Pulse-Counting

What our program does:

Our program reads the data from the file then stores it into an array. This data contains peaks, which the program has to count. The biggest problem with counting the peaks is that data can contain noise, which affects the program's ability to accurately count the amount of peaks.

What filter we used:

To make the data more accurate and lower the noise within it, our program uses a butterl filter. This filter was decided upon through a process of elimination. Before choosing this filter, a weiner, savgol and smoothing filter was attempted. While these filters did change the data and eliminate some noisy data, it was decided that the savgol filter was doing the best job at filtering the data but this didn't allow us to change the cutoff frequency so we had to use a butter filter instead which is more flexible. The butter filter managed to remove most of the noise and maintain the overall shape of the data which is important for counting peaks.

What this filter does:

The butter filter is a filter that is built into python. It is known as a maximally flat filter because it endeavours to make the data as flat as possible. We used a lowpass butter filter which means that any high frequency above the cut-off frequency rolls down to zero . The order of a butter filter determines how fast the filter rolls off. For a butter filter you want to keep this rather low and since we still wanted peaks in our data we elected to only use order of 2 since higher orders seemed to smooth out the data too much. A digital butter filter also requires the cutoff frequency to be normalized, so we did this by times it by 2 then dividing it by the sampling frequency which was 10. Since we wanted our filtered signal to remain fairly close to our original dataset we chose our cutoff frequency to be 2, because it counted the peaks most accurately and wasn't to the original sampling frequency.

Cutoff frequencies:

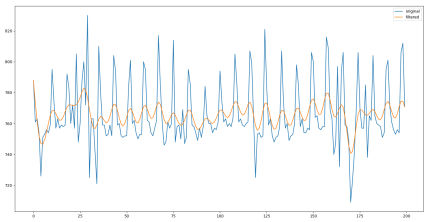
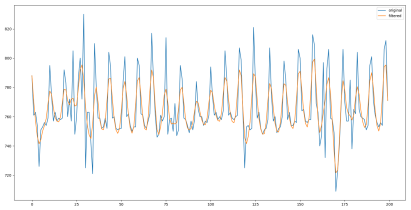
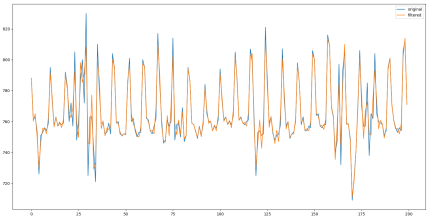
The higher the cutoff frequency, the higher the peaks of the filtered data. This is because with a higher cutoff frequency, higher frequencies remain in the filtered data because they're not removed or lowered by the butterfilter. It also means that the noisy peaks tend to remain and so the pulse counter counts more pulses than it should. Ultimately it means that the higher a cutoff frequency is to the original sampling frequency, 10 in our case, the closer to the original data the filtered signal will remain so very little noise will be removed.

pulse count			
Data	cutoff frequency 1	Cutoff frequency 2	CutOff frequency 4
pulsedata 1	24	25	30
pulsedata2	24	24	30
pulsedata3	23	23	25

pulsedata4	24	24	32
pulsedata5	22	24	30
pulsedata6	23	24	28

As the table shows with a lower cutoff frequency generally less peaks are counted. While there may not be much difference between 1 and 2, by jumping up to a cutoff frequency of 4 you can see how much more peaks are counted (in general). This is the result of more noise remaining. This means that the higher the frequency, the less accurate the peak counting will be because of noise. On the other hand it is important to not eliminate too much noise because then the data has lost more than just noise. For this reason it is important to find a cutoff frequency where noise has definitely been eliminated and where the new signal still represents the old signal. This can be found by trial and error or a good estimate can be about 0.2-0.3% of the original data.

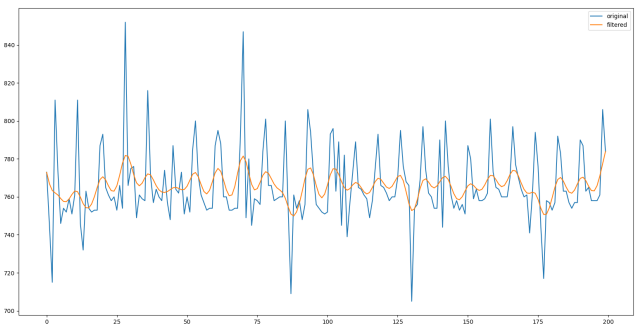
This is shown below with the left image showing a cutoff frequency of 1, middle 2 and right 4.

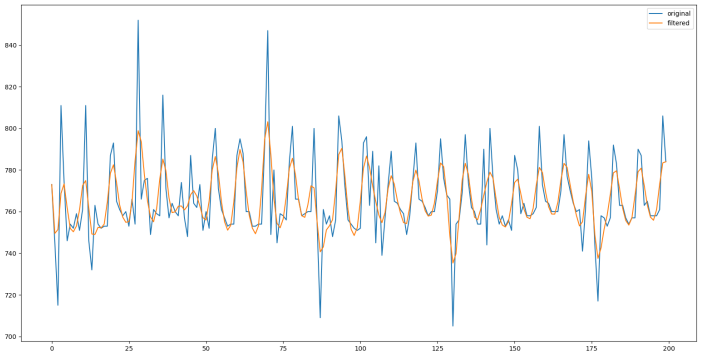
The cutoff frequency of 4 is closest to our signal so the filtered signal is very close to the original data and not much noise has been eliminated while a cutoff frequency of 1 is furthest from the original signal so it has eliminated the most noise and greatly lowered the heights of the peaks. A cutoff frequency of 2 was the best generalization for our datasets and this is why we chose to use it. Originally we were going to use one but in some cases such as pulsedata5, it appeared to eliminate too much noise.

Pulsedata 5 example: As you can see a cutoff frequency lowered the data too much and it wasn't able to accurately count peaks.

Cutoff frequency 1

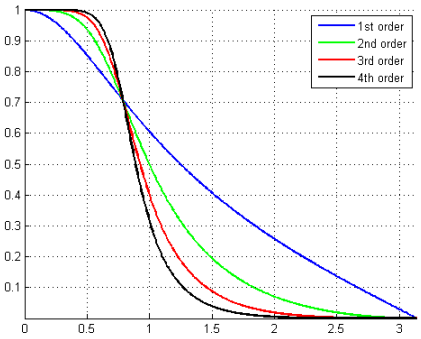


Cutoff frequency 2



How we used this filter:

The butter filter doesn't directly filter the signal and instead it returns numerator and denominator polynomials for another method to use. These polynomials are what the filtering method we use will try and fit the data to. The polynomials that it uses depend on the order that we input into the butter method.

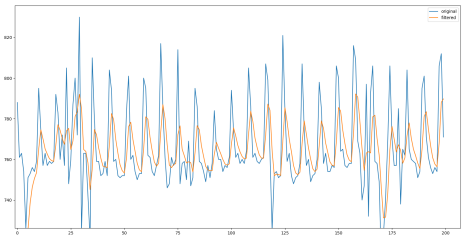


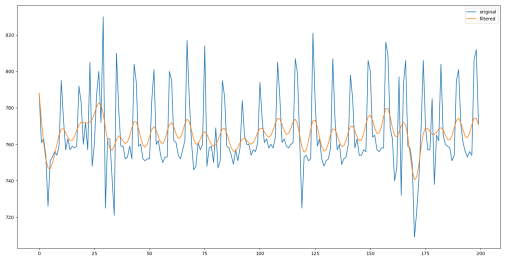
<- example of what different orders will try and fit the data to

These values are then put into a filtering method that takes 2 arrays (our numerator and denominator) and a signal to filter as parameters

Filtfilt vs lfilter:

After finding the numerator and denominator polynomial values, we still had to filter our actual signal (y). There were two good methods that could be used for this, the lfilter method or the filtfilt method. These methods take in the polynomials and original signal and then apply the filtering to the signal. Basically in our case they were applying the butter filter (in the form of its values) to our original signal. After testing both filtering methods we decided to go with the filtfilt method. This was because the filtfilt method applies the filter both forward and backwards to the original signal. This was more ideal in our case because it means that the start of the filtered signal starts at the same values as our original signal. This allows for cleaner filtered data to be returned. In comparison the lfilter just applied the filter forwards and so attempts to start from zero which creates an extra peak at the start of the data. As the filtering method (filtfilt in our case) travels along the signal it is fitting the data to our numerator and denominator values.



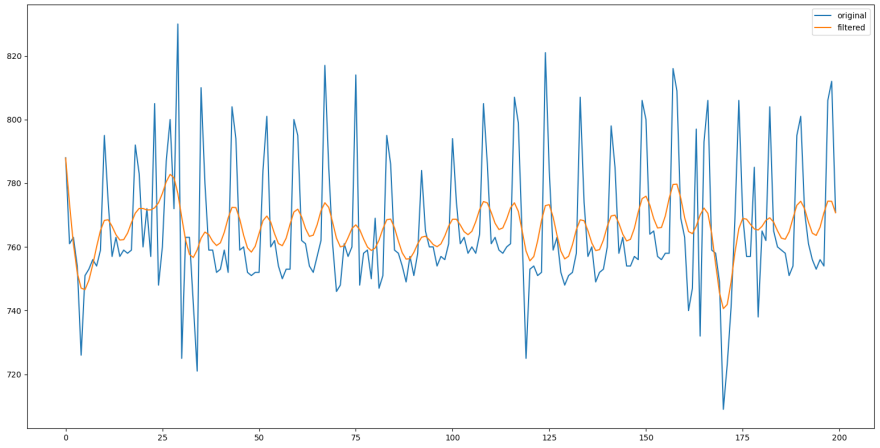


lfiltered: starts from zero	**Filtfilt: starts at where original signal starts**

How we counted peaks

After filtering, to count the actual amount of peaks we used another built-in Python method. This is called the argrelextrama method and it also takes in an array of data. We used this peak finder because it's compatible with other filters and because the amount of points we compare is setable.

Basically what this peak finding method is doing is taking a point, looking at N points on each side of our point. (N is chosen by us). And then returning the point which is the highest point in comparison to the points around it. This is our peak. The N value also acts as a sort of distance filter because it means that two peaks cannot be within N points of each other. This is useful for particularly noisy data.



Example of our filtered data

The blue line is the original signal

The orange line is the filtered signal

This was on pulsedata1

This counted 24 peaks at

11 19 27 36 43 52 60 67 75 84 93 101 108 116 125 133 142 150 158 165 175 182 190 197