

※Result display & Discussion

The spectrums of the output signals(before echo):

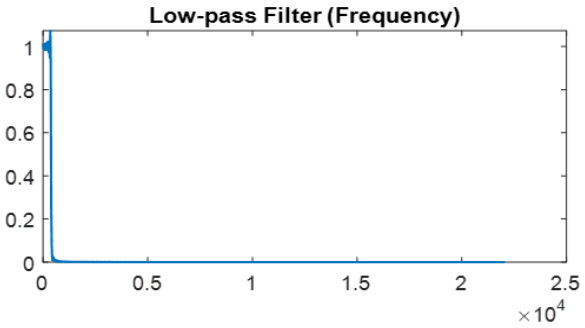


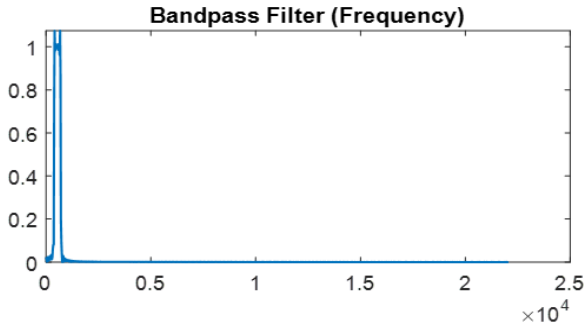


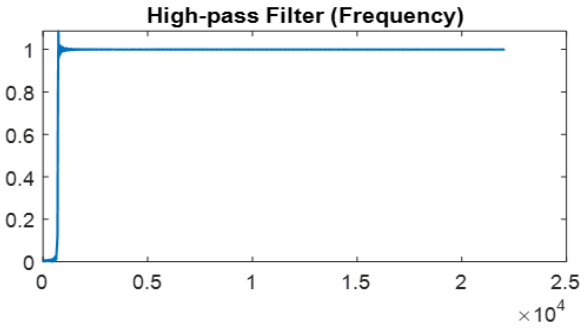


→Compare spectrum and shape of the filters

We can see clearly from the image below that the spectrum of three filters are meets their names. For low-pass filter, magnitude gets very high on low region, bandpass filter magnitude gets high in the middle and high-pass filter gets high after a certain frequency. So we can conclude that by their own characteristics , we can use them in different occasions.



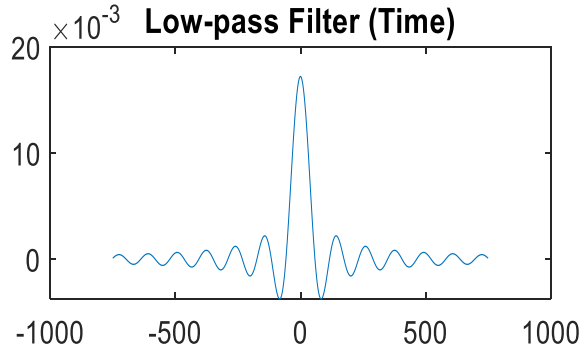


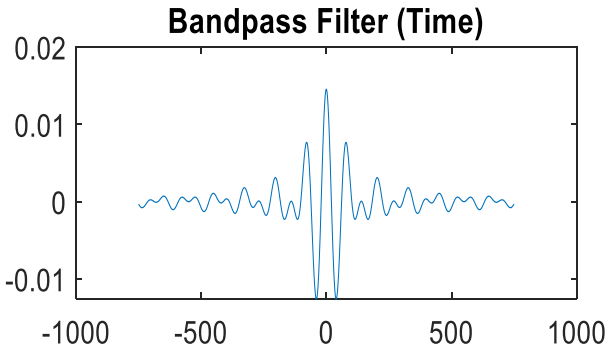


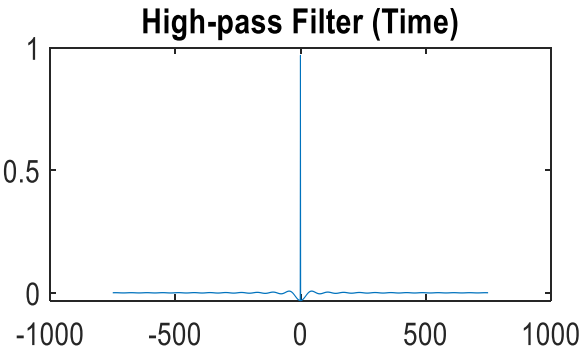
→difference between signals before and after reducing the sampling rates

For low-pass filtered signal and bandpass filtered signal, there is no obvious difference between them, since they are low frequency. However, high-passed audio signal gets worse after changing sampling rate to 2000HZ. Since we use high-pass filter due to its frequency is high, if we use low sample rate, we can't get signal from high region and thus the audio quality becomes worse.

The shapes of the filters (time domain):







→time domain

In the time domain, the low-pass filter is a sin function, and there are two functions subtracted in the bandpass filter, and high-pass filter is a negative sin function. So that's why the figures in time domain look like this.

→something worth mention

I encountered this problem first when I implemented multiple hold echo(It also happned for many times when I was doing question 2.)

```
Warning: Data clipped when writing file.
> In audiowrite>clipInputData (line 404)
  In audiowrite (line 184)
  In HW2_Q1 (line 130)
```

IN the beginning, I didn't take it seriously since it was a "warning", but then I found that

the results were very strange, so I googled and got this:

```
The valid range for the data in y depends on the data type of y.
   double    -1.0 ≤ y ≤ +1.0
```

If you data is out of that range you should divide it by max(abs(YourData(:)))

So I modified my code.

```
123 –    multi_fold_out = outputSignal;
124 –    for n=3200:length(outputSignal)-1
125 –        multi_fold_out(n+1,1)=outputSignal(n+1,1)+0.8*multi_fold_out(n-3200+1,1);
126 –    end
127
128      %% 5. Save the echo audios  'Echo_one.wav' and 'Echo_multiple.wav'
129 –    audiowrite('Echo_one.wav', one_fold_out, fs);
130 –    audiowrite('Echo_multiple.wav',multi_fold_out, fs);
```

Interesting Observation:

There's no problem with one_fold_out signal since multiple_fold_out accumulate the echo and add to it repeatedly. Therefore , it gets over the range of -1 ~ 1 ,which is the range that matlab audiowrite default type(double).

uint8	0 ≤ y ≤ 255
int16	-32768 ≤ y ≤ +32767
int32	-2^31 ≤ y ≤ 2^31–1
single	-1.0 ≤ y ≤ +1.0
double	-1.0 ≤ y ≤ +1.0

Q2. Audio dithering and noise shaping

※Implementation

→bit reduction

Since,input audio signal is double from -1 to 1, and I want to change to 8 bit, so I multiply 128 directly . Be careful that we have to store data in "uint8" type,which is another storing type in matlab ,int 0~255.

8 Bit Value (0-255) or (-128~127)
16 Bit Value (Integer) (0-65535) or (-32768~32767)

Furthermore , I forgot to set ('BitsPerSample' , 8) when audiowrite and my .wav file has no sound. Since we change bit per sample, we need to specify the bit number after we change it.

```
load handel.mat

filename = 'handel.flac';
audiowrite(filename,y,Fs,'BitsPerSample',24,...
'Comment','This is my new audio file.');
clear y Fs
```

```
% (Hint) remember to save the file with bit rate = 8
audiowrite('Tempest_8bit.wav', uint8(y_8bit), fs, 'BitsPerSample',8); %!!
```

→audio dithering

Just adding noise to the file. I use a random number vector from 0 to 1,
and add it to my 8_bit signal . At the beginning, I just add noise without any scaling, and therefore, no obvious difference before and after dithering . So I multiplied it to 128 times and we can hear clearly the background noise.

```
noise = rand(size(y_8bit));
y_noise = y_8bit + noise*128;
```

→ noise shaping

I think this is the most difficult part in question 2. Since I misinterpreted the information in slides.

- The assignment statement $F\_in_i = F\_in_i + D_i + cE_{i-1}$ dithers and noise shapes the sample. Subsequently, $F\_out_i = [F\_in_i]$ quantizes the sample.
- E_i is the error resulting from quantizing the ith sample after dithering and noise shaping.
- For $i = -1$, $E_i = 0$. Otherwise, $E_i = F\_in_i - F\_out_i$.

#18

- The general statement for an n-th order noise shaper noise shaping equation becomes $F\_out_i = F\_in_i + D_i + c_{i-1}E_{i-1} + c_{i-2}E_{i-2} + \cdots + c_{i-n}E_{i-n}$.

#19

I thought that every F_in = F_in + D + cE , and F_out are constructed by every F_in ,which is F_out = [F_in] , so I understanded it as F_out1 = F_in1 + D1 + cE0 + F_in2 + D2 + cE1+ F_in3 + D3 + cE2…. But in silde #19 , it's not accord with my understanding. So I got confused for a long time.

Lucky I found this and finally realized that there's only one F_in and many many D + cE in every steps to store to a F_out . So it is like F_out1 = F_in1 + D1 + cE0 + D2 + cE1 + D3 + cE2….

```
 s = (2^b_orig)/(2^b_new);
 c = 0.8;  //Other scaling factors can be tried.*/
 e = 0;
 for (i = 0; i < N; i++) {
*Get a random number between -1 and 1 from some probability density function*/
    d = pdf();
    F_scaled  = F_in[i] / s;  //Integer division, discarding remainder
    F_scaled_plus_dith_and_error = F_scaled + d + c*e;
    F_out[i] =  floor(F_scaled_plus_dith_and_error);
    e = F_scaled - F_out[i];
 }
```

→ low-pass filter

Just use my low-pass filter in question 1 , and I observed the image of after dithering audio and found out that most significant magnitude are under 2000 frequency , so I chose fsample of 2000 and cutoff frequency of 1500 to filter out noise.

```
ut, outputFilter] = myFilter(round(F_out)/128-1, fs, 2000,'B]
```

But a tricky place is that since I store audio after noise shaping in 0~255 uint8 style, so when I use myFilter, I had to change it to -1 ~ 1 default storing style. Otherwise , there's no sound in my .wav file(I stored .wav file in every step to help me debug and track which step I get wrong).

→audio limiting

Very simple step! Just referred image below:

I chose threshold 0.8.

→normalization

Use threshold in hard clipping step above(0.8) and change every signal back in that scale.

```
norm = 1/thresh;
for n=0:size(y_output)-1
    y_output(n+1,1) = y_output(n+1,1)*norm;
end
```

→something worth mention

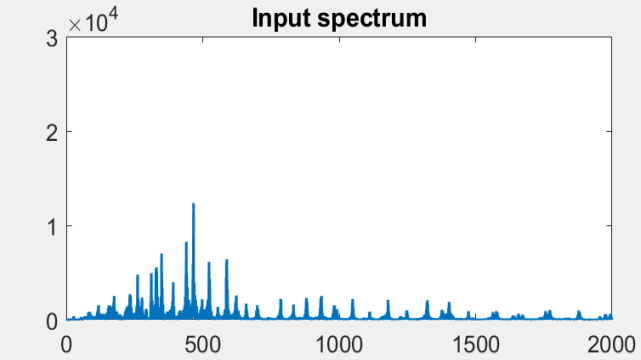
Two channels audio track

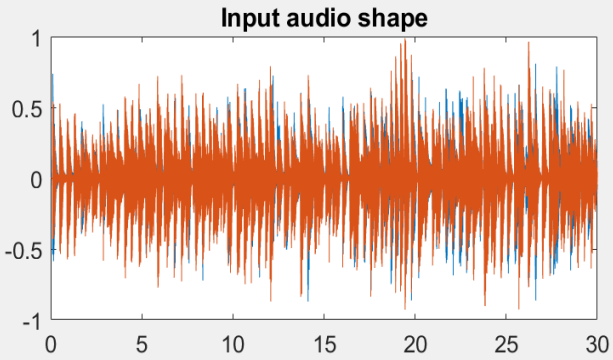
```
-0.1166    0.4379
-0.0819    0.4491
-0.0467    0.4496
-0.0099    0.4378
 0.0288    0.4137
 0.0693    0.3781
 0.1099    0.3330
 0.1489    0.2809
 0.1848    0.2250
 0.2164    0.1686
 0.2419    0.1147
 0.2593    0.0650
 0.2671    0.0213
 0.2649   -0.0160
 0.2542   -0.0465
 0.2372   -0.0707
```

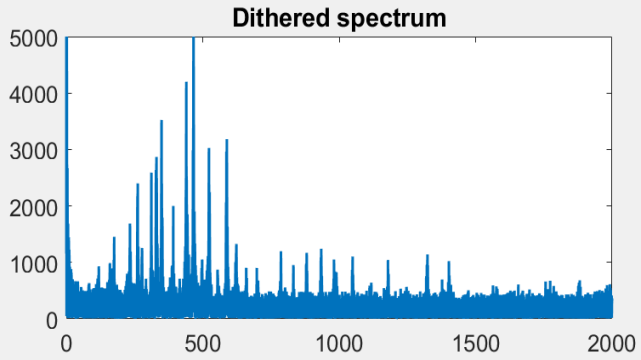
In the beginning, I forgot to test channel number of each input audio file, so I use only one for loop to process every step in Q2 (Q1 has no problem since it's one channel audio).

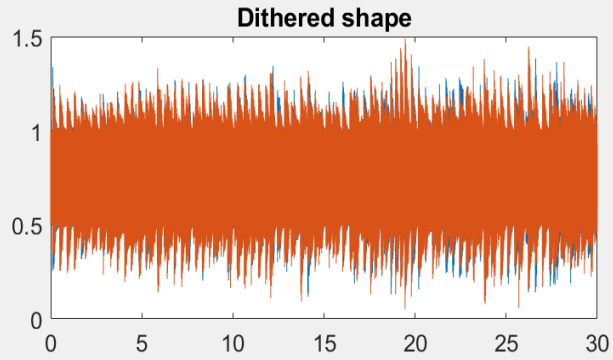
And thus, my result was totally wrong! So ,test the input first is important!

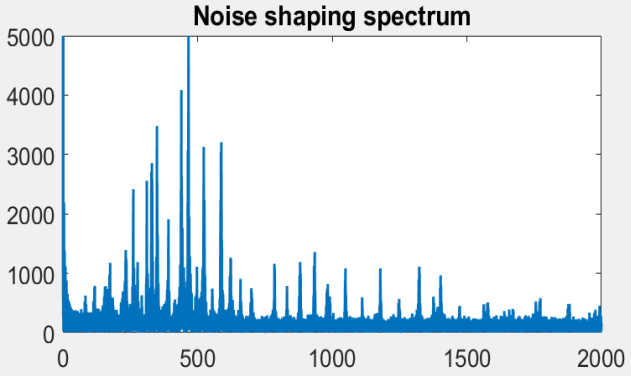
→effect of dithering and noise shaping according to the spectrums and shapes

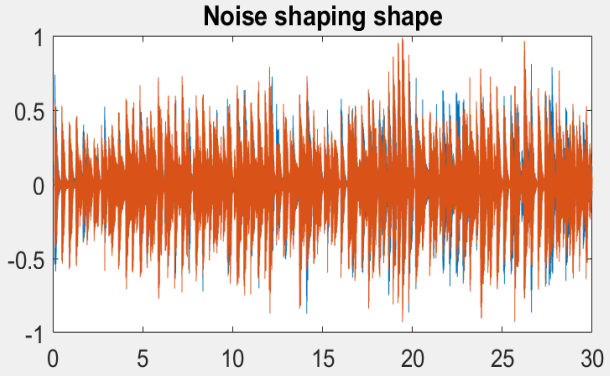


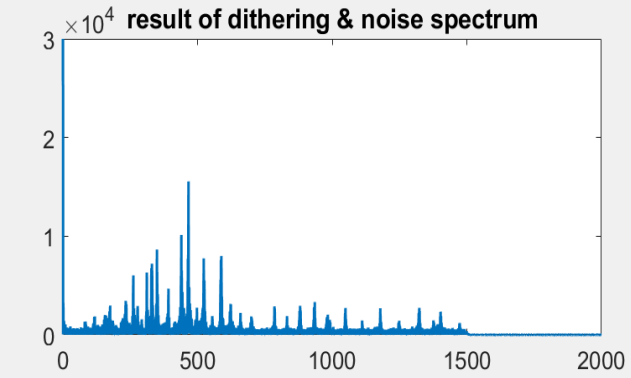


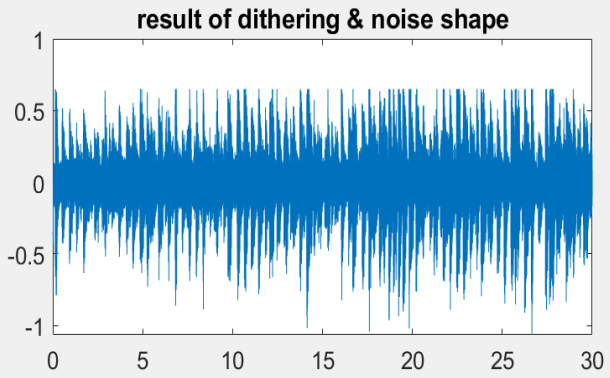












After dithering, there's steady noise signal in whole spectrum image(low frequency zone), and the shape of audio distorted by noise. My noise set to range -1 ~1 initially, but after listened the audio file, I could hardly hear the noise! Since comparing with my music signal(0~255) , noise are too little. So I decided to add noises by multiplied them with 128. (Another reason is that I want to "destroy" the music badly and rebuild it to see how perfect it will be repaired.)

After noise shaping, many noises disappeared ,but there were still certain amount of them couldn't be filtered out. And if we drag the spectrum to the end, we can see that there's a extremely steep(slope is almost 1) rise of frequency. That's because we add error continuously one after another, and thus noises are accumulated from the beginning to the end of the file.

Last step, we only need to cut off those noise region and save front of audio signal, which is without noise interfering. And we can see that the sound spectrum is almost the same as beginning. But since high frequency signal were cut off, we can notice that after my sampled spot, nothing remain.

(There's still some trivial flaw, I don't know if it's normal or my rebuilding procedure not perfect yet?)

P.S. I don't know why the output of shape of audio is in color blue while other shapes are in color orange?

Reference:

http://mirlab.org/jang/books/audiosignalProcessing/filterApplication.asp?title=11-1%20Filter%20Applications%20(%C2o%AAi%BE%B9%C0%B3%A5%CE)

Filter the input signal in time domain

https://www.mathworks.com/matlabcentral/answers/24184-convolution-discrete-time-not-using-conv

noise shaping

http://digitalsoundandmusic.com/5-3-7-the-mathematics-of-dithering-and-noise-shaping/

resample

https://www.mathworks.com/help/signal/ref/resample.html?fbclid=IwAR3SOPgGP_9KkvGyl4EneQ8aB_42XLLFGs7EfwoAy81n8sfMxQDkTFXmTQA#bumh2jz-1

audiowrite (extremely important)

https://www.mathworks.com/help/matlab/ref/audiowrite.html?fbclid=IwAR2bMX0qEFtjKEbs2M3-NUw5XXH5p0WMRslCrmmFNZs1TAsMaZ1T0FA2UAs#btiacgz-1-y