Assignment 3, Digital Signal processing: IIR filters
Weng, Wei-Ming(2305670)

Q1: Present an experiment which shows the difficulties of filtering when the spectra of signal and noise are overlapping…

A1: The experiment we chose was “Talking while stir frying with the extractor hoods on”, the basic concept is to give a short speech while stir frying the vegetables with the extractor hoods on. With the noise of the frying pan and the extractor hoods, the conversation in the kitchen can be vague. Nevertheless, by using the designed iir filter, the SNR ratio can be highly increased. Human’s voice frequency goes from 55Hz to 1200Hz. From the wild track we recorded, the noise frequency majorly locates on 100 and 200Hz, other small peaks also locates at 238, 596, 799 and1522Hz.

Experiment date: 2017.12.17
Location: Kitchen in Kelvin Court, 30 Yorkhill Street, Glasgow, G3 8RY, United Kingdom, Room: Block B/Floor 5/Flat 505/Room No B
Recording equipment: OPPO R9 smartphone
Website to convert .mp3 to .wav: https://online-audio-converter.com/tw/
Google map: https://www.google.com/-maps/@55.8666054,4.2941228,2a,90y,183.09h,68.31t/data=!3m6!1e1!3m4!1s9MQXQqtI4MkAAAQvO7aV1A!2e0!7i13312!8i6656

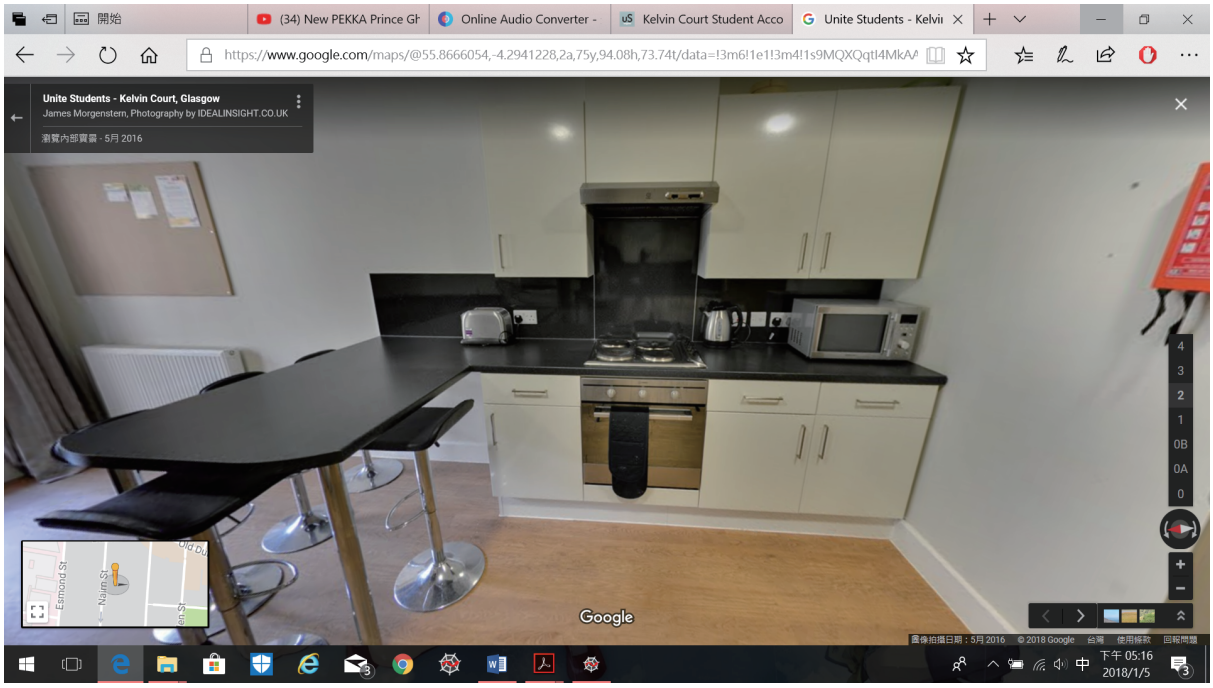
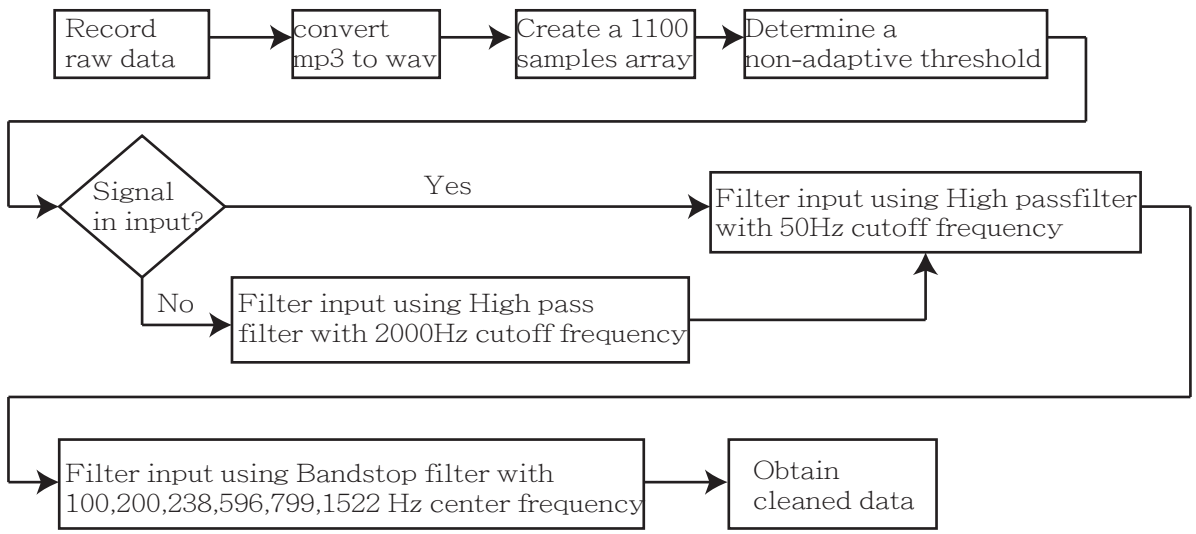


Fig. 1. Kitchen of Kelvin Court

Data flow diagram showing the concept of this experiment:



Q2: Determine the ideal filter response(s) which remove as much noise as possible while leaving the signal intact (i.e. maximises signal to noise ratio)…

A2: In this step, I will have to analyze the frequency and the time domain of the noise, signal + noise, and pure signal. After that, I tried to design a suitable system for this condition. The solution I came up with was to analyze the amplitude of the frequencies in the wild track first, find out where the majority frequency lies, then determine which kind of filter I would need. Moreover, I wrote a determine function to enable and disable the high pass filter with a constructed threshold, which filters the noise when I am not speaking. The plotted pictures of the noise, signal + noise, and pure signal are down below.

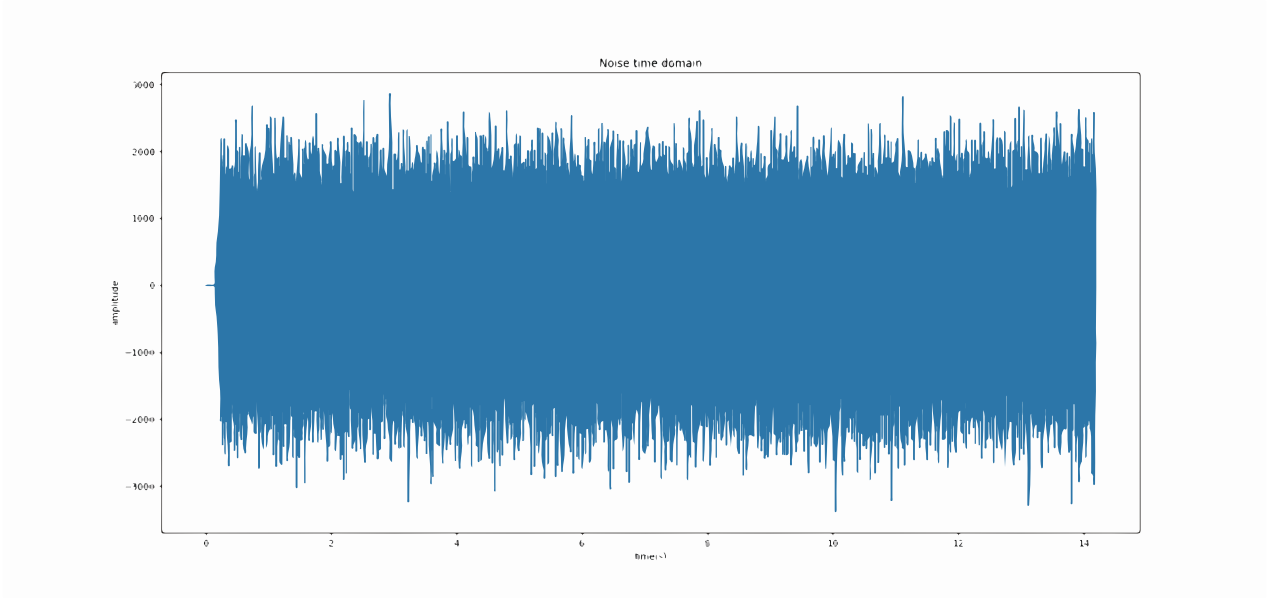


Fig. 2. Noise time domain

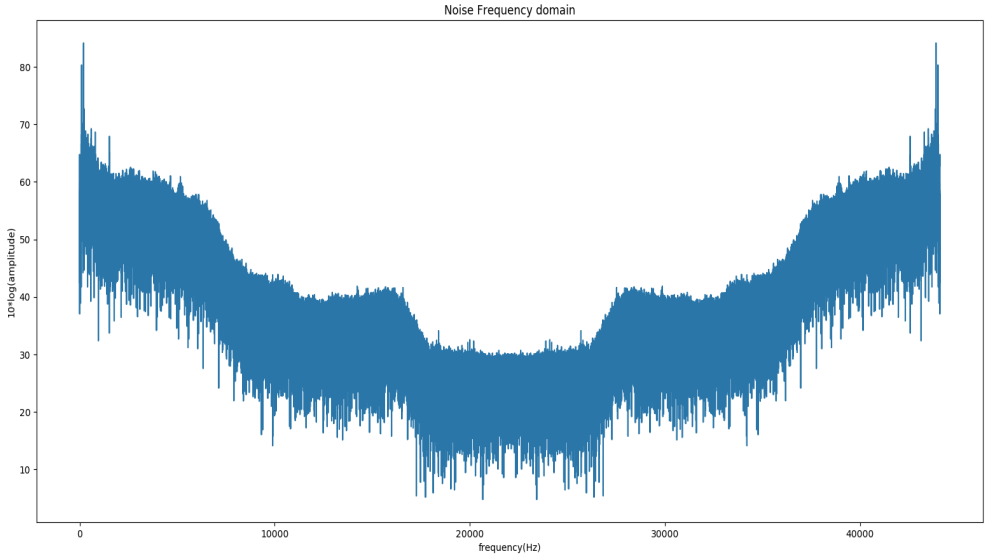


Fig. 3. Noise frequency domain

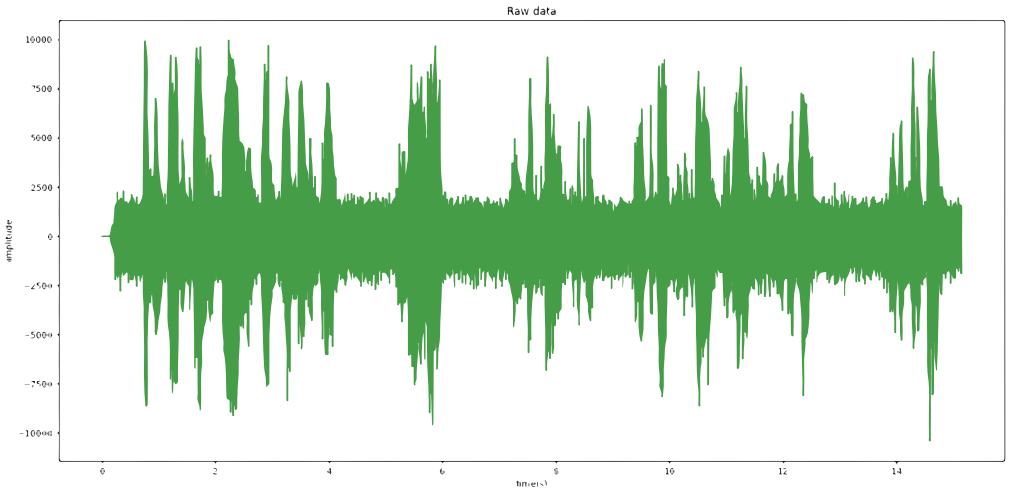


Fig. 4. Raw data time domain

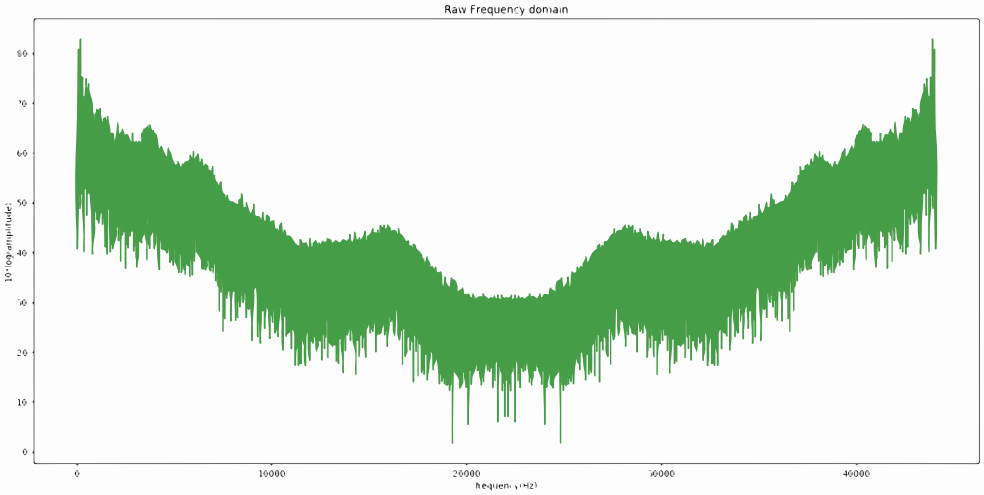


Fig. 5. Raw data frequency domain

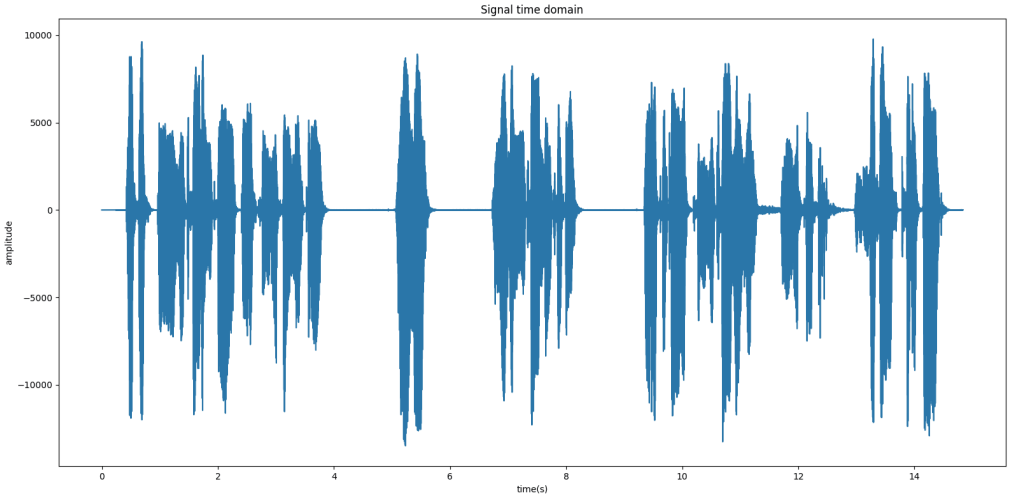


Fig. 6. Pure signal time domain

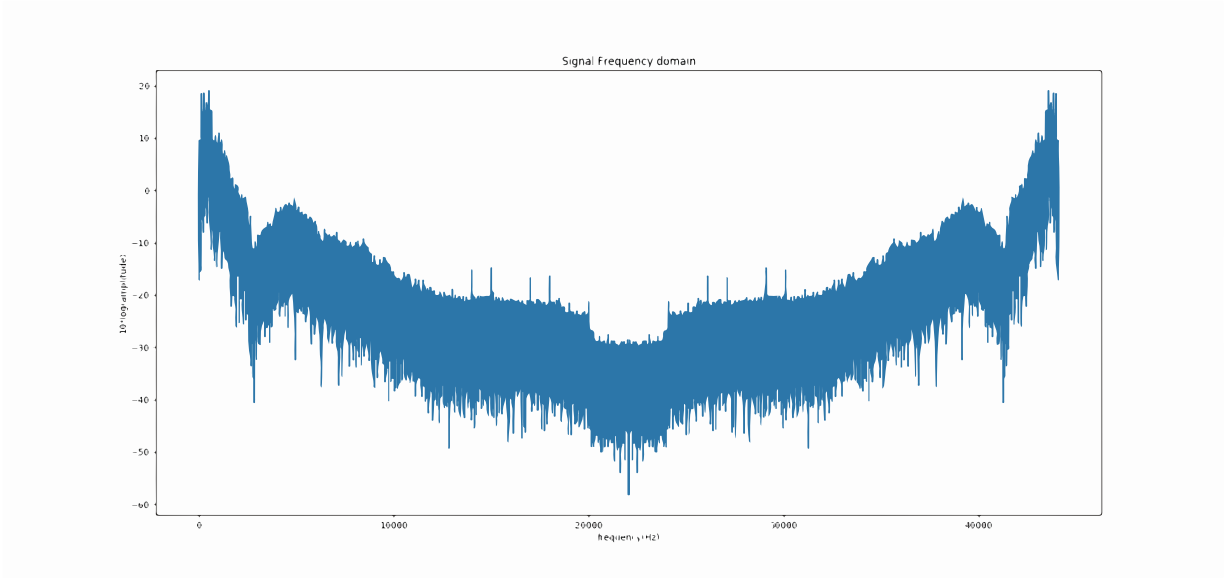


Fig. 7. Pure signal frequency domain

In the first place, a determine formula is being used to decide whether the input has human sound or not. By using a threshold (not adaptive) and a template to store 1100 samples, we could determine whether there is the signal in this 0.025 second. If there are no signal in the input, it applies a Chebyshev highpass filter with a cutoff frequency at 2000Hz to the input. The code is shown down below:

```
clean_data=np.zeros(len(raw_data))
clean_data=raw_data
#not adaptive threshold and implement it on the time domain
fc=2000
fc=fc/rate
b,a=signal.cheby2(2,20,2*fc,'high')
f0=IIR2filter(b[0],b[1],b[2],a[1],a[2])
#check every 0.05 seconds whether to apply the filter or not
interval=1100
template=np.zeros(interval)
temp_data=np.array([])
threshold=5150
counter=0
for i in range(len(clean_data)):
    if(counter!=interval-1):
        template[counter]=clean_data[i]
        counter=counter+1
    elif(counter==interval-1 or i==(len(clean_data)-1)):
        template[counter]=clean_data[i]
        if ((np.max(template)-np.min(template))<threshold):
            for j in range(len(template)):
                template[j]=f0.filter(template[j])
        temp_data=np.concatenate((temp_data,template))
        counter=0
clean_data=temp_data
```

After the threshold judgment, Butterworth high pass filter with a cutoff frequency at 50Hz is being used to get rid of the DC signal and non-human voice under 50 Hz. The code is shown down below:

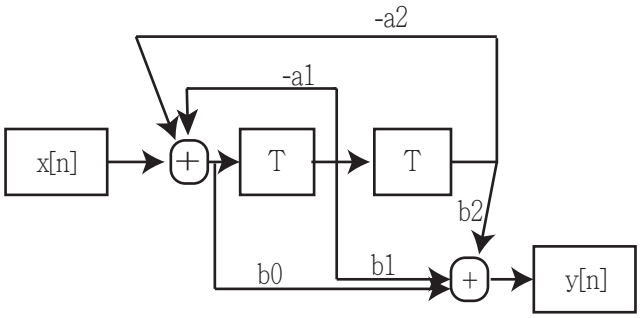
```
#highpass, indicate which central frequency to use for the different input
fc=50
fc=fc/rate
b,a=signal.butter(2,2*fc,'high')
f1=IIR2filter(b[0],b[1],b[2],a[1],a[2])
for i in range(len(clean_data)):
   clean_data[i]=f1.filter(clean_data[i])
```

The third step is to apply 6 Butterworth bandstop filter to the outcome of the high pass filter, since at 100, 200, 238, 596, 799, 1522 Hz noise have a significant amplitude. The code is shown down below:

```
#third, require bandstop to all of the data
freq=np.array([100,200,238,596,799,1522])
freq=freq/rate
w=np.pi*2*freq
for i in range(0,len(freq)):
   b,a=signal.butter(1,[2*(freq[i]-5/rate),2*(freq[i]+5/rate)],'stop')
   f2=IIR2filter(b[0],b[1],b[2],a[1],a[2])
   for j in range(0,len(clean_data)):
      clean_data[j]=f2.filter(clean_data[j])
clean_data=np.real(clean_data)
```

Q3: Write a class IIR2Filter which implements a 2nd order IIR filter which takes the coefficients in the constructor…

A3: Basically, the IIR 2 order data flow diagram looks like:



By turning the data flow diagram into Python, the code is shown down below:

```
class IIR2filter:

   def __init__(self,b0,b1,b2,a1,a2):
      self.a1=a1
      self.a2=a2
      self.b0=b0
      self.b1=b1
      self.b2=b2
      self.buffer1=0
      self.buffer2=0
   def filter(self,x):
      acc_input=x-self.buffer1*self.a1-self.buffer2*self.a2
      acc_output=acc_input*self.b0+self.buffer1*self.b1+self.buffer2*self.b2
      self.buffer2=self.buffer1
      self.buffer1=acc_input
      return acc_output
```

Q4: Compare your filtered results with the original recordings and discuss if you have been successful. Do a critical analysis.

A4: From the comparison of the plotted time domain and frequency domain pictures, cleaned data and the raw data has a major difference. The data of the cleaned data is shown down below:

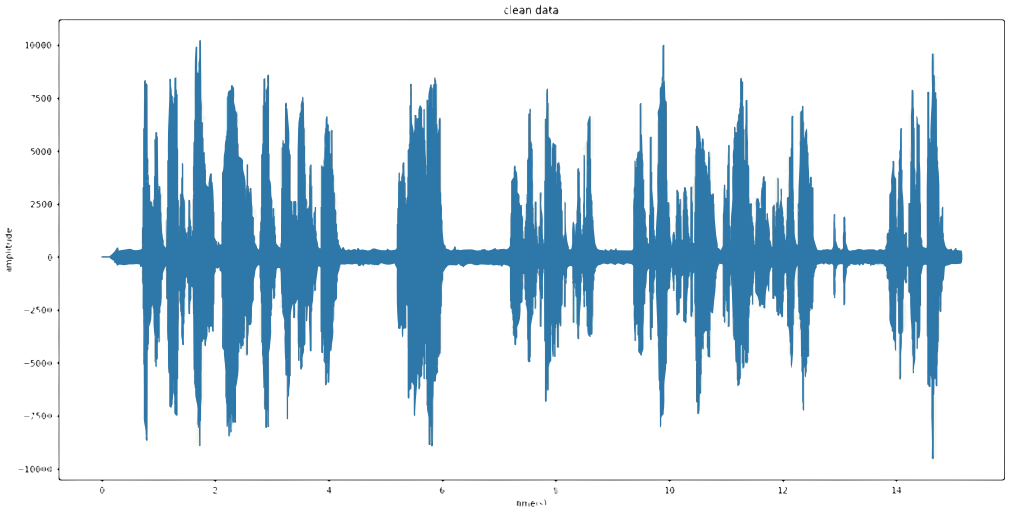


Fig. 8. Cleaned data time domain

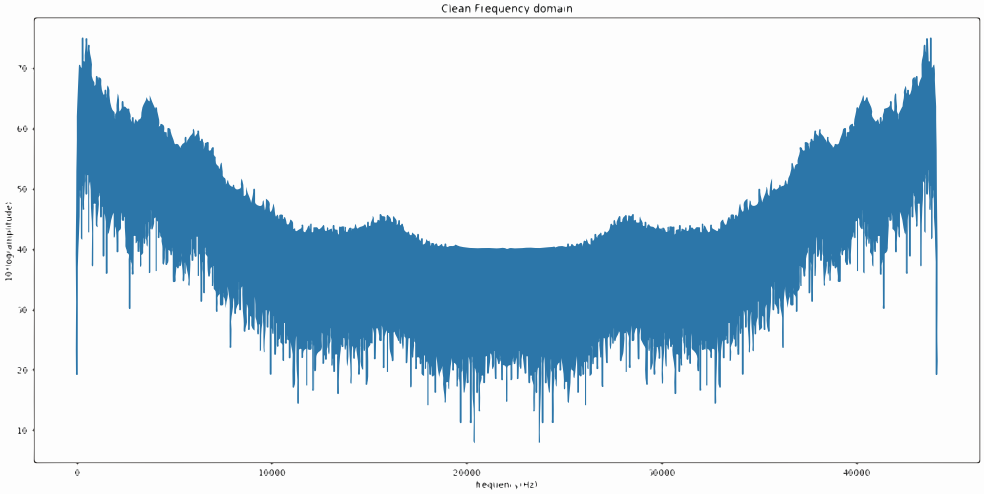


Fig. 9. Cleaned data frequency domain

By observation, the frequency near 0Hz has significantly dropped, however the amplitude above 1200Hz still has a great amount of amplitude. Since our noise source - frying pan and the extractor hoods has a wide range of frequency which goes from 0~10000+Hz, we only got rid of the lower frequency noise which is lower than 2000. Nevertheless, form the noise frequency domain amplitude, we know that lower frequency has a bigger contribution to the noise. In this case, despite only getting rid of the frequency below 2000Hz, the SNR and the cleaned data time domain looks much cleaner than the raw data. However, there is really no need to increase the cutoff frequency of the filter since once we speak, the threshold will not work, and background noise will not go through the 2000Hz high pass filter.

Appendix:

```
"""
Created on Mon Dec 25 20:09:56 2017

@author:Weng Wei-Ming 2305670
"""
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pylab as matplt
import scipy.io.wavfile as wavfile
import scipy.signal as signal

rate,raw_data = wavfile.read("Signal+Noise.wav")
rate,noise = wavfile.read("Noise.wav")
rate,data = wavfile.read("Signal.wav")
raw_dataf=np.fft.fft(raw_data)
noisef=np.fft.fft(noise)
dataf=np.fft.ifft(data)

#method two:many bandpass
#create a class that basically contain two order filter

class IIR2filter:
    def __init__(self,b0,b1,b2,a1,a2):
        self.a1=a1
        self.a2=a2
        self.b0=b0
        self.b1=b1
        self.b2=b2
        self.buffer1=0
        self.buffer2=0
    def filter(self,x):
        acc_input=x-self.buffer1*self.a1-self.buffer2*self.a2
        acc_output=acc_input*self.b0+self.buffer1*self.b1+self.buffer2*self.b2
        self.buffer2=self.buffer1
        self.buffer1=acc_input
        return acc_output

#by judging the wild_track we can analysize the main noise frequencies
#The freqiencies are 100,200,238,596,799,1522
clean_data=np.zeros(len(raw_data))
clean_data=raw_data
#not adaptive threshold and implement it on the time domain
fc=2000
fc=fc/rate
b,a=signal.cheby2(2,20,2*fc,'high')
f0=IIR2filter(b[0],b[1],b[2],a[1],a[2])
#check every 0.05 seconds whether to apply the filter or not
interval=1100
template=np.zeros(interval)
temp_data=np.array([])
threshold=5150
counter=0
for i in range(len(clean_data)):
    if(counter!=interval-1):
        template[counter]=clean_data[i]
        counter=counter+1
    elif(counter==interval-1 or i==(len(clean_data)-1)):
        template[counter]=clean_data[i]
        if ((np.max(template)-np.min(template))<threshold):
            for j in range(len(template)):
                template[j]=f0.filter(template[j])
        temp_data=np.concatenate((temp_data,template))
        counter=0
clean_data=temp_data
#highpass, indicate which central frequency to use for the different input
fc=50
fc=fc/rate
b,a=signal.butter(2,2*fc,'high')
f1=IIR2filter(b[0],b[1],b[2],a[1],a[2])
for i in range(len(clean_data)):
    clean_data[i]=f1.filter(clean_data[i])
```

```
#third, require bandstop to all of the data
freq=np.array([100,200,238,596,799,1522])
freq=freq/rate
w=np.pi*2*freq
for i in range(0,len(freq)):
    b,a=signal.butter(1,[2*(freq[i]-5/rate),2*(freq[i]+5/rate)],'stop')
    f2=IIR2filter(b[0],b[1],b[2],a[1],a[2])
    for j in range(0,len(clean_data)):
        clean_data[j]=f2.filter(clean_data[j])
clean_data=np.real(clean_data)

clean_dataf=np.fft.fft(clean_data)
faxis=np.linspace(0,rate,len(clean_dataf))
plt.figure(1)
matplt.title('Clean Frequency domain')
matplt.xlabel('frequency(Hz)')
matplt.ylabel('10*log(amplitude)')
plt.plot(faxis,10*np.log10(abs(clean_dataf)))

taxis=np.linspace(0,len(clean_data)/rate,len(clean_data))
plt.figure(2)
matplt.title('clean data')
matplt.xlabel('time(s)')
matplt.ylabel('amplitude')
plt.plot(taxis,clean_data)

faxis=np.linspace(0,rate,len(raw_dataf))
plt.figure(3)
matplt.title('Raw Frequency domain')
matplt.xlabel('frequency(Hz)')
matplt.ylabel('10*log(amplitude)')
plt.plot(faxis,10*np.log10(abs(raw_dataf)))

taxis=np.linspace(0,len(raw_data)/rate,len(raw_data))
plt.figure(4)
matplt.title('Raw data')
matplt.xlabel('time(s)')
matplt.ylabel('amplitude')
plt.plot(taxis,raw_data)

faxis=np.linspace(0,rate,len(noisef))
plt.figure(5)
matplt.title('Noise Frequency domain')
matplt.xlabel('frequency(Hz)')
matplt.ylabel('10*log(amplitude)')
plt.plot(faxis,10*np.log10(abs(noisef)))

taxis=np.linspace(0,len(noise)/rate,len(noise))
plt.figure(6)
matplt.title('Noise time domain')
matplt.xlabel('time(s)')
matplt.ylabel('amplitude')
plt.plot(taxis,noise)

faxis=np.linspace(0,rate,len(dataf))
plt.figure(7)
matplt.title('Signal Frequency domain')
matplt.xlabel('frequency(Hz)')
matplt.ylabel('10*log(amplitude)')
plt.plot(faxis,10*np.log10(abs(dataf)))

taxis=np.linspace(0,len(data)/rate,len(data))
plt.figure(8)
matplt.title('Signal time domain')
matplt.xlabel('time(s)')
matplt.ylabel('amplitude')
plt.plot(taxis,data)

clean_data=np.float32(clean_data/np.max(np.abs(clean_data)))
wavfile.write('clean_signal.wav',44100,clean_data)
```