BRNO UNIVESITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

# Signals and Systems

Project - analysis and filtering of audio signal

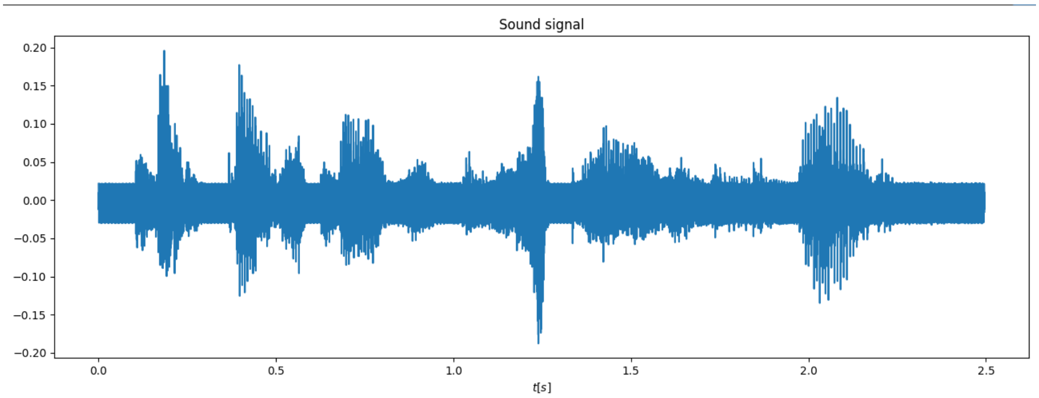
January 7, 2022 Alexander Rastislav Okrucký (xokruc00)

# Basic assignment

For this project I used Python 3.9.9 and Jupyter Notebook 6.4.6.

## 1 Basics

Signal length = 2.496 s
Signal length in samples = 39936
(minimum, maximum) = (-0.18768310546875, 0.195892333984375)
Maximum and minimum is already normalized, so values are in range $< -1, 1 >$.
I used *soundfile.read* function for reading the input signal.

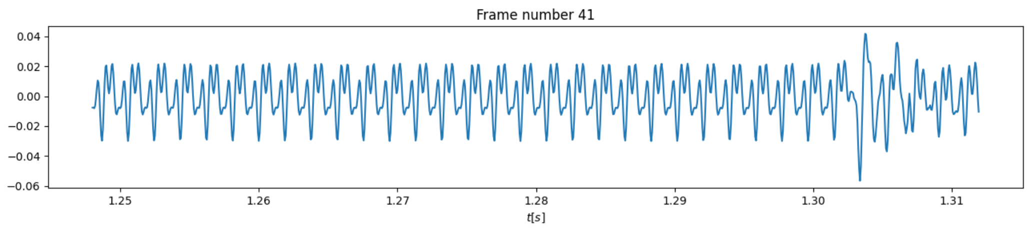



**Graph of input signal *xokruc00-original.wav***

## 2 Preprocessing and Frames

My input signal is already normalized so I do not have to. If I would normalizing my signal I would devide whole signal by maximum of this signal. This normalization method I am using in part 7 to normalize my generated cosine signal.

After splitting my signal to frames I have 76 frames. I chose frame number 41 as a nice vocal sound frame. I used special engineering skills for that. I randomly picked frame 40 then looked at frames 39 and 41. Frame 41 was the best one.

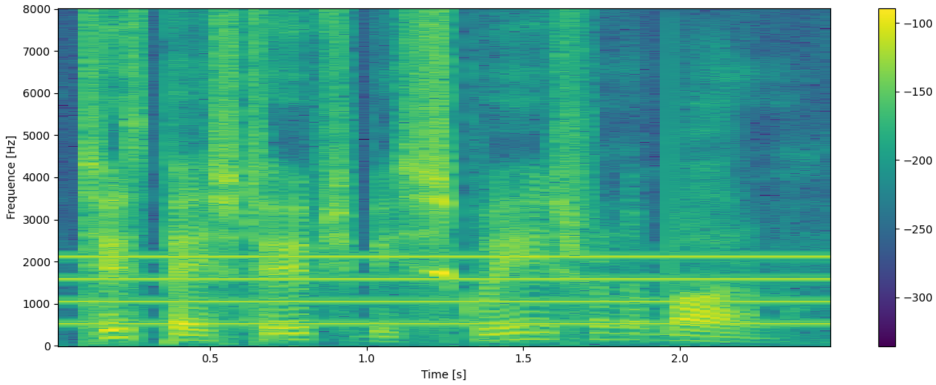



**Graph of frame 41**

# 3 DFT

```
def myDFT(function):
    N = function.size
    result = []
    num = 0
    k,n = 0,0
    for k in range(1024) :
        for n in range(1024) :
            realPart = math.cos(((2 * math.pi) / N) * k * n)
            imagPart = math.sin(((2 * math.pi) / N) * k * n)
            complexNum = complex(realPart , -imagPart)
            num += function[n] * complexNum
        result.append(num)
    return result
```

# 4 Spectrogram

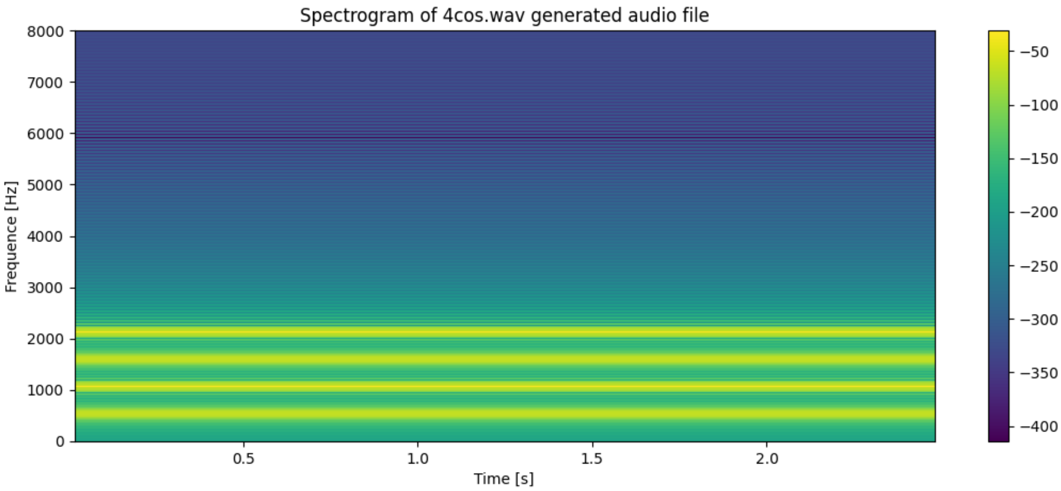



**Spectrogram of input signal**

# 5 Determination of interfering frequencies

For determination of interfering frequencies I used function *find peaks()* from scipy library. I used as input fft output od frame 41.
f1 = 531.25 Hz
f2 = 1062.5 Hz
f3 : 1593.75 Hz
f4 : 2125.0 Hz
I found out that the interfering frequencies are harmoniously related. I used for it my code.
f2 : 1062.5 == f1 * 2 : 1062.5
f3 : 1593.75 == f1 * 3 : 1593.75
f4 : 2125.0 == f1 * 4 : 2125.0

# 6 Signal generation




**Spectrogram of generated cosine signal**

```
samples = []
for i in range(data.size):
    samples.append(i/fs)

outCos1 = np.cos(2 * np.pi * f1 * np.array(samples))
outCos2 = np.cos(2 * np.pi * f2 * np.array(samples))
outCos3 = np.cos(2 * np.pi * f3 * np.array(samples))
outCos4 = np.cos(2 * np.pi * f4 * np.array(samples))

outCosSum = outCos1 + outCos2 + outCos3 + outCos4

norm = outCosSum / outCosSum.max()

soundfile.write("../audio/4cos.wav", (norm * np.iinfo(np.int16).max)
.astype(np.int16), fs)
```

I normalized generated signal by deviding generated signal by its maximal value.

# 7 Filter design

I have chosen the third way how to design filters. I created four bandstop filters, by using *buttord()* and *butter()* functions. To make my life easier I used my own function for creating theese filters.

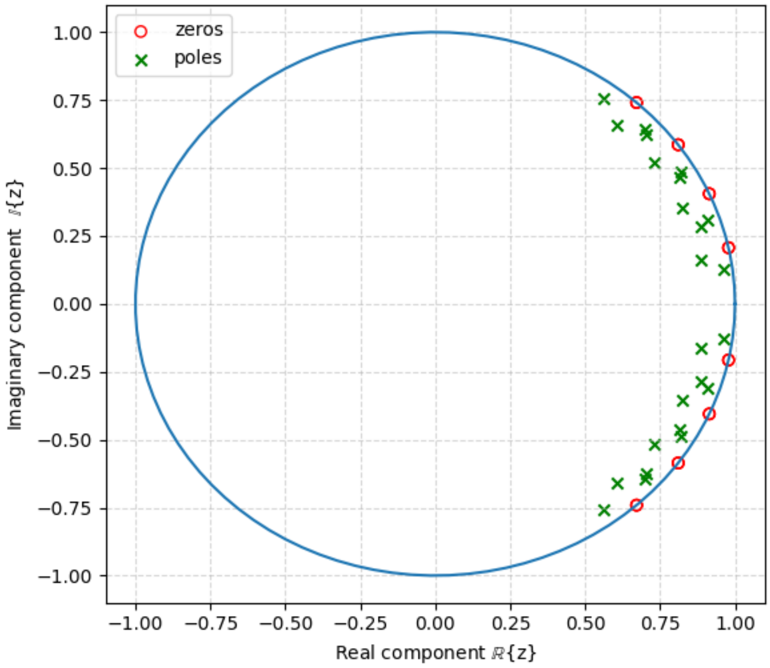
```
def createFilter(frequency):

    N, Wn = buttord([(frequency-90)/(fs/2), (frequency+90)/(fs/2)],
                    [(frequency-30)/(fs/2), (frequency+30)/(fs/2)], 30, 50)

    b, a = butter(N, Wn, 'bandstop')
```

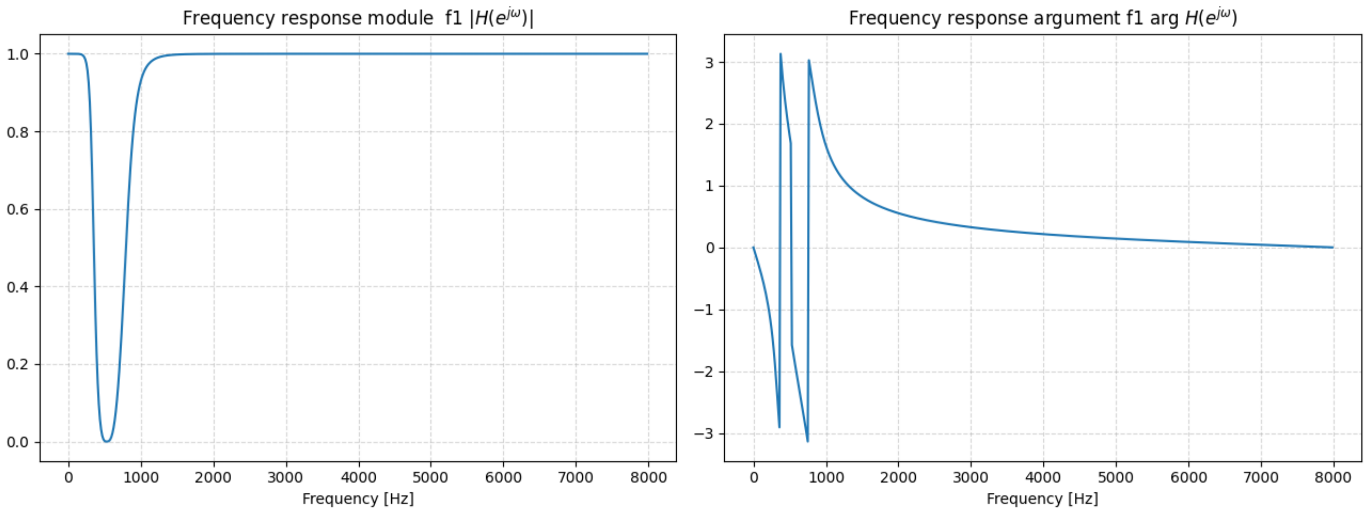
```
    return b, a
```

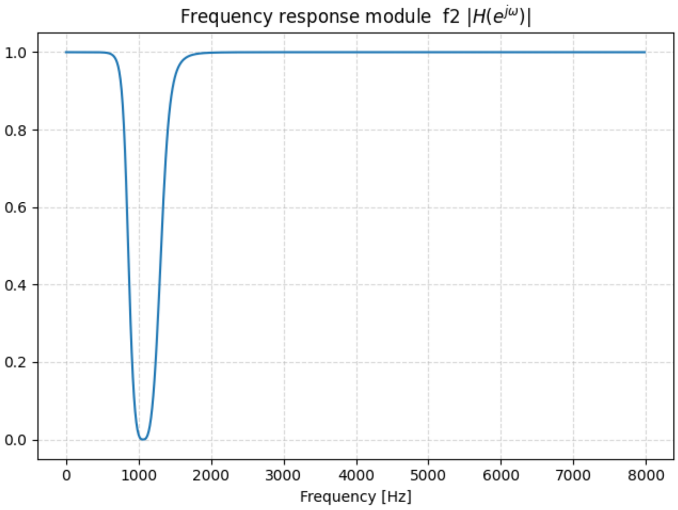
# 8 Zero points and poles

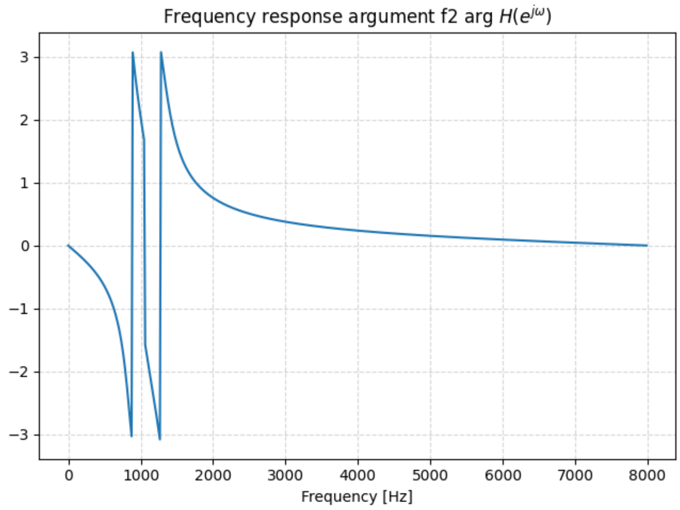


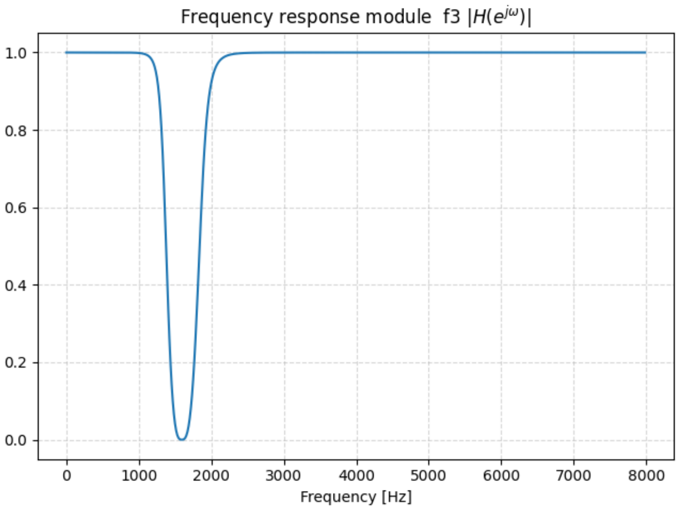

**Graph of zeros and poles**

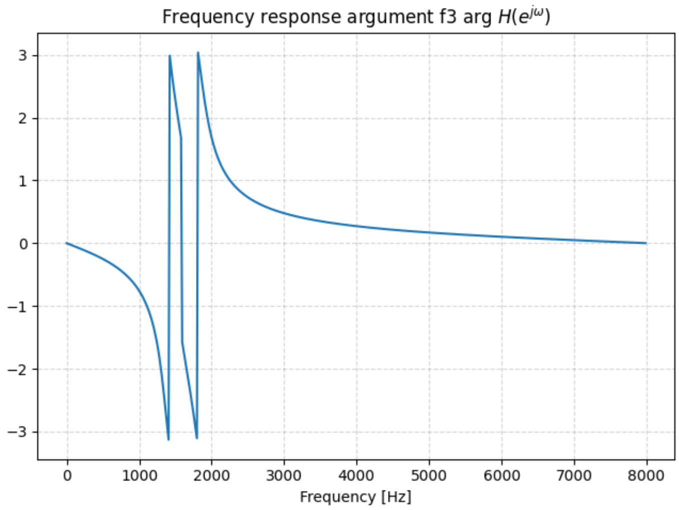
# 9 Frequency response

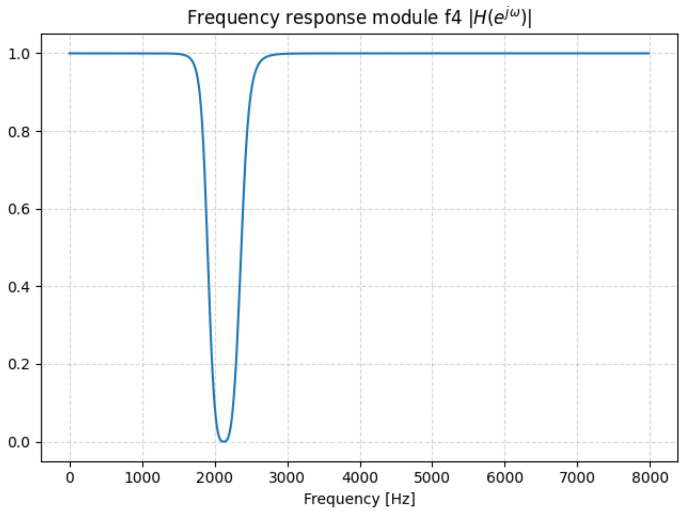


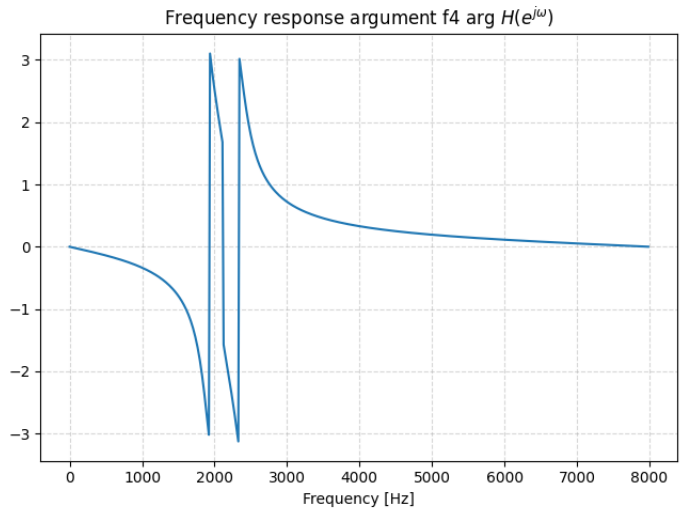

Frequency response module f2 $|H(e^{j\omega})|$
1.0
0.8
0.6
0.4
0.2
0.0
0 1000 2000 3000 4000 5000 6000 7000 8000
Frequency [Hz]


Frequency response argument f2 arg $H(e^{j\omega})$
3
2
1
0
−1
−2
−3
0 1000 2000 3000 4000 5000 6000 7000 8000
Frequency [Hz]


Frequency response module f3 $|H(e^{j\omega})|$
1.0
0.8
0.6
0.4
0.2
0.0
0 1000 2000 3000 4000 5000 6000 7000 8000
Frequency [Hz]

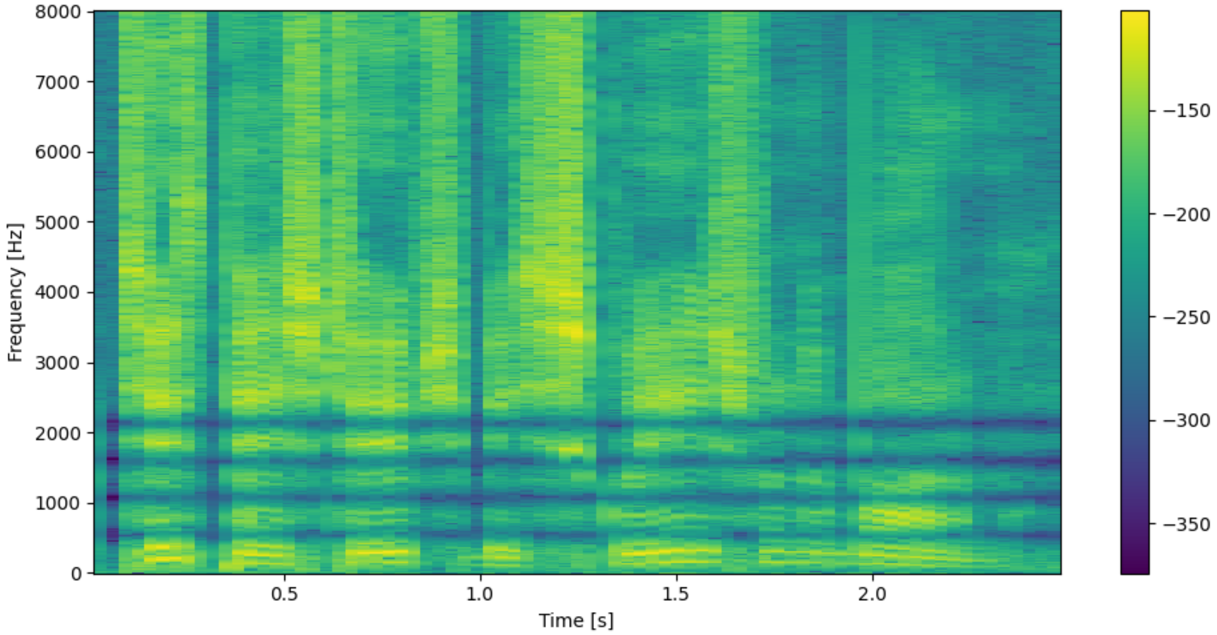

Frequency response argument f3 arg $H(e^{j\omega})$
3
2
1
0
−1
−2
−3
0 1000 2000 3000 4000 5000 6000 7000 8000
Frequency [Hz]


Frequency response module f4 $|H(e^{j\omega})|$
1.0
0.8
0.6
0.4
0.2
0.0
0 1000 2000 3000 4000 5000 6000 7000 8000
Frequency [Hz]


Frequency response argument f4 arg $H(e^{j\omega})$
3
2
1
0
−1
−2
−3
0 1000 2000 3000 4000 5000 6000 7000 8000
Frequency [Hz]

# 10 Filtering

The final signal after filtering is without interfering frequencies but the person sounds like he is talking in some kind of tube. But his speech is understandable very well.




**Spectrogram of filtered signal**

# 11 Conclusion

I managed to use my knowledge gained from ISS course through semester to successfuly completed this project. Finally I learned Python and Jupyter Notebook and I will be able to use this skill in future projects. The hardest part of this project was to create my own DFT function. I have almost lost my hearing, because I used wrong values for my filter and I generated very loud and aggressive noise.