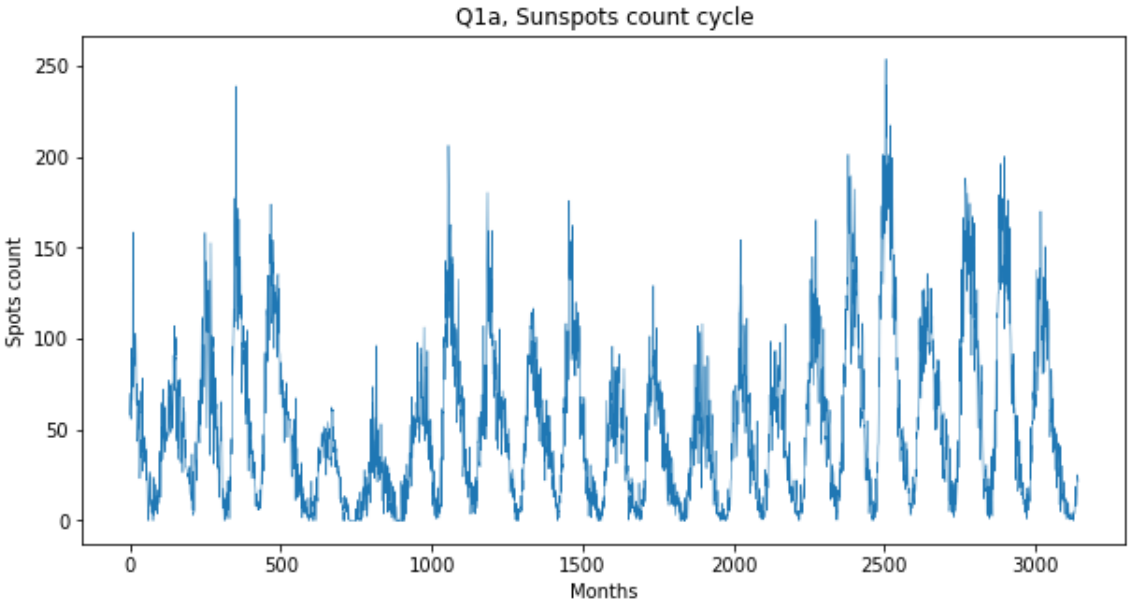
# Lab5 Report

Zirui Wan (Question 2), Rundong Zhou (Question 1)

October 16, 2020

## 1 Question 1

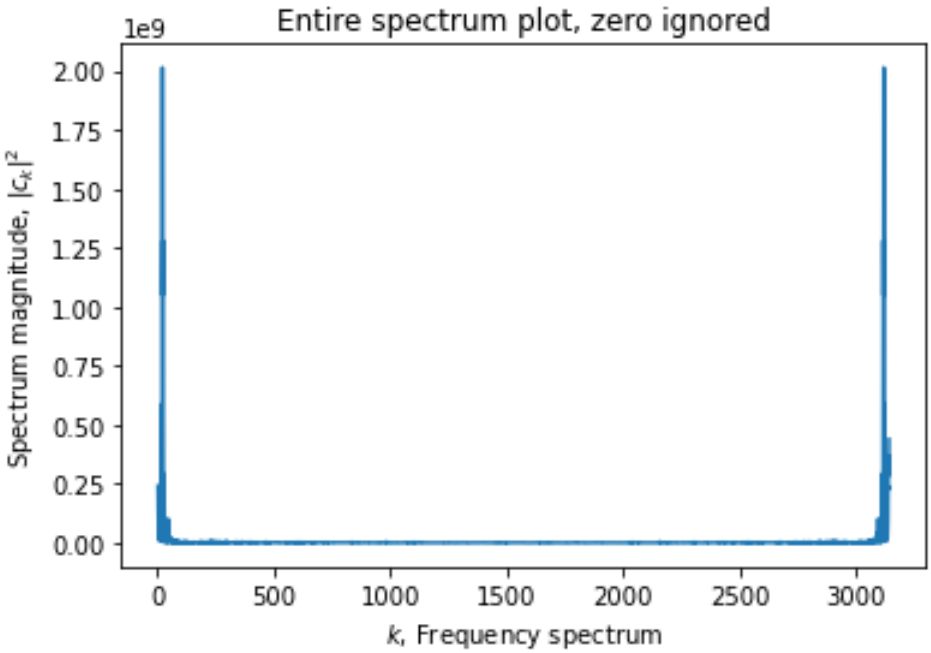
### 1.1 Q1a, Newman 7.2



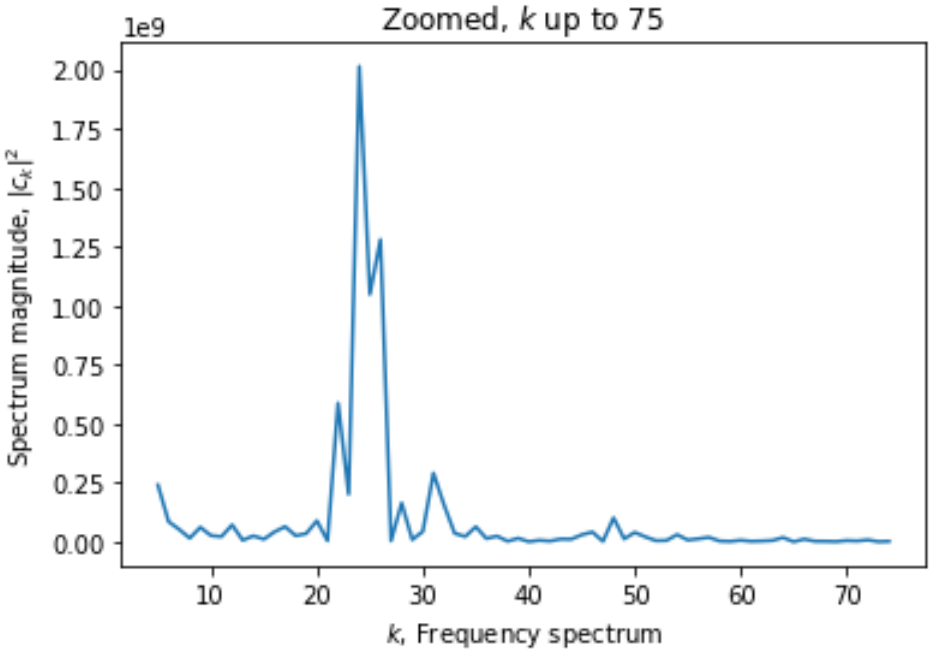

I plot the count of sunspots over the entire time span. From the graph we can tell that there are roughly 4 cycles in around 500 months period. So the estimated period can be easily calculated:

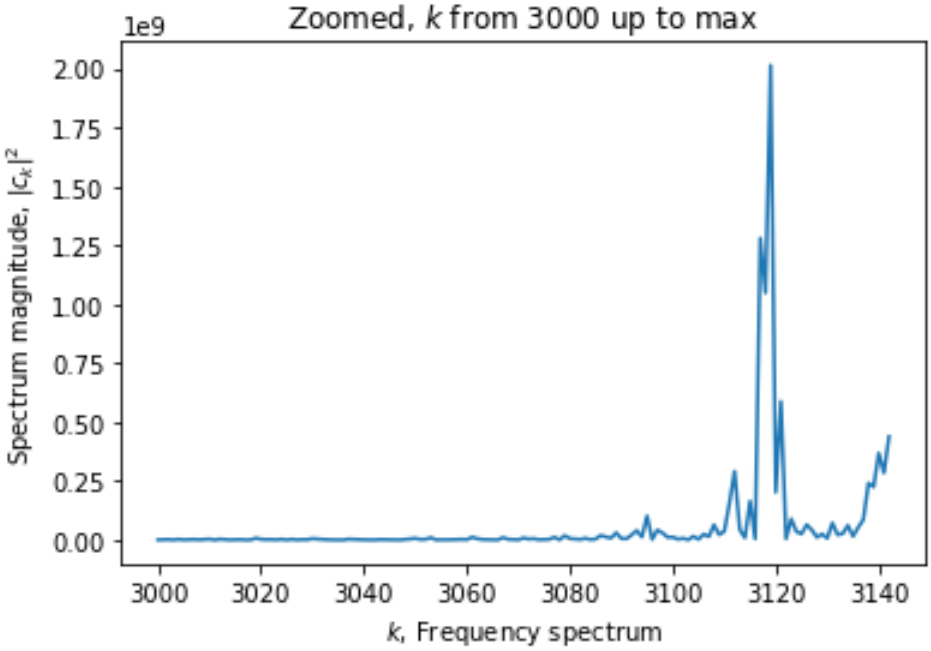
$$500 \div 4 = 125 \text{ months}$$

However, as we can tell the graph, there are tiny spikes within the big period, which are too small to be estimated by just eye-balling. So a Fourier transform need to be performed to reveal this small period.




I performed `numpy.fft.fft` on the sunspots data and transformed it into spectrum space. I ignored the huge spike at $k = 0$, since it is due to the data is all positive (the mean from the signal).
From the spectrum plot, we can tell that there are 2 main spikes at both ends of the plot. So I zoom into these two areas to determine the peak $k$ values.




From the graphs above, we can tell that the first peak happens at about $k = 24$, and the second is at around $k = 3118$.

From the Fourier transform formula:

$$c_k = \sum_{n=0}^{N-1} y_n \exp(-i\frac{2\pi kn}{N})$$

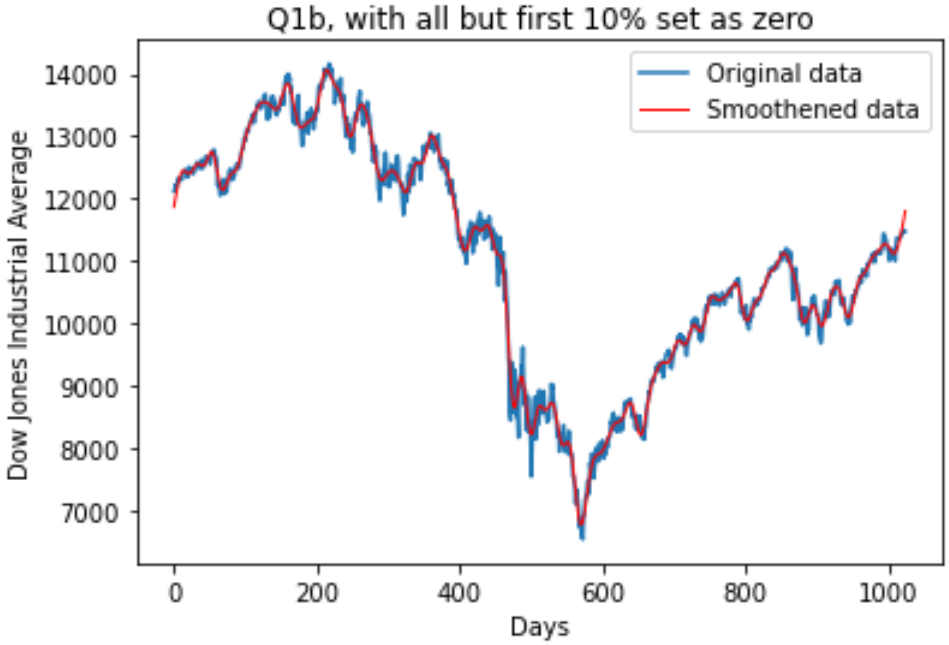
The period can be calculated from $k$ and $N$:

$$T = \frac{N}{k}$$

The $N$ value can be obtained by python function `len()`, which $N = 3143$ for our data.
So, for $k = 24$, the corresponding period is $T = \frac{3143}{24} \simeq 131$months. This confirms our eye-ball estimated period in previous part.
For $k = 3118$, the corresponding period is $T = \frac{3148}{3118} \simeq 1$months. This confirms the tiny spikes appear in the big period, which we can't eye-ball it.

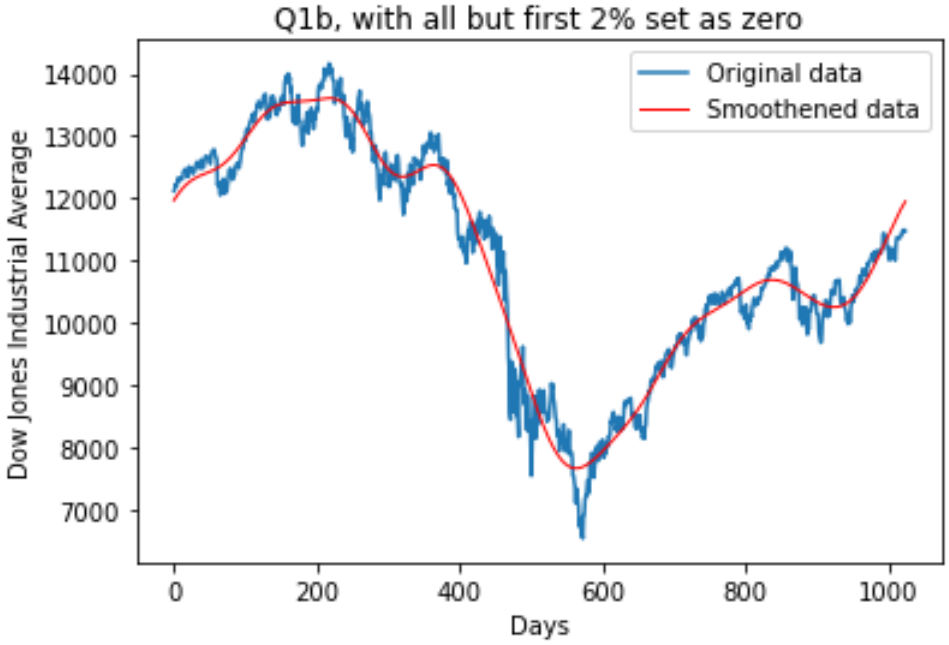
## 1.2 Q1b, Newman 7.4

So I perform `numpy.fft.rfft` on the original data from `dow.txt` and transform it into spectrum space.
Then I set all but the first 10% of the elements of the spectrum array to zero.
And I perform `numpy.fft.irfft` on the spectrum array to transform is back into real space to obtain a new set of data.
Finally, I plot the new smoothed data together with the original data:



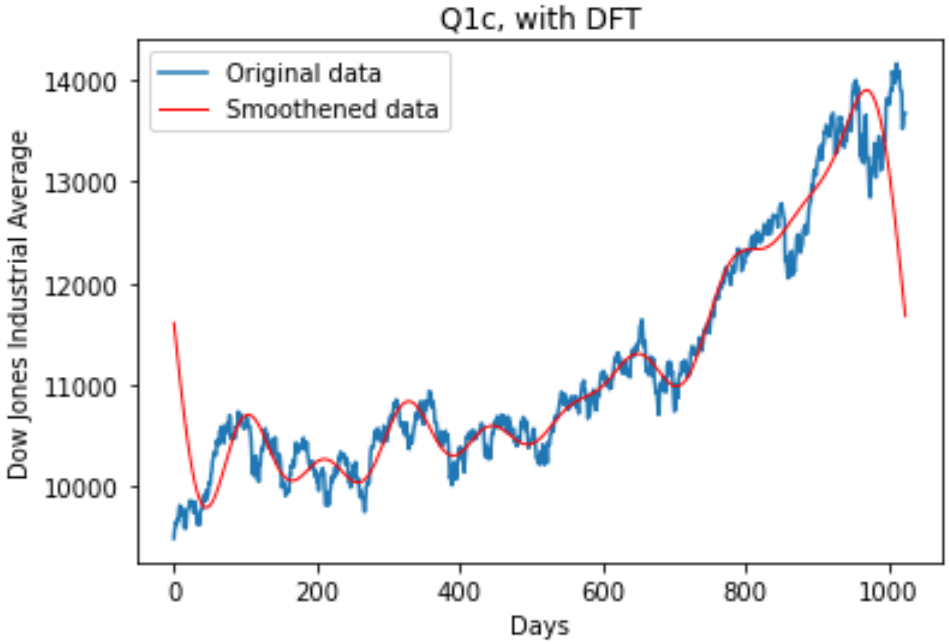

From the graph, we can tell that the smoothed data (red line) follows the trend

of the original data, however, those tiny spikes in the original data (blue line) have been removed.
This is because, in the spectrum space, we set elements with higher orders to zero. Thus, we removed the high frequency signals from the original data. The new data looks like been 'smoothed'.
The following graph is generated by set all but the first 2% of the elements of the spectrum array to zero:

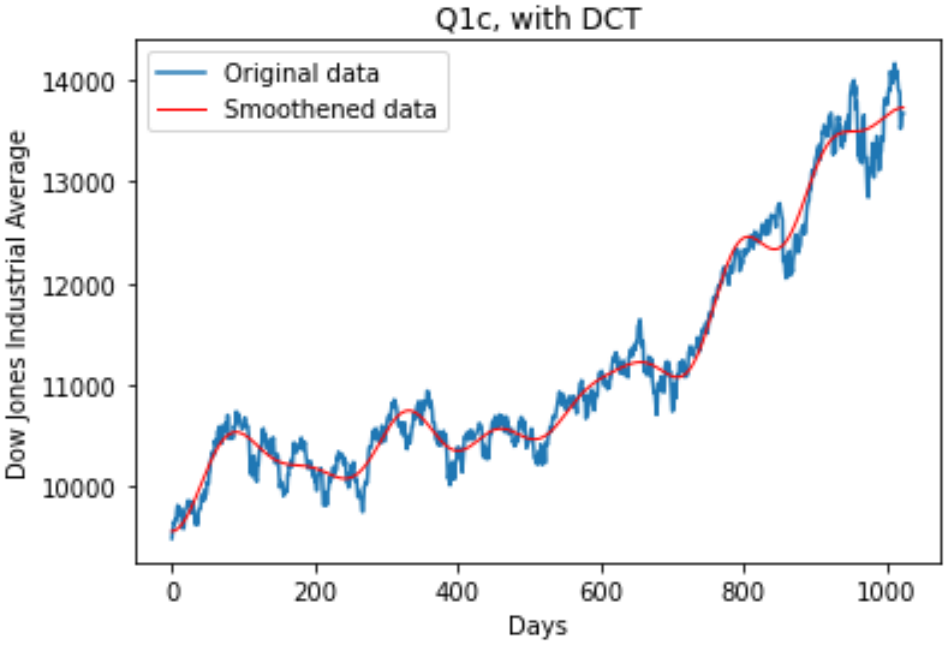



The curve is smoothed even further. It lost some details but still follows the trend of the original data. Since we keep the 2% low frequency signals.

### 1.3 Q1c, Newman 7.6

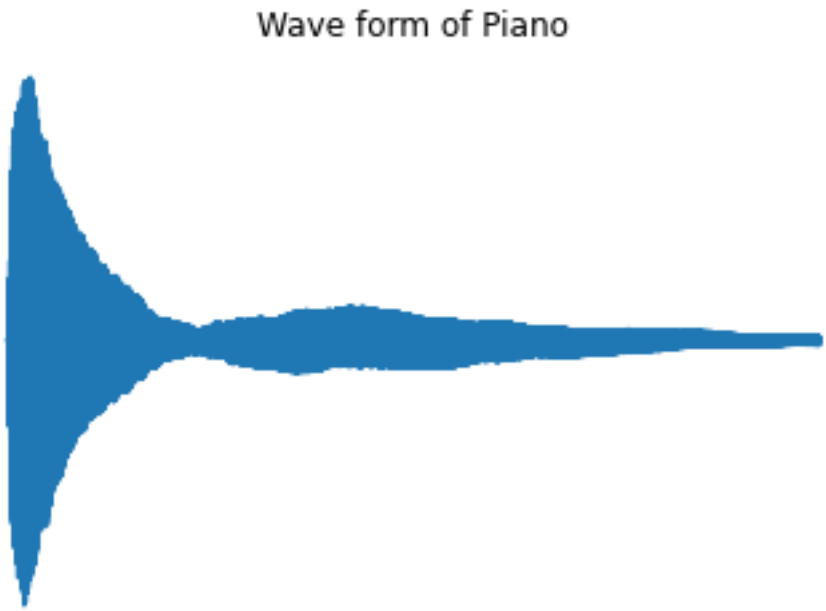


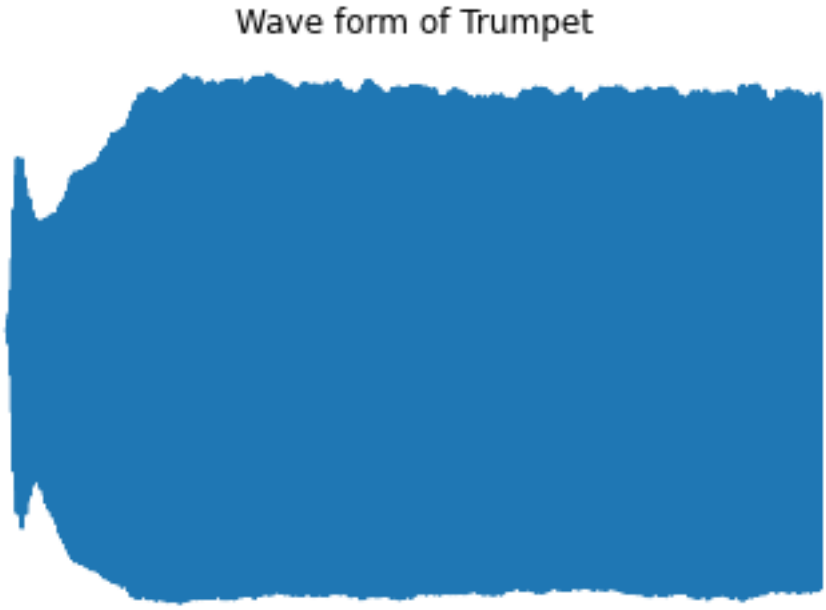

From the graph we can tell, if we perform DFT on a non-periodic function (the first value is not equal to the end value), discard high frequency signals, and transform back. The smoothed signal's start and end values are forced to be equal due to the nature of sine function.
This 'artifacts' can also be explained by Gibbs phenomenon. Fourier transform does not perform very well with discontinuous functions, especially at low orders. In our case, the non-periodic nature of the function can be considered as discontinuity at both ends. When we smooth the original signal, we remove higher orders, which increase the Gibbs phenomenon.




From the graph, we can tell that the ends are no longer forced to be equal thanks to the cosine function. However, a new 'artifact' introduced, the derivatives at the ends are forces to be zero now, also due to the nature of cosine function.
I don't agree with what is said in the lab handout, that the smoothed DCT 'struggles' more at following the raw data. The DFT fails pretty badly at the ends, while at the middle, both methods follows the original data pretty well at least according to my eye-balling. Perhaps the difference between DFT and DCT will be revealed at higher/lower orders.

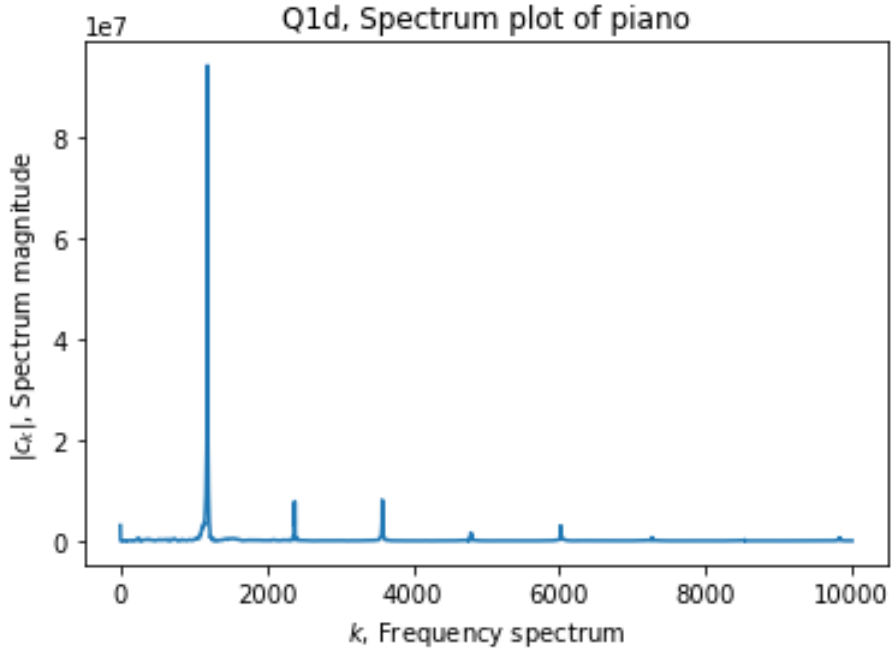
### 1.4 Q1d, Newman 7.3

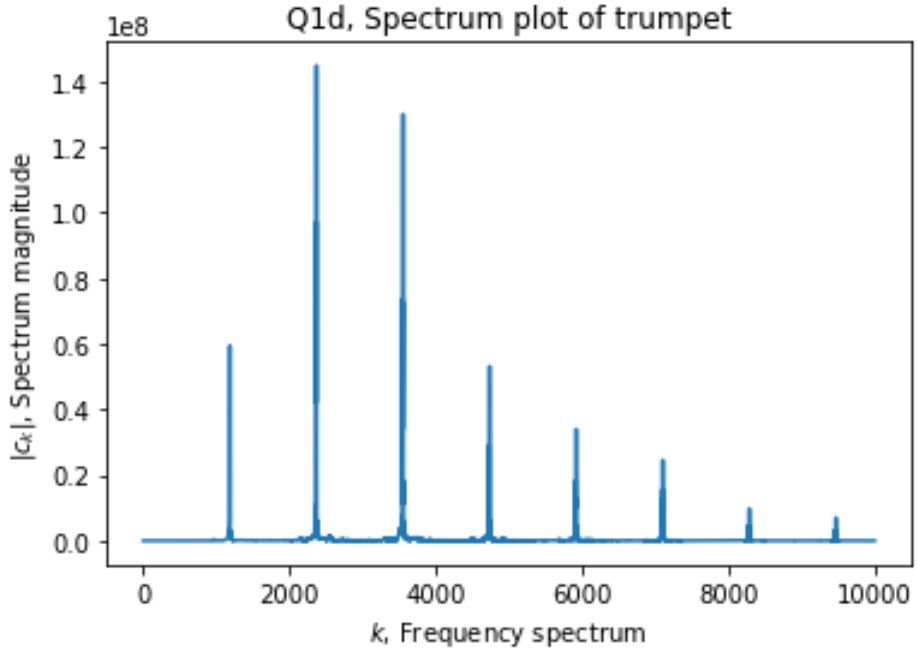



The graphs above are the wave forms of piano and trumpet. Please note that I hide the axis since the values at this stage don't have any physical meanings.

They are just relative values subjected to the recording standard. So I performed `numpy.fft.rfft` on both wave forms, and obtain the frequency spectrum plots:




From the spectrum plots, we can tell that the piano has only one main peak

at around $k = 1000$, the trumpet however, has several significant peaks which separate evenly.
This phenomenon is called 'Overtone', it is due to the nature of the instrument. The piano has a straight forward structure where the note is produced by vibrating a certain string. Thus, the piano has less overtone effect and only one main peak. The trumpet however, produce the note more complicatedly. It is a cooperation among lips, air, and bronze, which causes much more overtones. So the trumpet often sounds more round.
Each overtone peak is an exact multiple of the base frequency. That's why all the peaks are separated evenly in the spectrum plots.
To determine the exact peak $k$ values, I use a for loop to scan through the spectrum arrays:

```
for i in range(len(piano_spec)):
    if piano_spec[i] > 3e7:
        print('Piano, k:', i)

for i in range(len(trumpet_spec)):
    if trumpet_spec[i] > 0.3e8:
        print('trumpet, k:', i)
```

So I obtain the following peak $k$ values:

- Piano: $k = 1192$
- Trumpet: $k_1 = 1181$, $k_2 = 2367$, $k_3 = 3545$, $k_4 = 4731$

The peak $k$ values of trumpet confirm with the overtone theory, they are just multiples of the base peak $k_1$. To determine the note they are playing, we just need to look at the base frequency.
In our case, we have $N = 100000$ sample points, and the sample rate is 44.1kHz, the frequency is given by:

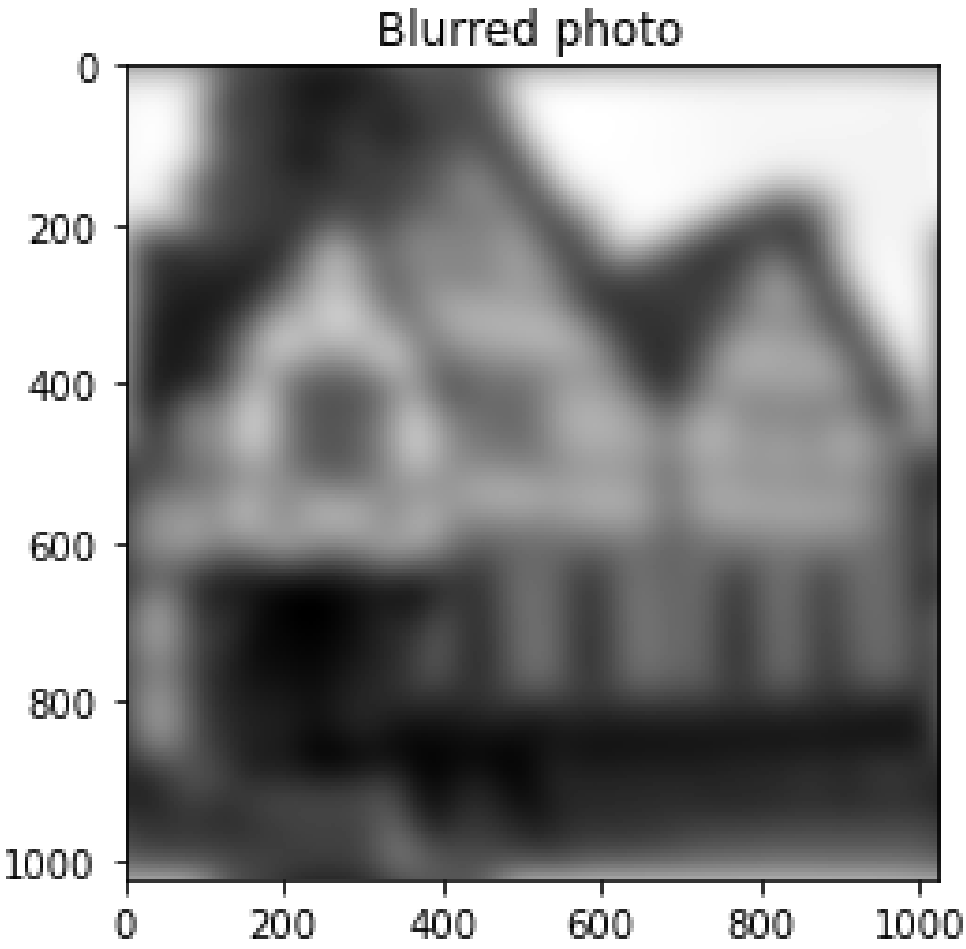
$$f = \frac{k}{N} \times 44.1\text{kHz}$$

So, for $k = 1192$, $f_{\text{piano}} = 525\text{Hz}$; for $k = 1181$, $f_{\text{trumpet}} = 521\text{Hz}$.
They are both playing the note C5 or simply, C note. In music, every octave doubles the frequency, say:

$$f_{C(n+1)} = 2 \times f_{Cn}$$

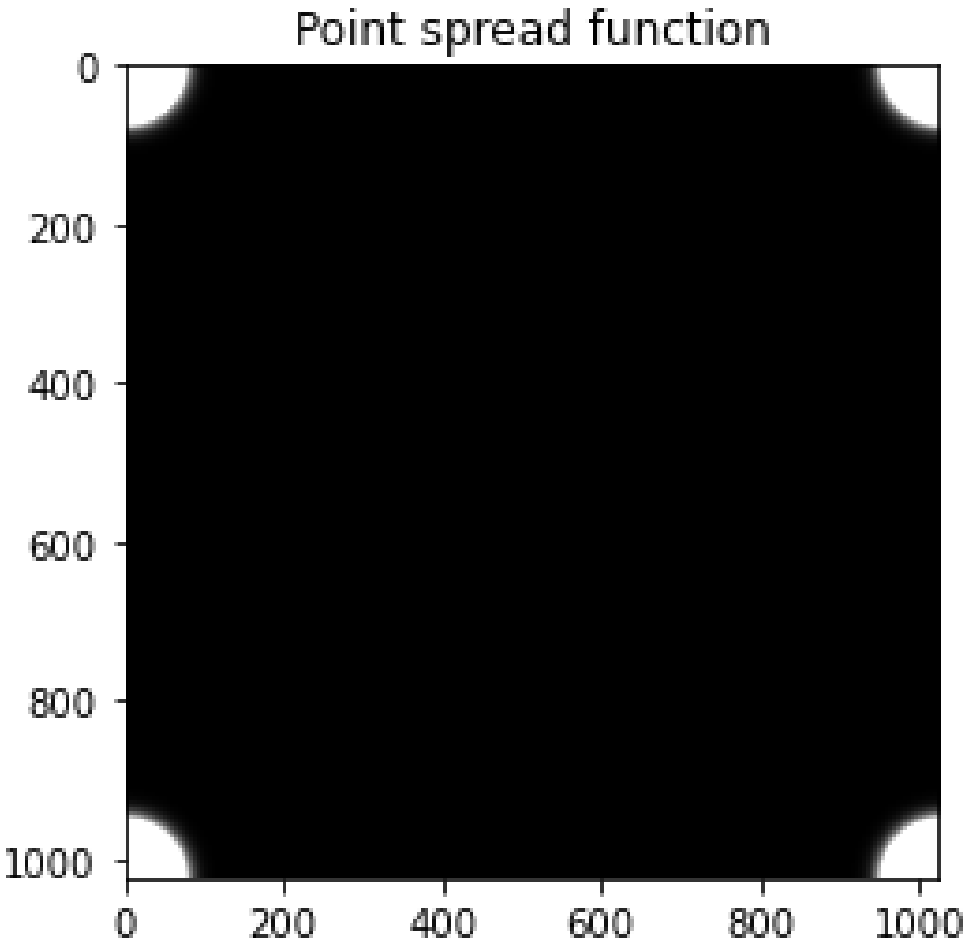
## 2 Question 2

### 2.1 Part(a)



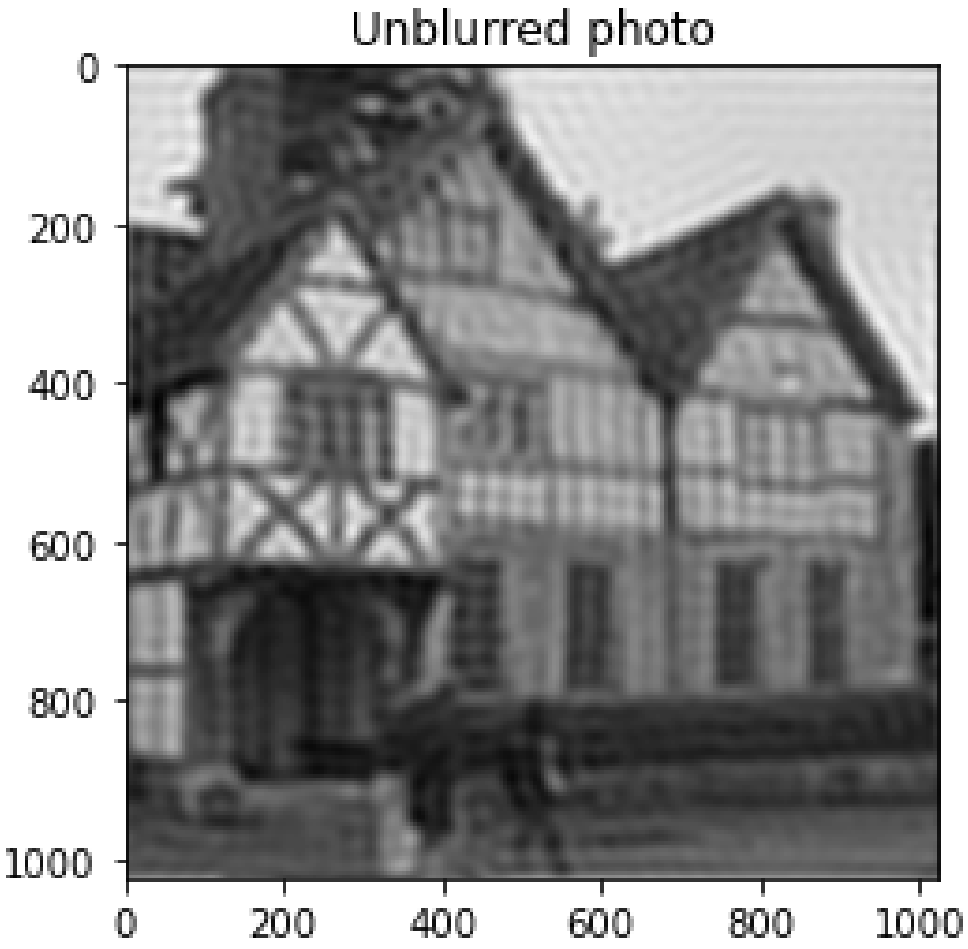

The blurred photo is attached above, generated from the "blurr.txt" file downloaded from Newman's website.

### 2.2 Part(b)




The point spread function is shown above, same as expected.

### 2.3 Part(c)




The unblurred picture is attached above. You can see more details can be identified from it, compared to the blurred one.