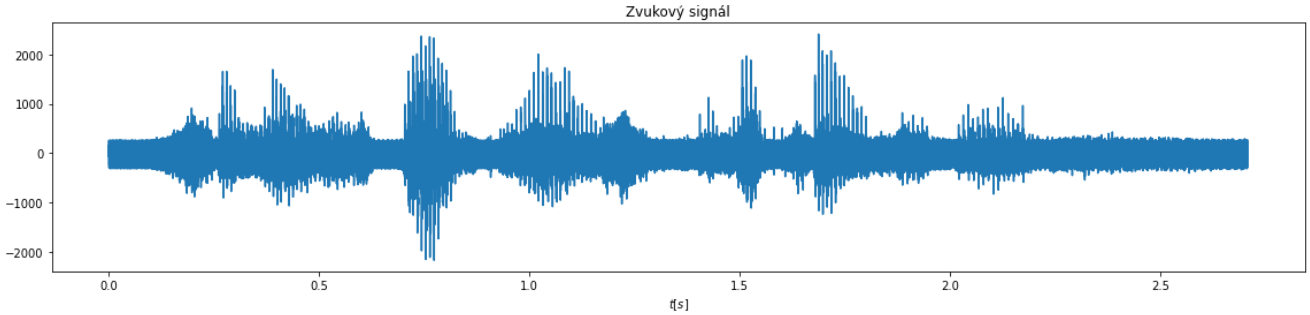
Základy

Pracujeme s nahrávkou *xbartu11.wav*, kterou pomocí knihovní funkce *wavfile* načteme. Tato funkce vrací vzorkovací frekvenci a pole hodnot. Počet prvků v poli nám udává počet vzorků. Pokud počet vzorků podělíme vzorkovací frekvencí, tak dostaneme délku nahrávky v sekundách. Pomocí funkce max() a min () zjistíme nejmenší hodnoty v poli respektive vzorků.

Získané výsledky:
Délka ve vzorcích je: 43316
Délka v sekundách je: 2.70725[s]
Nejmenší hodnota je: -2173
Největší hodnota je: 2413

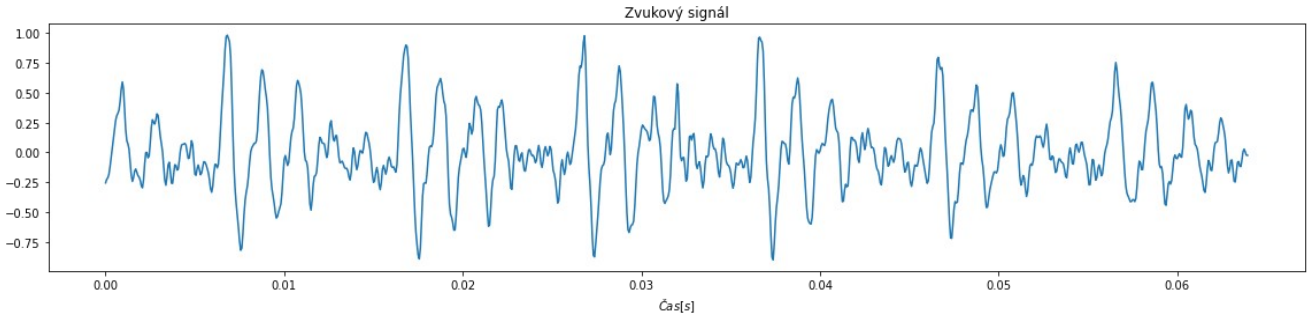


Předzpracování a rámce

Střední hodnotu z dat jsme vypočítali pomocí vzorce $\frac{1}{n}\sum_{i=1}^{n} x_i$, kterou jsme následně odečetli od dat. Normalizaci jsme provedli pomocí následujícího příkazu

```
data = data / max(abs(min_val), max_val)
```

data jsme následně s překrytím 512 vzorků uložily jako řádkový vektor, z kterého jsme následně pomocí funkce *resize* udělali matici o 1024 sloupcích. Díky tomuto jsme mohli lehce prohlížet jednotlivé rámce a mohli najít pěkný znělý rámec.



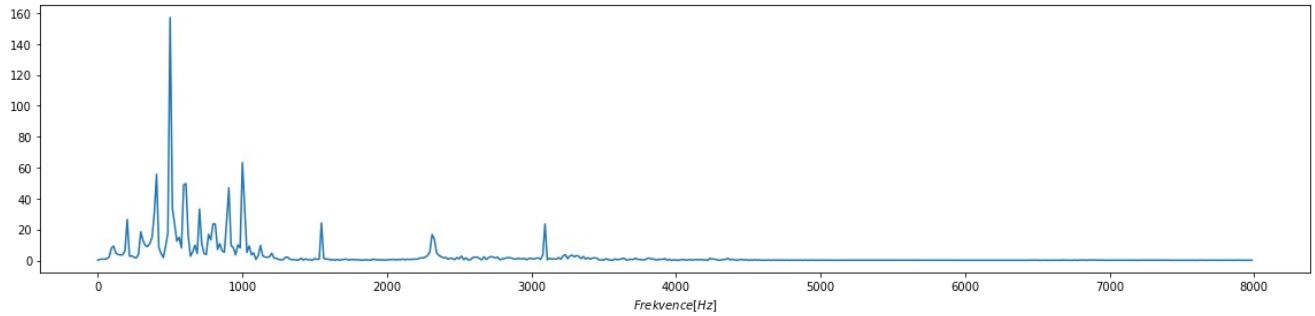
DFT

Při implementaci vlastní diskrétní Fourierovy transformace vychazíme ze vzorce $X´[k] = X[k] * e^{-j\frac{2\pi}{N}kn}$ kde $X[k]$ je původní signál a $e^{-j\frac{2\pi}{N}kn}$ je hodnota, kterou budeme signál násobit. Respektive, protože pracujeme vektorově tak budeme

maticově násobit.

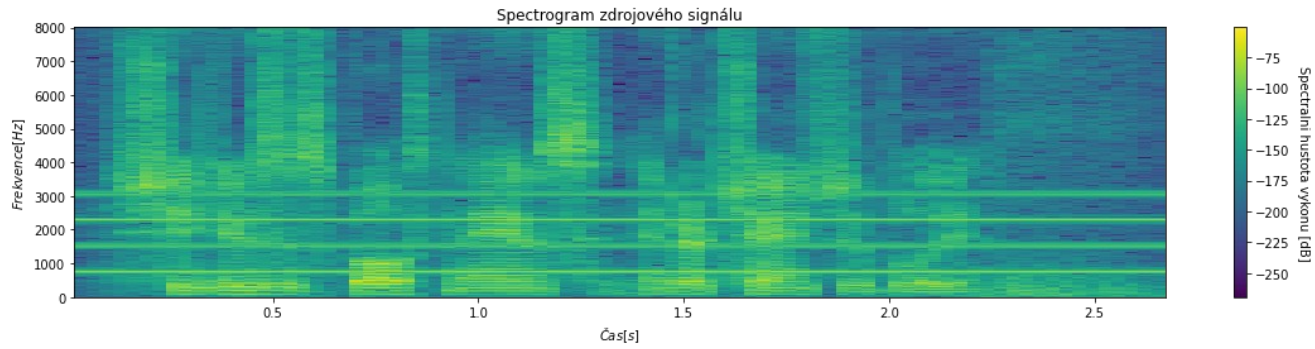
Vstupem vlastní funkce DFT je vektor hodnot. Z něho si pomoci funkce *len( )* zjitíme parametr *N*, následně vytvoříme řádkový vektor *n* a sloupcový vektor *k*, které nabývají hodnot od 0 po *N-1*. Díky tomu nám po vynásobení vznikne čtvercová matice $N * N$. Následně pak maticově vynásobíme vektor na vstupu funkce s námi vypočítanou maticí pomocí binárního operátoru @ Výsledkem bude řádkový vektor, který následně vrátíme.

DFT chceme zobrazit od 0 do $\frac{F_s}{2}$ tudíž z vektoru, který jsme dostali z *DFT( )* potřebujeme jen dolní polovinu dat, tudíž pomoci *range* odstraníme horní polovinu dat. Následně vytvoříme vektor frekvencí na kterých jsou jednotlivá data a nakonec zobrazíme graf.



Spektrogram

Pro zobrazení spektrogramu jsme využili funci *spectrogram()*. Tato funkce vnitřně implementuje *fft*. Pomocí parametru *nperseg=1024* jsme nastavili délku segmentu a pomocí *noverlap=512* překrytí segmentu. Následně jsme vzorky signálu upravili logaritmickou funkcí $P[k] = 10 * \log_{10}|X[k]|^2$



Určení rušivých frekvencí

Ručně pomocí zmenšování rozsahu na ose y jsme odhadly rušivé frekvence a to následující:

$f_4 = 3094$

$f_3 = 2320.5$

$f_2 = 1547$

$f_1 = 773.5$

Nyní oveříme zda jsou frekvence f_4, f_3, f_2 násobky té nejnižší respektive f_1

$f_4 : \frac{f_4}{f_1} = \frac{3094}{773.5} = 4$

$f_3 : \frac{f_3}{f_1} = \frac{2320.5}{773.5} = 3$

$f_2 : \frac{f_2}{f_1} = \frac{1547}{773.5} = 2$

Generování signálu

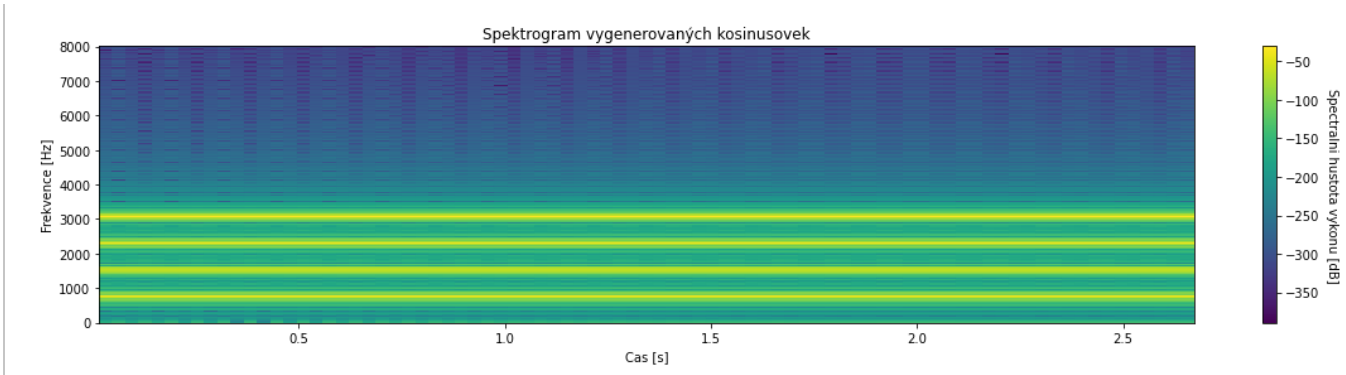
Pomocí již zjištěných frekvení f_4, f_3, f_2, f_1 vytvoříme 4 signály o těchto frekvencích, které sečteme dohromady. Signály jsou vytvořeny z následujícího vzorce: $y[k] = cos(2\pi ft)$ Zobrazíme spektrogram pomocí funkce zmíněné výše. Předtím než uložíme signál tak ho znormalizujeme. Zjistíme normu a následně signál normou podělíme

```
norm = np.linalg.norm(cos)
```

```
norm_cos = cos / norm
```

Následně pomocí funkce *wavfile.write()* uložíme signál a tento signál musí mít vzorkovací frekvenci 16kHz, tudíž můžeme použít naší zjištěno a bitová šířka by měla být 16bit a to uděláme následovně.

```
wavfile.write("audio/4cos.wav", samplingFrequency, (norm_cos *
np.iinfo(np.int16).max).astype(np.int16))
```



Čistící filtr

Čistící filtr je implementován jak 4 pásmové zádrže(bandstop). Pro zjednodušení byla vytvořena funkce *BandStopKoef()*, která má dva paremetry a to frekvenci, kterou chceme eliminovat a vzorkovací frekvenci a funkce nám vrácí koeficinety b, a. Dále byla vytvořena funkce *Filter()*, která má tři parametry a to koeficienty b, a společně s nějakým signálem. V tomto případě se jedná o impulzní odezvu.

BandStopKoef()

V této funkci využíváme funkce *buttord a butter* z knihovny *scipy*, pro správný chod funkce *buttord* bylo třeba vypočítat Nyquistovu frekvenci $\frac{F_s}{2}$ a touto hodnotou bylo třeba podělit data použitá v parametrech viz. následující kód.

```
N, Wn = buttord(wp=[(f-15)/nygs, (f+15)/nygs], ws=[(f-15-25)/nygs, (f+15+25)/nygs], gpass=0.5,
gstop=5)
```

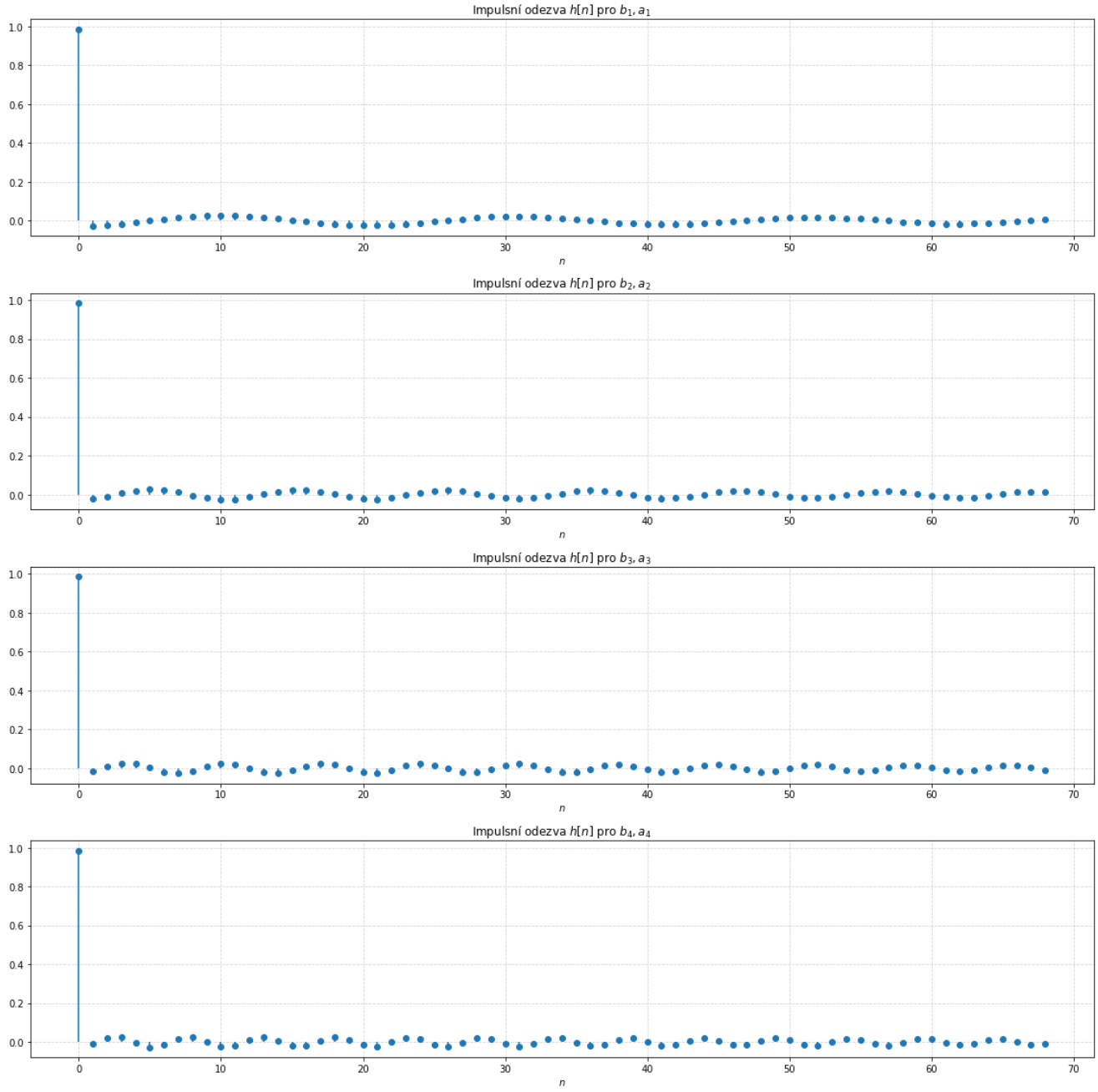
Jednotlivé parametry byly zvoleny tak, aby co možno nejlépe vyčistili signál. Následující funkce *butter* nám pomocí hodnot, které nám vrátila předešlá funkce, vypočíta jednotlivé koeficienty b, a, které následně vrátíme.

Filter()

Tato funkce aplikuje koeficinenty filteru na signál a to díky funkci *lfilter()* z knihovny *scipy* a vrací nám vyfiltrovaný signál, který poslezé také vrátíme.

Zobrazení impulzní odezvy

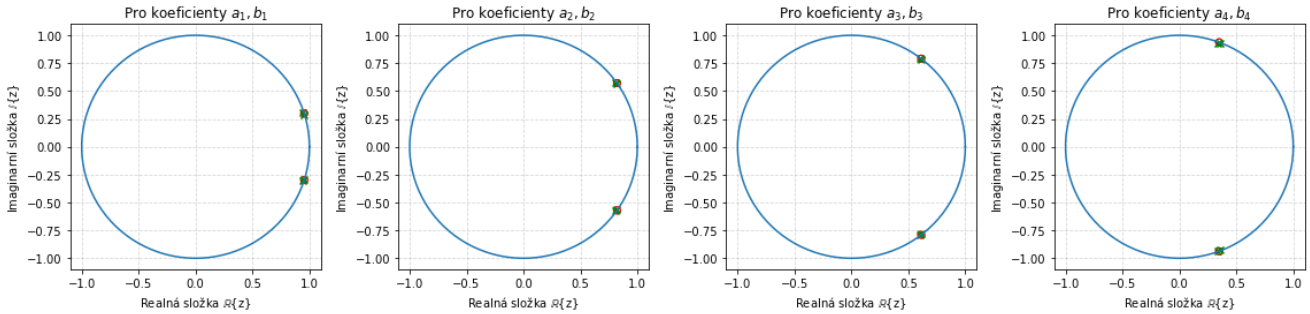
Vytoříme si jednotkový impluz, který se pak zobrazí po aplikovaní filtru, respektive po aplikaci funkcí *BandStopKoef()* a *Filter()*. Toto jsme provedli celkem 4 krát a to pro frekvence f_1, f_2, f_3, f_4.



Nulové body a póly

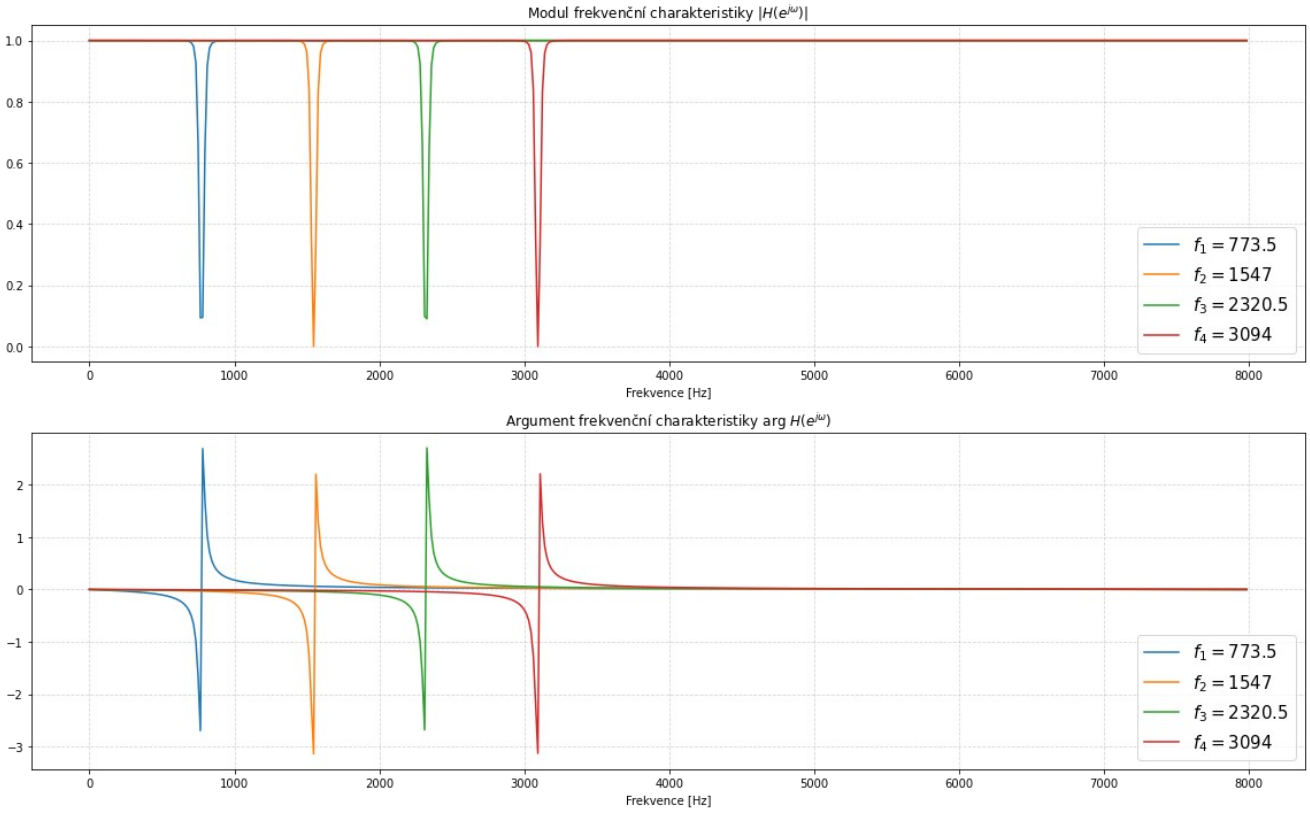
Protože budeme chtít vypočítat nuly a póly celkem 4 krát tak jsme vytvořili funkci *PolesAndZeros()*, v které se pomocí funkce *tf2zpk* z knihovny vypočítají nuly a póly z koeficientů b, a. Funkce vrací pole nul, pólů a přírustek systému.

Nuly a póly jsme si zobrazily a zjistili jsme, že se zobrazili, pro jednotlive koeficienty b_i, a_i kde i = {1, 2, 3, 4}, sami do sebe.



Frekvenční charakteristika

Moduly frekvenční charakteristiky frekvencí f_1, f_2, f_3, f_4 jsme zobrazily do jednoho grafu a z něho lze vidět, že by měl filtr skutečně potlačovat frekvence na určených frekvencí a argumenty frekvenční charakteristiky daných frekvencí, tak nám určily, jak se jednotlivé frekvenční složky zpozdí/předběhnou.



Filtrace

Postupně jsme vyfiltrovali signál pomocí námi již předem zjistěných koeficinetů b, a.

```
sf = lfilter(b1, a1, data)
```

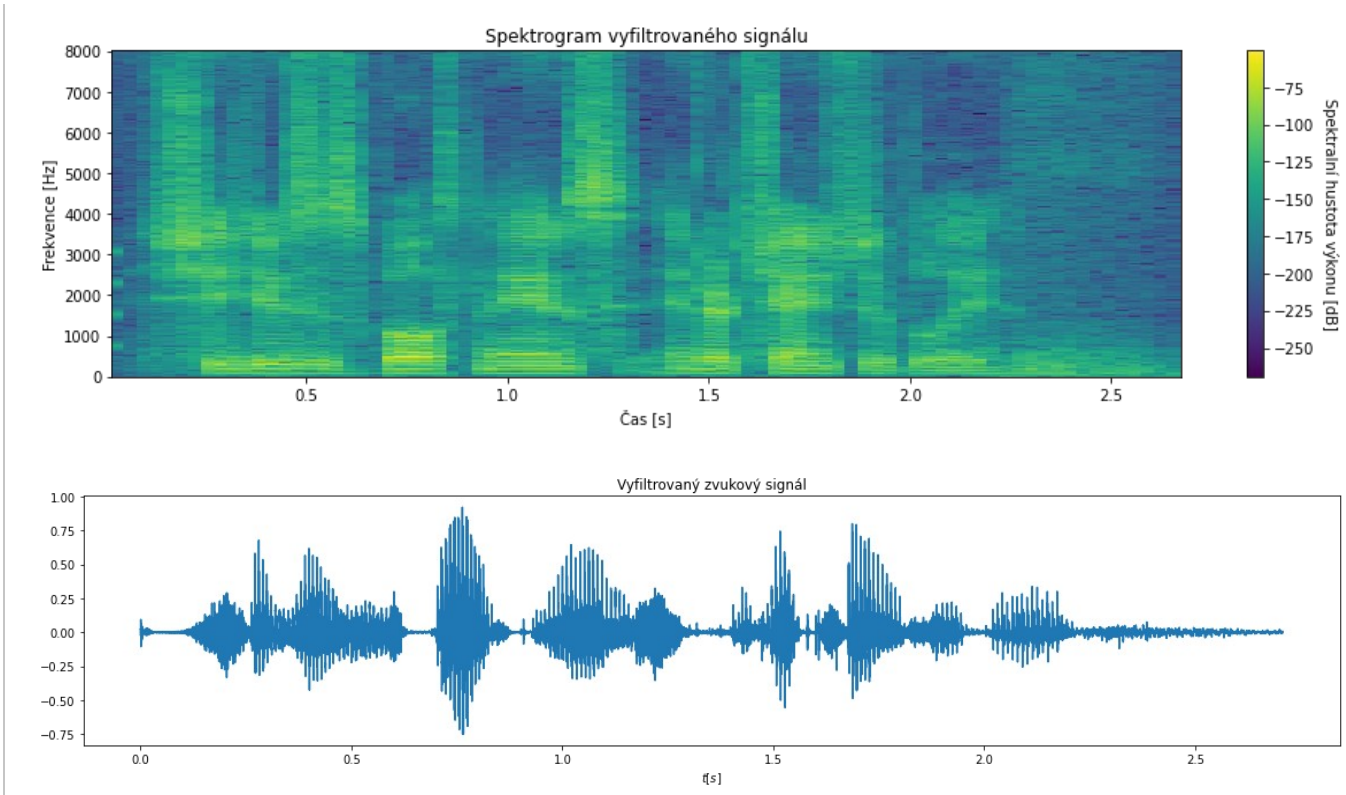
```
sf = lfilter(b2, a2, sf)
```

```
sf = lfilter(b3, a3, sf)
```

```
sf = lfilter(b4, a4, sf)
```

Pro kontrolu zda byly skutečně rušivé frekvence odstraněny, tak jsme zobrazili spectrogram vyfiltrovaného signálu a také jeho časový průběh. Je zde vidět, že se nám frekvence podařilo vyfiltrovat kromě počátku audia. Toto "pípnutí" se mi nepodařilo zcela vyfiltrovat ikdyž jsme měnil vlastnosti filtru. Nic méně filtr vyfiltroval rušivé frekvence v podstatě v celém audiu.

Následně vyfiltrovaný signál jsme uložili do souboru stejným způsobem, který byl zmíněn výše.



Zdroje

https://nbviewer.org/github/zmolikova/ISS_project_study_phase/blob/master/Zvuk_spektra_filtrace.ipynb

https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.spectrogram.html

https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.wavfile.write.html

https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.buttord.html