BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

ISS - Project

Adrián Kálazi (xkalaz00)
January 4, 2021

Abstract

The calculations were implemented using **Python (3.5+)** with the following libraries:

- numpy - arrays, `fft`, `ifft`
- scipy - `lfilter`
- soundfile - wav reading/writing
- matplotlib - plotting

The implementation consists of a Python module and a launcher script (`main.py`).
Running the script:

- On a system without the required libraries - execute `./run-venv.sh` in the `src/` folder.

  This will setup a virtual environment, install the requirements and run the script.
- On a system with the required libraries - run `python3 main.py` or execute `./main.py` directly.

Solution

1. The audio was recorded with `KWave` in 16-bit resolution at 48 kHz.

   The sampling rate was later scaled down to 16 kHz using `ffmpeg`

   Durations of recorded signals:

File	Samples	Seconds
maskoff_tone.wav	27436	1.71
maskon_tone.wav	36696	2.29

2. The sentence recording process was the same as above

   Durations of recorded signals:

File	Samples	Seconds
maskoff_sentence.wav	41162	2.57
maskon_sentence.wav	43312	2.71

3. Firstly, one second long signals were extracted from the wav files.

   Then the DC bias has been removed and the signals were normalized.

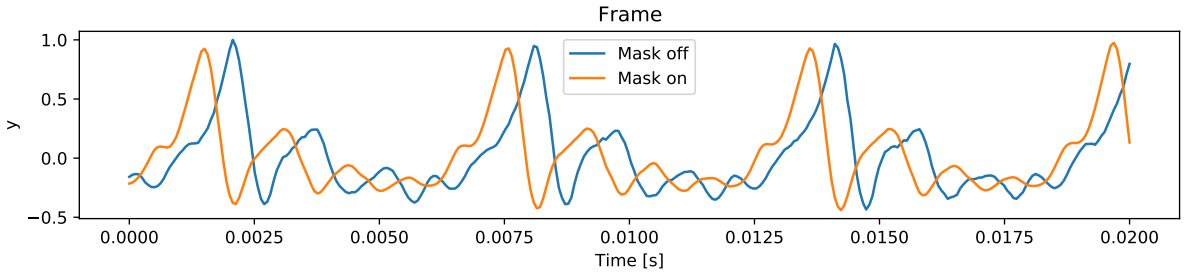
   The base tasks are calculated with 20 ms frames extracted from the signal, resulting in a total of 99 frames (the last half-frame is ignored).

   We later recreate the frames with a 25 ms duration for task 15.

   Calculation of the frame size:

   Frame duration $t_F = 20\,ms$
   Sampling frequency $F_s = 16\,kHz$

   Frame size (in samples): $n_F = t_F \times F_s = 20\,ms \times 16\,kHz = 320\,samples$



4. The correlation function implementation is located in `iss.operations.correlate_frame` and the function is also used in task 15 for correlation.

   Mean and variance of base frequencies:

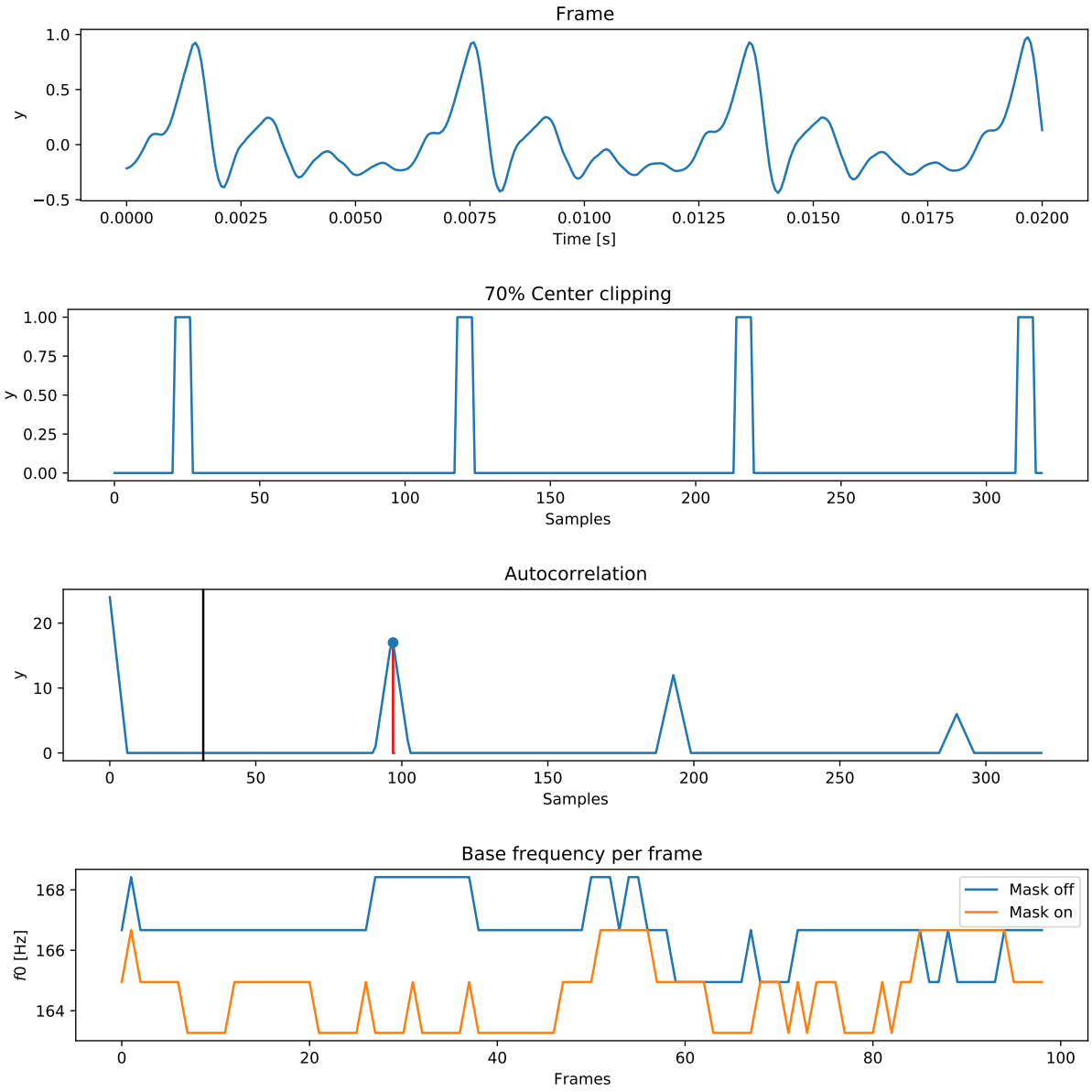
Signal	**Mean** $[Hz]$	**Variance** $[Hz]$
mask off	166.638	1.094
mask on	164.563	1.503

   Q: How can the f_0 change be reduced for ± 1 errors of the lag?

   A: There are two possible solutions that come to my mind to the mentioned problem:

   - In our implementation, lag was defined as `np.argmax(autocorrelation_array)` and therefore it was an integer.

     An alternate implementation would be to approximate lag as a weighted arithmetic mean from the surrounding indices and their values which would make it a floating point number resulting in better precision. This method would eliminate rapid changes of f_0 in most cases.
   - Autocorrelation with higher resolution - this would eliminate the rapid changes in f_0 but would require a longer computation time.



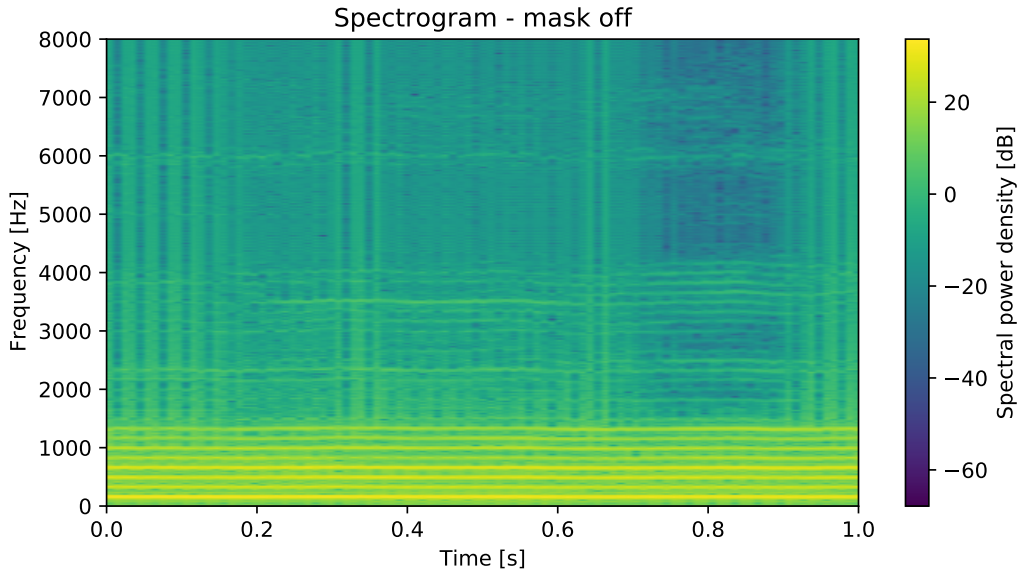
After center clipping and autocorrelation, the resulting base frequencies were examined and we came to the conclusion that the frequencies are ”close enough” for them to be used safely for designing our filter.

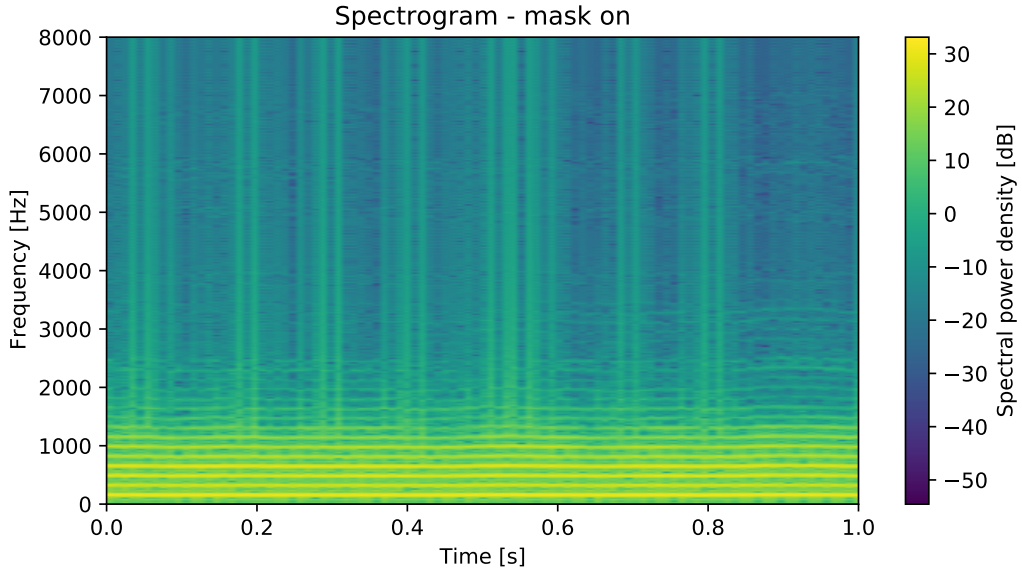
Task 11 also implements an alternative where only frames with matching base frequencies are used for determining the frequency response.

5. For calculations, the fast fourier transform function (`fft` from `np.fft`) was used

   Example DFT function implementation

```
def dft(x):
    X = []
    N = len(x)
    for k in range(N):
        a = 0
        for n in range(N):
            a += x[n] * cmath.exp(-2j * cmath.pi * (k/N) * n)
        X.append(a)
    return X
```





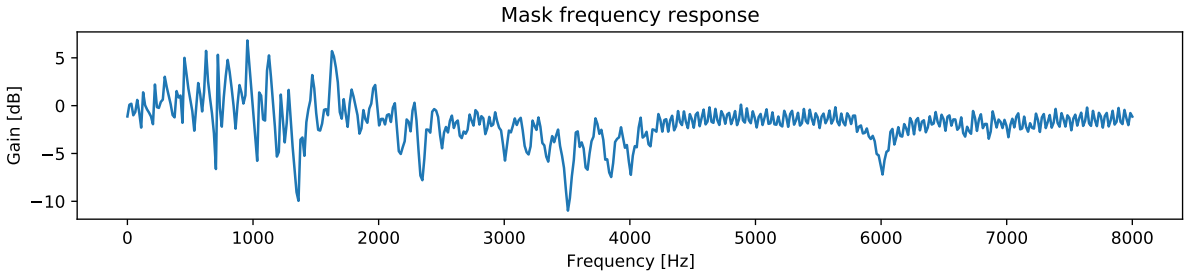
The spectrograms show a bigger presence of high-frequency noise in the ”mask on” than the ”mask off” signal

6. Frequency response

   For the frequency response we need to calculate the the mean from all frames.

$$H(e^{j\omega}) = \frac{\frac{1}{N}\sum_{n=0}^{N-1} B_n(e^{j\omega})}{\frac{1}{N}\sum_{n=0}^{N-1} A_n(e^{j\omega})}$$

N - frame count, A_n, B_n - spectrums for frame n

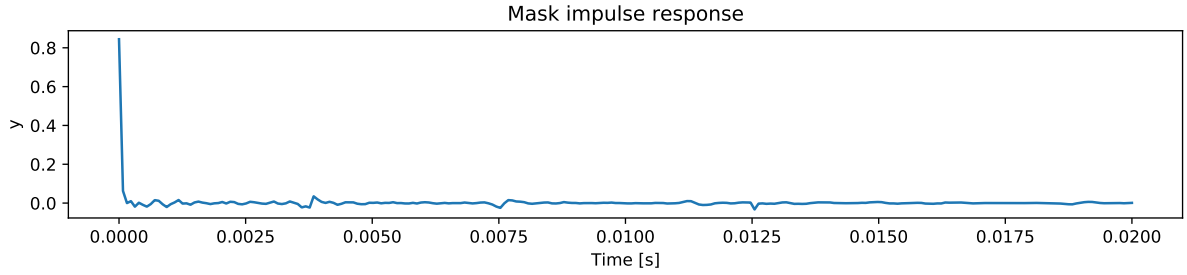


From the filters frequency response we can see that the gain oscillates frequently over the lower half of the frequency range, which may be caused by the varying base frequency of the input signals.

Around 6 kHz we can see a sudden negative gain spike, so our filter supresses these frequencies.

7. Impulse response

   The inverse fourier transform (`ifft` from `np.fft`) was applied to the frequency response calculated in the previous step.

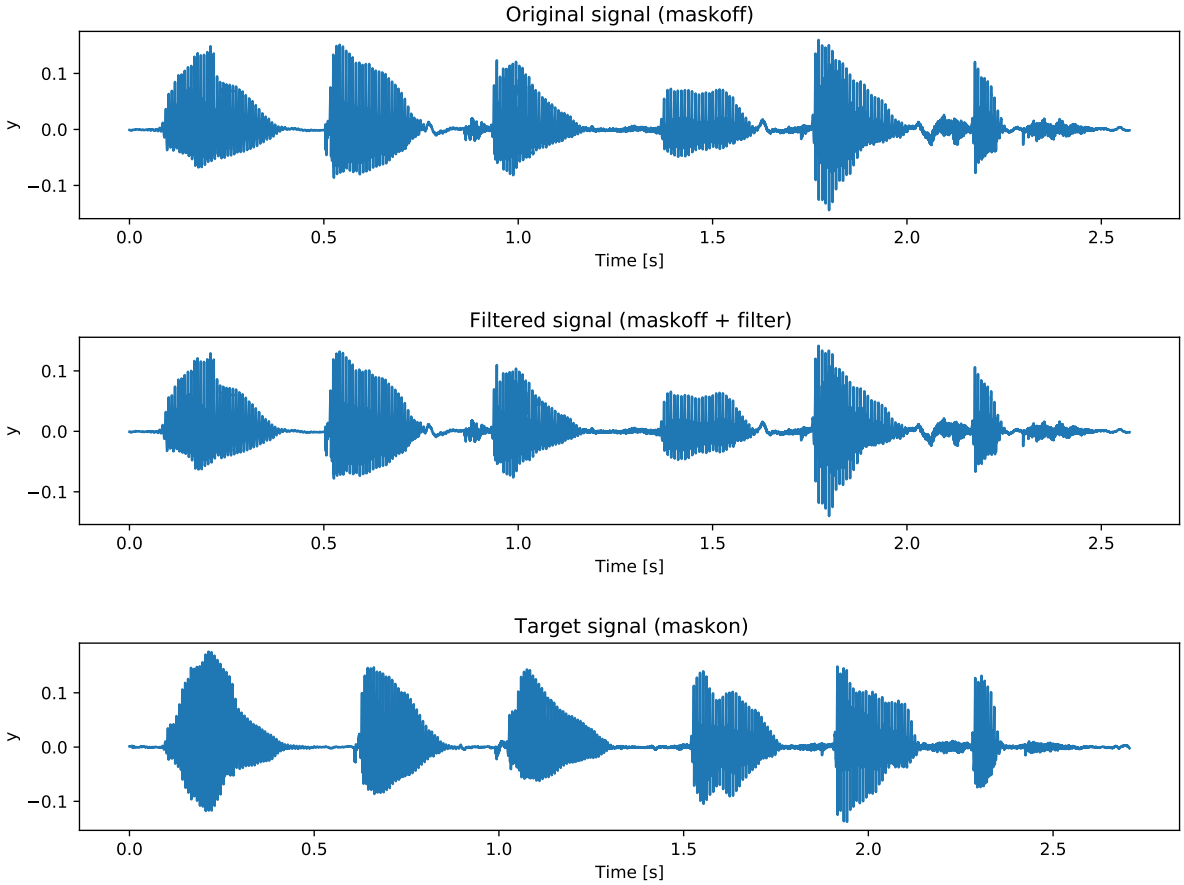


IDFT function implementation

```
def idft(X):
    x = []
    N = len(X)
    for n in range(N):
        a = 0
        for k in range(N):
            a += X[k] * cmath.exp(2j * cmath.pi * (k/N) * n)
        x.append(a / N)
    return x
```

8. Filtered signal

   We applied the signal using `lfilter` from `scipy.signal`



   From the above figures we can see that the target signal and the filtered signal are different in most aspects.

   This is probably because of the differences in recording (we can see that the tones between the ”mask off” and ”mask on”) recordings are not very similar.

   The filter seems to reduce a little bit of low frequency noise, but it’s hardly noticeable.

   Overall the filter seems to ”not do much”, although in the recordings we can hear a little bit of a ”dampening” effect.

9. Conclusion

   The biggest problem we encountered was that we didn’t have access to an example signal on which our calculations could be tested and this caused our results to be very dependent on our own recordings.

   This introduced quite a few problems as we didn’t know if our calculations are right at any point.

   The filters weird behaviour may also be caused by a smaller difference between the tone recordings that the filter has been constructed from.

   Even though our calculations should be right, the base results aren’t good enough to be considered ”acceptable”.

   Our additional tasks yielded a little bit better results, which in the case of an applied phase shift could even be considered as ”somewhat acceptable”.

Solution - additional tasks

All of the next tasks recreate the signal frames, so that each task can be compared to the results from the base solution

10. Overlap-add

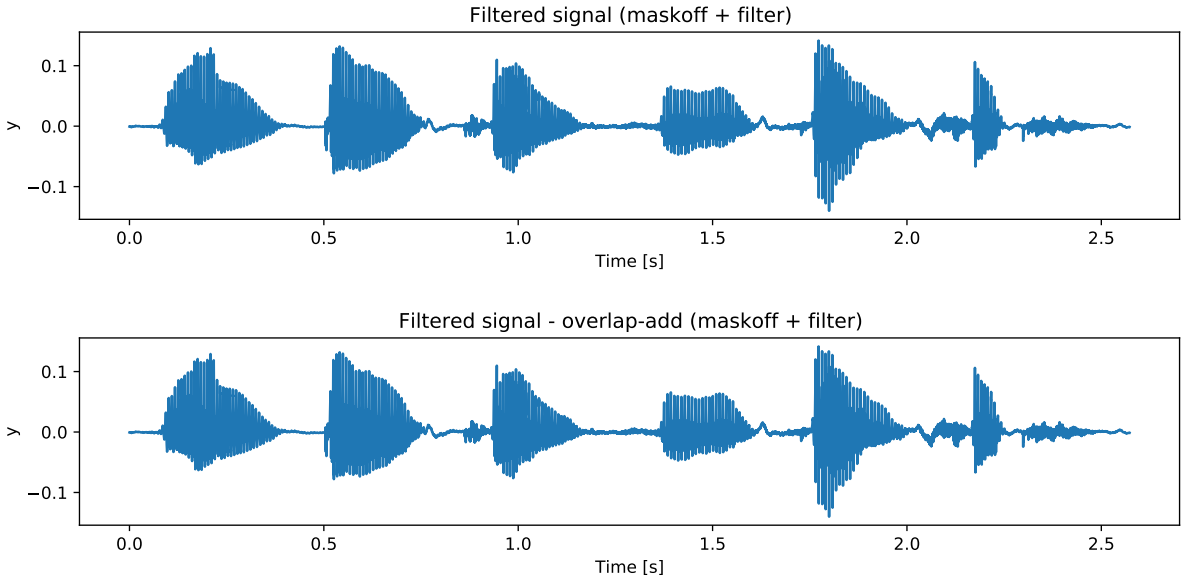
    The overlap-add method implementation:

```
def overlap_add(data, flt):
    # L is chosen such that N = L+M-1 is an integer power-of-2
    N = 2 << (flt.shape[0] - 1).bit_length()
    step = N - flt.shape[0] + 1
    samples = data.shape[0]

    flt_fft = fft(flt, n=N)

    filtered_data = np.zeros(samples + N)
    for n in range(0, samples, step):
        filtered_data[n:n + N] += ifft(
            fft(data[n:n + step], n=N) * flt_fft
        ).real

    return filtered_data[:samples]
```

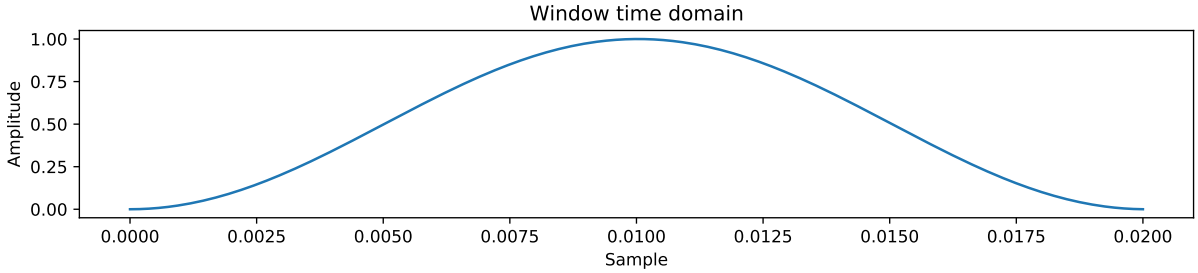


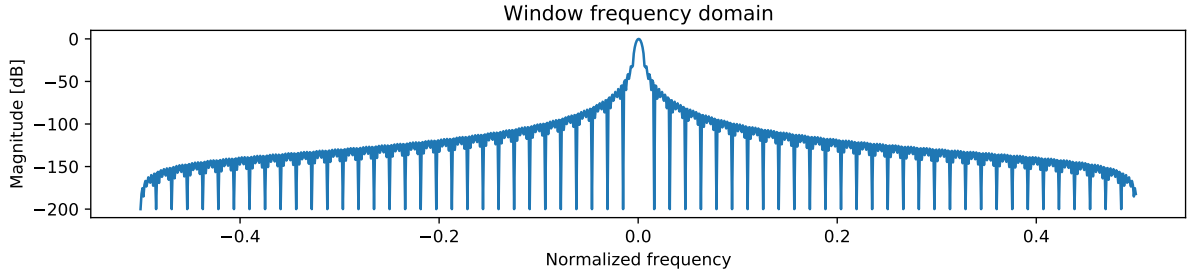
From the above figures, we can see that both filtering methods produce very similar results.

11. Window function

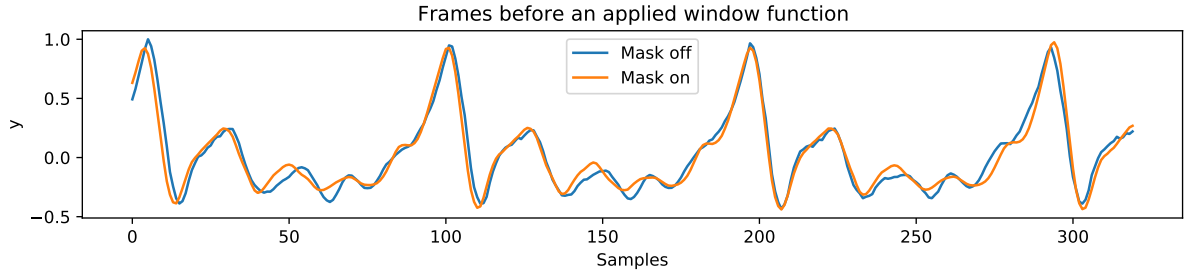
We created the window using `get_window` from `scipy.signal`

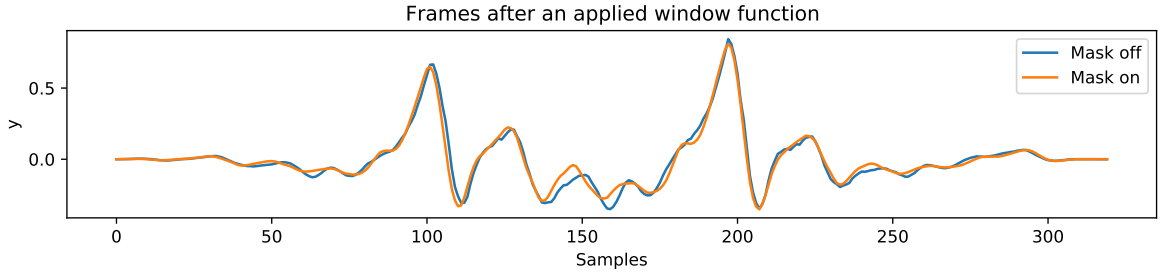
The slected window function that has been used was the ”Hann window” with the following properties:



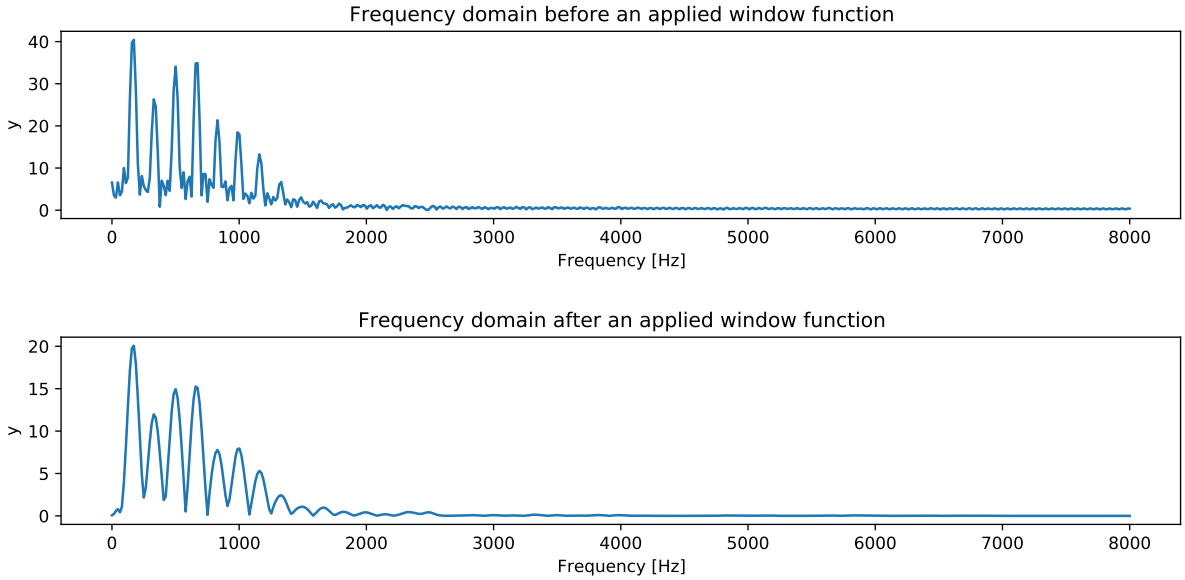


Phase shift (from task 15) has been applied before the window function, this choice was made because it shows better how the function behaves in the next figures.





Comparison of specral densities before and after applying the window function:

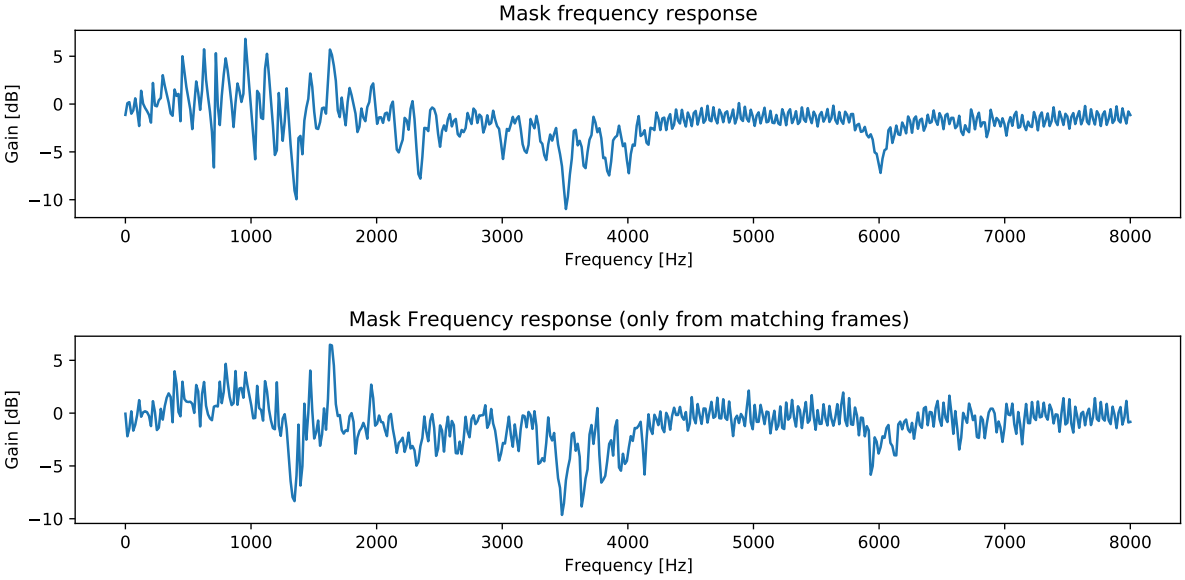


From the above figures we can see that the window function supresses the frequencies on the edge of the spectrum and the resulting frequency spectrum is overall smoother.

Window functions are useful for distributing the spectral leakage introduced by examining longer signals.

13. Filter constructed from only frames with matching base frequencies

    The needed frames were collected during task 4 and the frequency response (which is normally constructed in task 5) was instead contructed from these selected frames only.

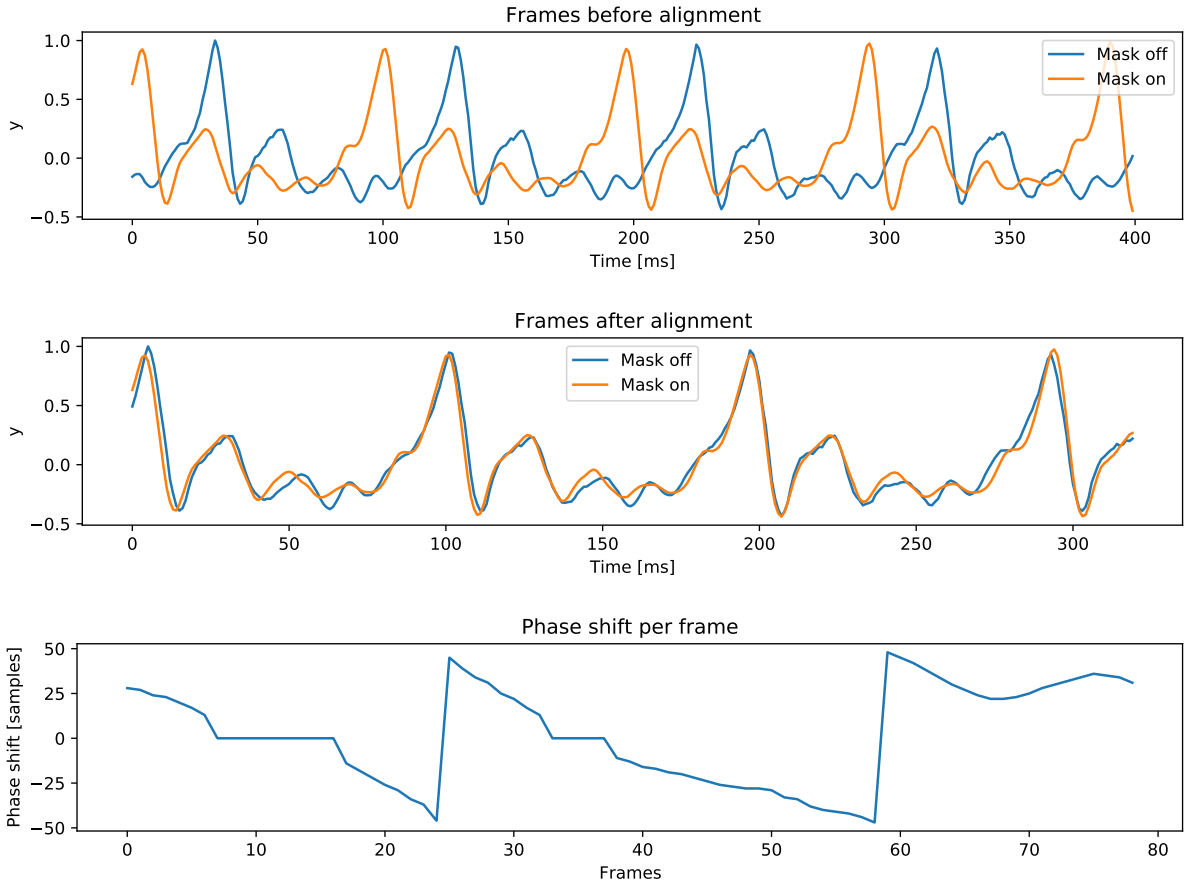


From the above figures, we can see that using only frames with matching base frequencies distributes sudden spikes in the frequency response, such as the spike around 6 kHz.

15. Phase shift

The advised operations were performed on the frames, including center clipping, correlation both ways and determining the needed shift in the righ direction (with the smaller number of samples).

Next the frames were cut so their phases match and a final cut was made to shorten every frame to 20 ms.



From the above graph, we can see that the required phase shift decreases almost linearly - this is caused by the frequency of the ”mask on” signal being a little bit lower than the ”mask off” signal

In the best case scenario (when both signal frequencies match) the phase shift would be constant.

Q: When we take the sum of phase shifts from both sides, we can see that it resembles the lag in auto-correlation, what is the reason for this behaviour?

A: This is due to the coefficients (and lag) representing a shift at which both signals are the ”most similar”.

In autocorrelation this results in the lag representing the number of samples until the next period occurs.

Because autocorrelation is done on only one signal, the shortest shift at which the signal is similar to itself is its own period.

The autocorrelation coefficient (and therefore lag) represents one period.

In cross-correlation the coefficient represents the shortest shift in a certain direction, but because the signals have similar frequencies and the cross-correlation is done from both sides, summing up these two values results in them being equal to the signals period T in samples.

Both cross-correlation coefficients represent a part of the period (e.g. 1/3 and 2/3) which summed up results in one period.