BRNO UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATION TECHNOLOGY

ISS project 2021/2022

Author: Samuel Dobron (xdobro23)

Task 4.1 - load audio

To load input signal I have created function `load_file()` located in `files.py` which returns array of 2 elements. The first one is audio signal stored in 1D array *(type `numpy.ndarray`)*, the second element is sampling rate of audio file detected by function `load()` from `librosa` library, this sampling rate is not used anymore because sampling rate of input audio is already known `- 16 kHz`. If sampling rate of loaded file is not `16 kHz` program raises `NotImplementedError`.

Audio length is determined by following formula:

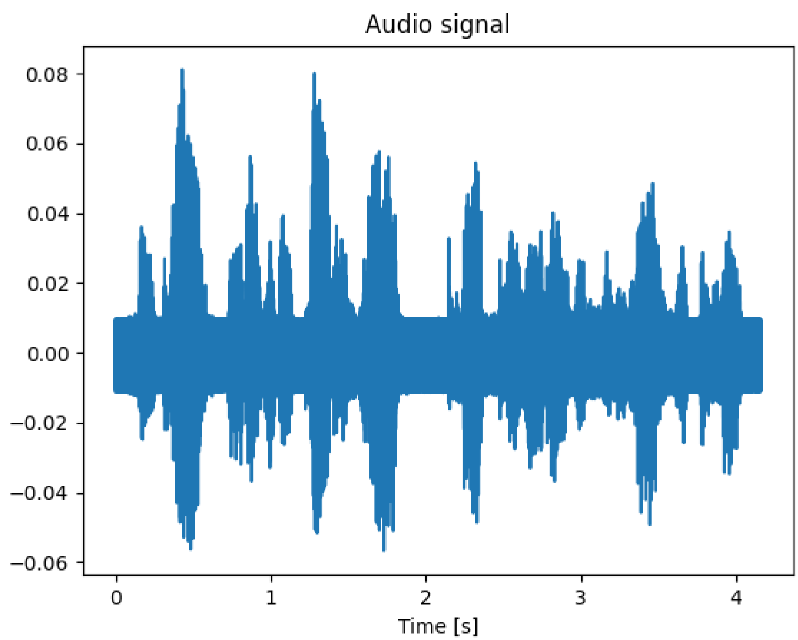
$$length = samples/sampling\_rate$$

Audio length: 4.16 seconds

Audio samples: 66 560

Audio values

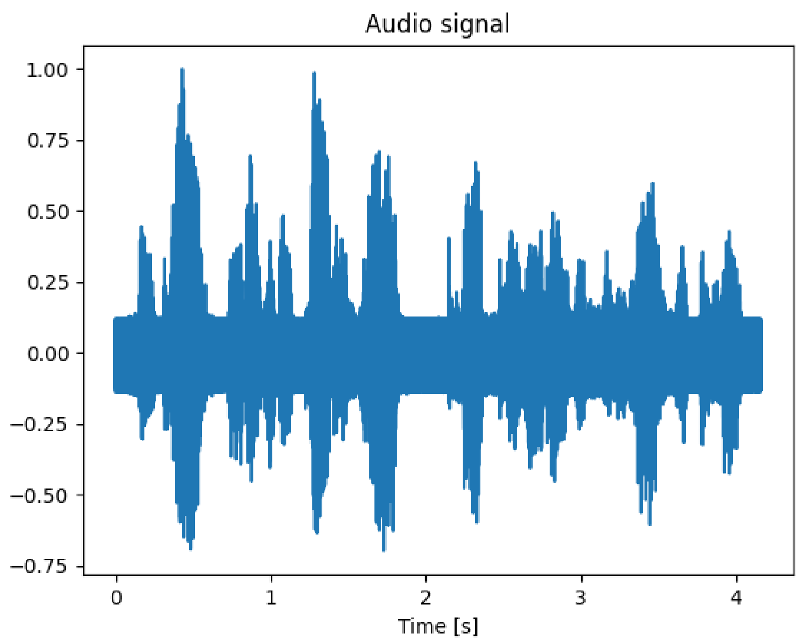
- Minimal: -0.0567
- Maximal: 0.0811



Task 4.2 - normalization

Normalized signal:

Normalization is implemented in function `output.normalize()`.



Signal above is normalized to interval <-1, 1> using following pseudo-code:

```
maximum_value = samples_array.maximum_value()
maximum_value = abs(maximum_value)

for sample in samples_array:
    sample = sample / maximum_value
```

Periodic frame

Signal is divided in to 130 chunks and each contains 1024 samples:

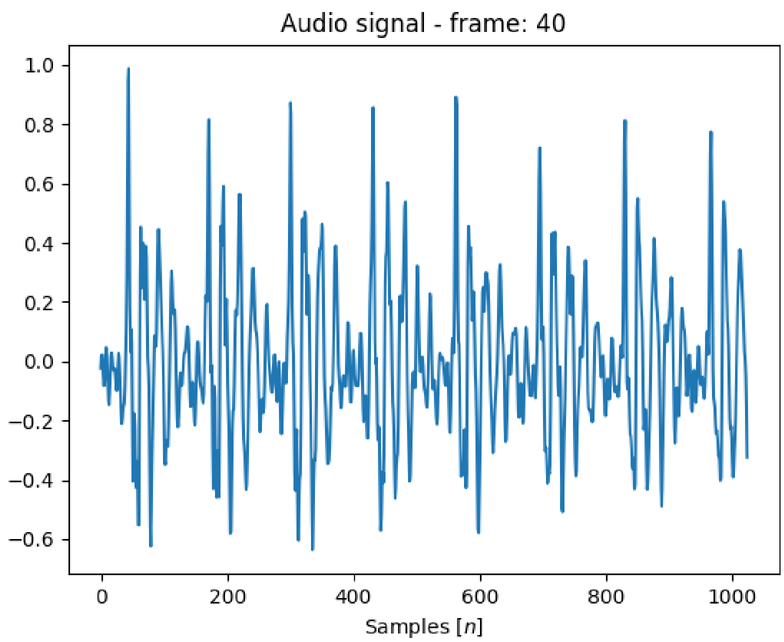
$$chunks\_count = samples\_count // samples\_overlap$$

That results that last 512 and first 512 samples of next chunk are the same. To find periodic frame i chose pretty straight-forward method. I just plotted every frame and have chosen one pretty periodic frame. Using following code:

```
time = np.arange(0, 1024, 1)

for i in range(0, columns - 1):
    plt.title(f"Audio signal - frame: {i}")
    plt.plot(time, normalized[i * SAMPLES_OVERLAP : (i * SAMPLES_OVERLAP) + 1024])
    plt.show()
```

I have chosen frame 40 (actually 41 due to Python indexing)

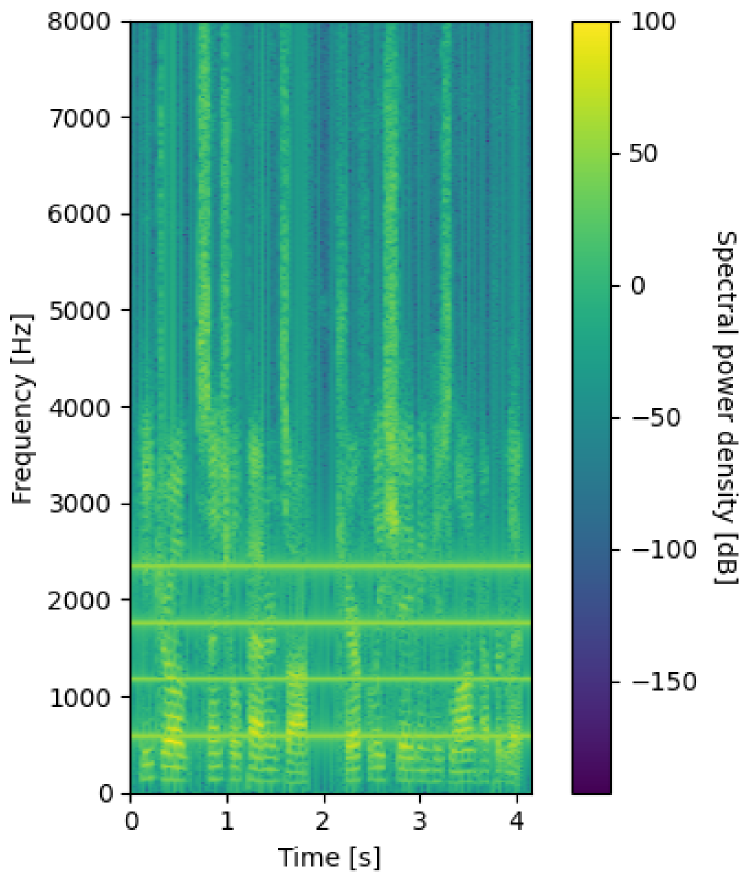


Task 4.3 - DFT implementation

Task 4.4 - audio spectrogram

Firstly, samples are "ordered" to 2D array `filtered_X`, it is just array of arrays, which contains 1024 samples each with 512 samples overlapping. That means that last 512 and first 512 samples of following frame are the same. For calculation is used faster version of DFT - FFT implemented in function `fft()` in `numpy.fft` library.
DFT coefficients are then modified by formula:

$$P[k] = 10log_{10}|X[k]|^2$$



Task 4.5 - disturbing frequencies

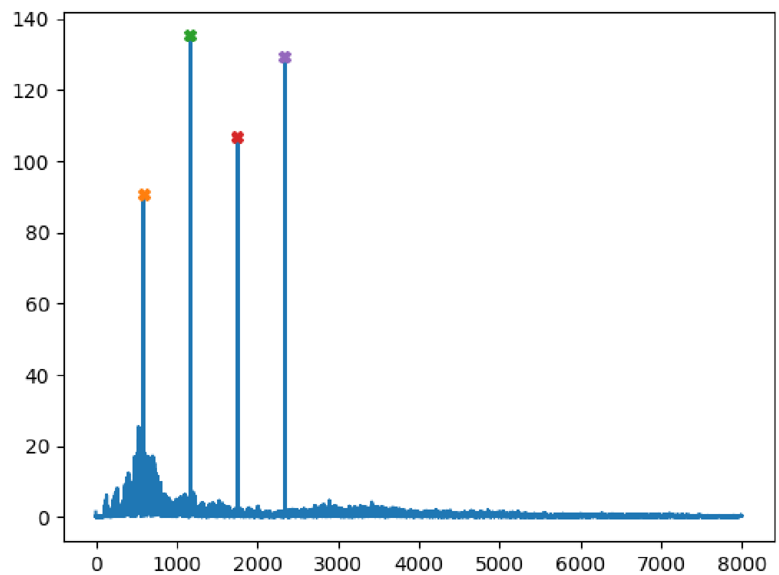
Firstly I tried to read peaks from spectrogram from Task 4.4 but this method is not effective also not very accurate. Because of that I decided to use function `find_peaks()` from `scipy.signal` library, function is used in `get_peaks()` located in `output.py` and it returns list of frequencies with peak. At graph bellow we can see these frequencies, `X` marks peak that found `find_peaks()` function.

$$f_1 = 584.855Hz$$

$$f_2 = 1169.471Hz$$

$$f_3 = 1754.327Hz$$

$$f_4 = 2338.9423Hz$$

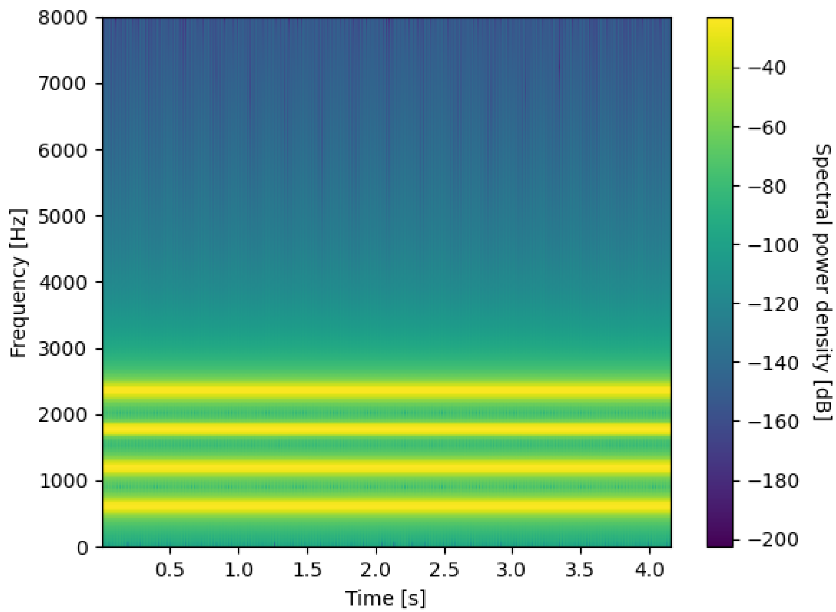


Task 4.6

Generating signal of 4 mixed cosines is implemented as sum of cosine at frequency f1, cosine at frequency f2, ...
For example:

```
DURATION = 4.16
f1_cos = generate_cos(584.855, DURATION)
f2_cos = generate_cos(1169.471, DURATION)
...
mixed_cosines = f1_cos + f2_cos + ...
```

For generating cosines I use function `cos()` from `numpy` library. Spectrogram shows 4 mixed cosines functions at frequencies from Task 4.5.



As we can see, 4 cosines are at the frequencies from Task 4.5.
Generated signal is saved in `audio/4cos.wav` file.

Task 4.7.3 - band-stop filter

My filter consists of set 4 band-stop filters. Every filter filters single frequency from Task 4.5. I am using function `iirfilter` with parameters `btype="bandstop"` and `output="ba"` from library `scipy.signal` to create each filter.
Filtered band is always in interval:

$$< F_i - 50, F_i + 50 >; i»(1, 2, 3, 4)$$

For example for band-stop of 584.855 Hz frequency:

$$< 534.855, 634.855 >$$

I do not think, it is necessary to filter additional `30 Hz` around frequencies, `50 Hz` is enough, disturbing tone is gone so expanding the range could result to filter "valid" frequencies.

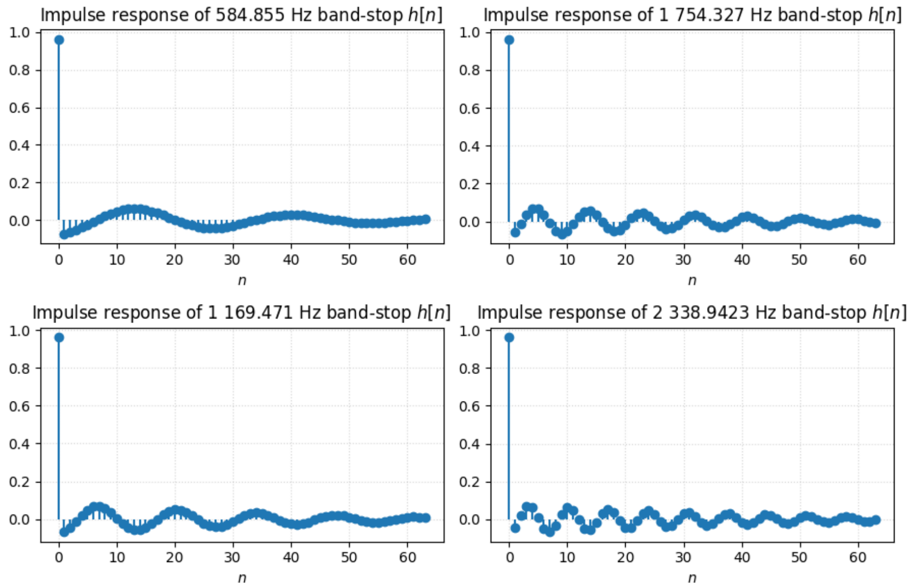
Coefficients

Coefficients bellow are rounded by Python's built in function `round()` to 5 decimal numbers. To generate table without rounding just disable it in `constants.py` - `ROUND_COEFFICIENTS`.

Band-stop frequency	b coefficients	a coefficients
584.855 Hz ± 50 Hz	[0.96149, -5.61853, 13.82856, -18.34291, 13.82856, -5.61853, 0.96149]	[1.0, -5.76709, 14.00884, -18.34002, 13.6468, -5.47286, 0.92446]
1 169.471 Hz ± 50 Hz	[0.96149, -5.17218, 12.15879, -15.88768, 12.15879, -5.17218, 0.96149]	[1.0, -5.30894, 12.3172, -15.88502, 11.99889, -5.03809, 0.92446]
1 754.327 Hz ± 50 Hz	[0.96149, -4.45408, 9.76228, -12.4483, 9.76228, -4.45408, 0.96149]	[1.0, -4.57185, 9.88932, -12.44601, 9.63375, -4.3386, 0.92446]
2 338.9423 Hz ± 50 Hz	[0.96149, -3.50249, 7.1374, -8.72636, 7.1374, -3.50249, 0.96149]	[1.0, -3.5951, 7.23007, -8.72456, 7.04324, -3.41168, 0.92446]

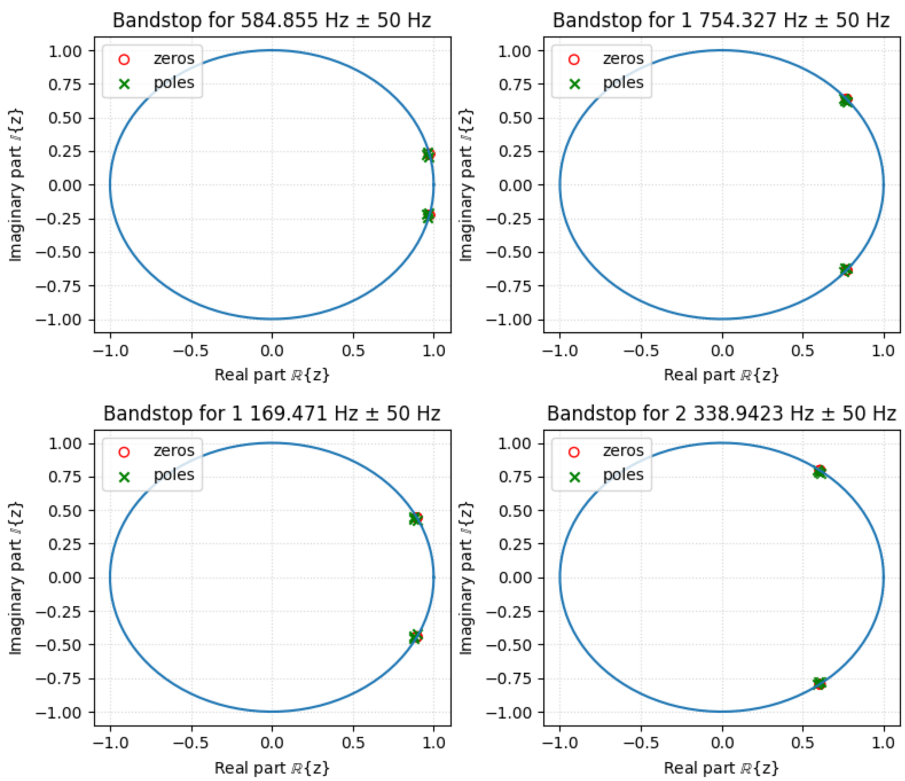
Impulse responses

Coefficients used for calculating impulse responses are not rounded.

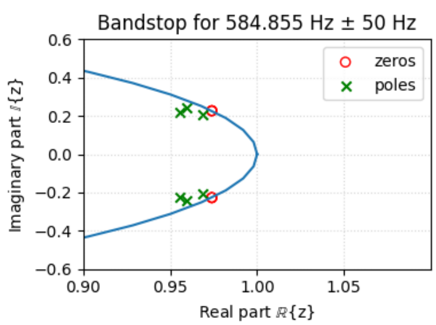


Task 4.8 - zeros & poles

Zeros and poles are calculated using function `tf2zpk()` from `scipy.signal` library.

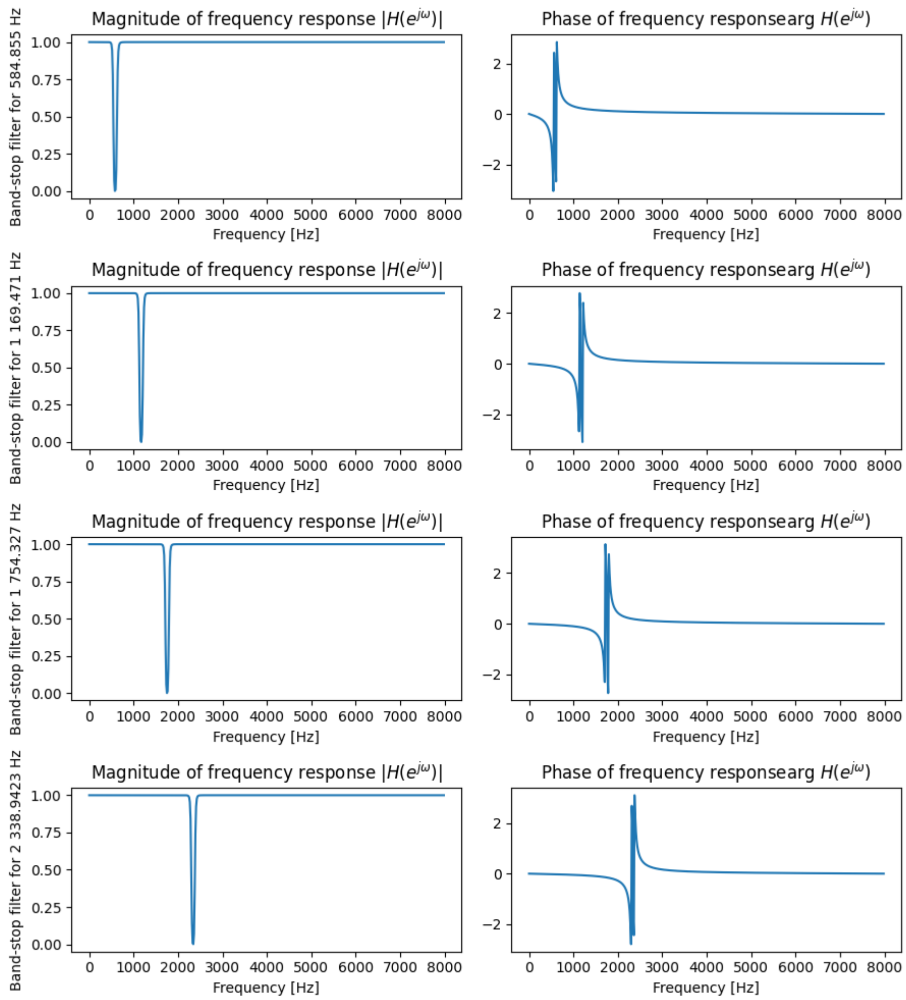


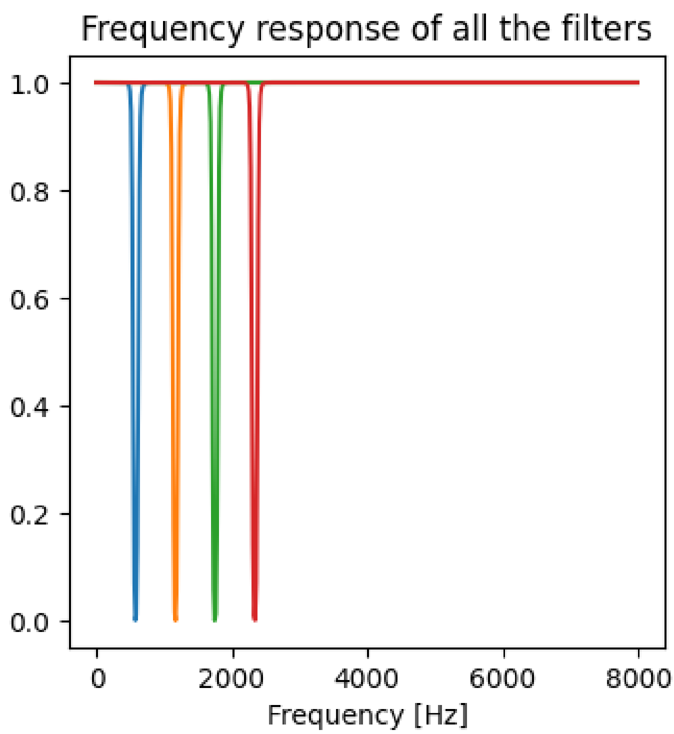
As we can see, zeros and poles are not much visible, here is zoomed plot of band-stop 584.855 Hz:



Task 4.9 - frequency responses

To calculate frequency response is used `freqz()` function from `scipy.signal` library.





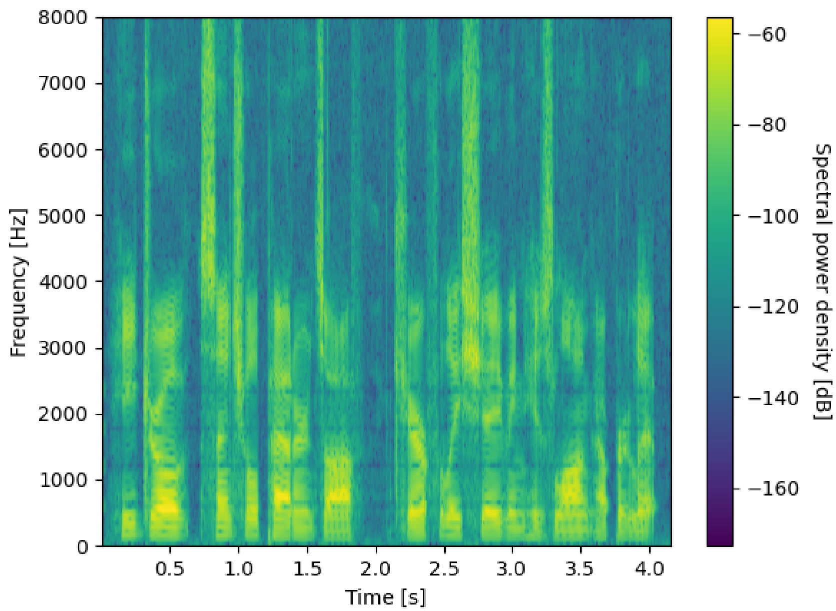
As we can see at the chart, filter drops to 0.0 at the frequencies

$$f_i; i»1, 2, 3, 4$$

Task 4.10 - filtration

Filters are applied using, `lfilter()` function from `scipy.signal` library, one by one in increasing frequency order using following pseudo-code:

```
output = input
for freq in freqs:
    output = filter(output, freq)
```



At spectrogram bellow we can see, that noisy frequencies has been removed. Frequencies are basically "silent" but before filters application they contain only `cos` noise.
Audio without noise is saved in `audio/clean_bandstop.wav` file.