EE3025 ASSIGNMENT- 1

Mohammed Sadiq - EE18BTECH11051

Download all python codes from

```
https://github.com/Sadiq0123/EE3025−IDP/tree/
    main/Assignment−1/codes
```

And Latex-tikz codes from -

```
https://github.com/Sadiq0123/EE3025−IDP/tree/
    main/Assignment−1
```

1 Problem

Modify the following code given in problem 2.3 with different input parameters to get the best possible output.

```
import soundfile as sf
from scipy import signal

#read .wav file
input_signal,fs = sf.read('Sound_Noise.wav')

#sampling frequency of Input signal
sampl_freq=fs

#order of the filter
order = 3

#cutoff frequency 4kHz
cutoff_freq=4000.0

#digital frequency
Wn=2*cutoff_freq/sampl_freq

# b and a are numerator and denominator
    polynomials respectively
b, a = signal.butter(order,Wn,'low')

#filter the input signal with butterworth filter
output_signal = signal.filtfilt(b, a, input_signal)
#output_signal = signal.lfilter(b, a, input_signal)

#write the output signal into .wav file
sf.write('Sound_With_ReducedNoise.wav',
    output_signal, fs)
```

2 Solution

As we can see in the code, the input parameters that can be modified are:

- Order of the filter
- Cutoff frequency
- Function for applying the filter

2.1 *Order of the Butterworth filter*

As we know that higher the order of the butterworth filter, steeper will be it's fall off rate. The figure 0 plots the amplitude response for different orders of butterworth low pass filter.

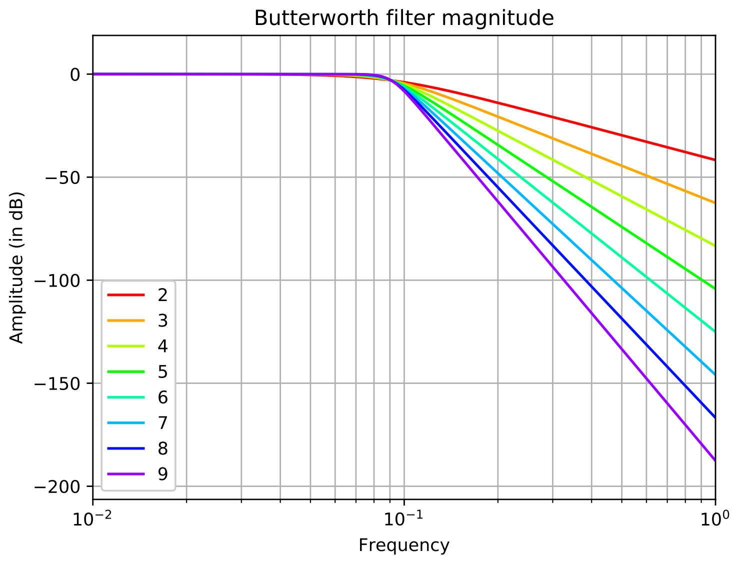

Fig. 0: Filter order Magnitude response

It would look like simply increasing the order of the filter would give better results. But higher order filters are unstable when simulating. This is because the values of coefficients of numerator go below the single-precision of float numbers as order increases. At order 4, the values of coefficients of numerator are 10^{-3}. Whereas at order 10, the order of values is 10^{-8}. This makes it clear that we can't simply select a very high order to filter out all the noise. As we know that the transfer function for an N order filter is:

$$H(\jmath\omega) = \frac{1}{\sqrt{1 + \left(\frac{\omega}{\omega_c}\right)^{2N}}} \tag{2.1.1}$$

As very high order filter is not possible, we can apply the more accurate lower order filter multiple times for better noise filtering. Now we try to establish that these two are essentially the same. Response for an order of 2N would be:

$$|H(\jmath\omega)| = \frac{1}{\sqrt{1 + \left(\frac{\omega}{\omega_c}\right)^{2*2N}}} \tag{2.1.2}$$

$$|H(\jmath\omega)|_{indB} = -10\log_{10}\left(1 + \left(\frac{\omega}{\omega_c}\right)^{4N}\right) \tag{2.1.3}$$

$$At\frac{\omega}{\omega_c} >> 1, |H(\jmath\omega)|_{indB} = -40N\log_{10}\left(\frac{\omega}{\omega_c}\right) \tag{2.1.4}$$

Now doing the same for an order N filter, but applying it twice we get:

$$|H(\jmath\omega)| = \left(\frac{1}{\sqrt{1 + \left(\frac{\omega}{\omega_c}\right)^{2N}}}\right)^2 \tag{2.1.5}$$

$$|H(\jmath\omega)|_{indB} = -20\log_{10}\left(1 + \left(\frac{\omega}{\omega_c}\right)^{2N}\right) \tag{2.1.6}$$

$$At\frac{\omega}{\omega_c} >> 1, |H(\jmath\omega)|_{indB} = -40N\log_{10}\left(\frac{\omega}{\omega_c}\right) \tag{2.1.7}$$

Clearly from 2.1.4 and 2.1.7, applying a low order filter twice gives the same result as the filter with double the order.

2.2 *Cutoff frequency*

To get better noise filtering, we also need to tweak the cutoff frequency. One method of doing this is to locate the location of all the peaks in the frequency magnitude plot, and choose the frequency at which the last peak occurs that has height above a fixed threshold. For the given soundfile, a threshold of 500 gives the last frequency with a peak as 2111. So, we filter all the frequency components above 2111 Hz.

2.3 *Function for applying the filter*

The code has 2 options for applying the filter **signal.filtfilt** and **signal.lfilter**. The latter function just passes the array through the butterworth filter. Doing so introduces a phase shift of $-\pi/2$. Whereas **signal.filtfilt** does more than that. It is actually a forward-backward filter(i.e. after applying the filter once, it reverses the signal and passes it through the filter once again). This helps cancel the phase shift caused earlier. So to get best results, we can apply **signal.filtfilt** more than once.

But there is one big drawback to using the forward-backward filter. That is that it is not causal. So, it can only be used on recorded signals that are already complete. For real-time filtering like in case of phone calls, we have no choice but to use **signal.lfilter** which injects a phase shift.

The code used for plotting the filter response and printing the coefficients can be found at-

```
https://github.com/Sadiq0123/EE3025−IDP/tree/
    main/Assignment−1/codes/generate_plot.py
```

The soundfile generated from this algorithm can be found at -

```
https://github.com/Sadiq0123/EE3025−IDP/tree/
    main/Assignment−1/soundfiles/
    Sound_optimized.wav
```

3 Results

To verify that the new parameters have filtered the signal, we plot the frequency response and time-series plot of original and filtered soundfiles. The time plot of the filtered signal is smoother due to lack of all the noisy high frequency components. The table below summarizes the differences between the parameters and outputs from the original code.

We observe from the table that the sum of components before the cutoff is slightly less for the modified code. This is because the components near cutoff also slightly reduce in magnitude as this is not an ideal filter. Before, noise had almost $\frac{1}{4}^{th}$ of the strength of useful signal. This goes down to less than 1% in the filtered signal.

Parameter	Original Signal	Filtered Signal
Cutoff Frequency	4000 Hz	2111 Hz
Order of filter	3	4 (applied multiple times)
Integral of FFT from 0 to cutoff	$1.57 * 10^7$	$1.48 * 10^7$
Integral of FFT after cutoff	$3.65 * 10^6$	$1.41 * 10^5$
Ratio of components after and before cutoff frequency	0.2318	0.0095

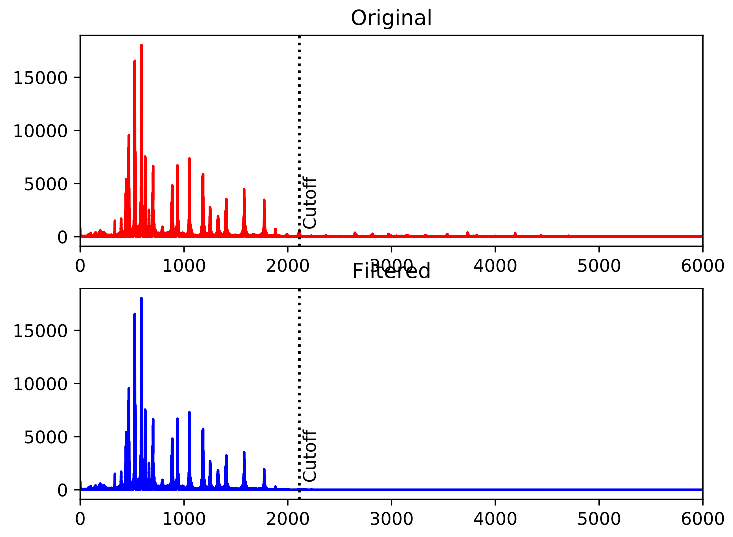

Fig. 0: Frequency Response

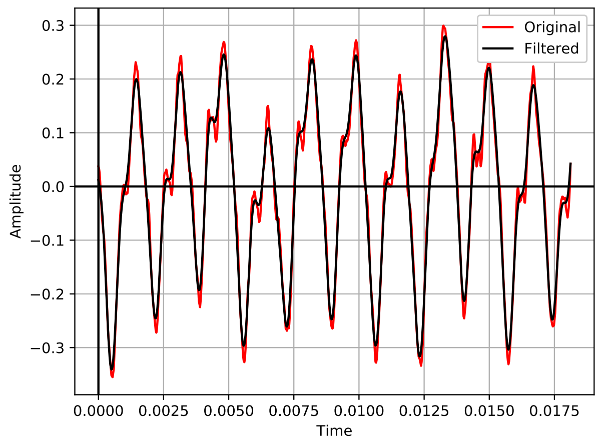

Fig. 0: Time series signal