VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Signály a systémy - projekt

Čistenie signálu pomocou filtru

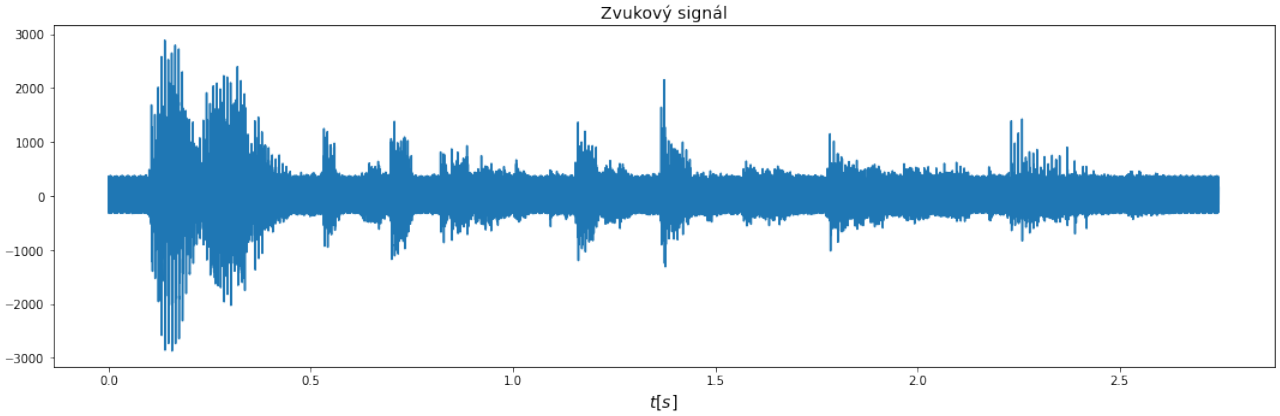
15. decembra 2021

Denis Kramár

1. úloha - základy

Vstupný signál som načítal pomocou knižnice *scipy* ako reťazec. Zistil som dĺžku daného reťazca, ktorá predstavuje dĺžku vo vzorkách a pomocou pomeru *dĺžka vo vzorkách / 16000* som získal dĺžku nahrávky v sekundách, kde číslo 16000 predstavuje vzorkovaciu frekvenciu. Pomocou funkcií *min* a *max* knižnice *numpy* som zistil minimálnu a maximálnu hodnotu.[2]

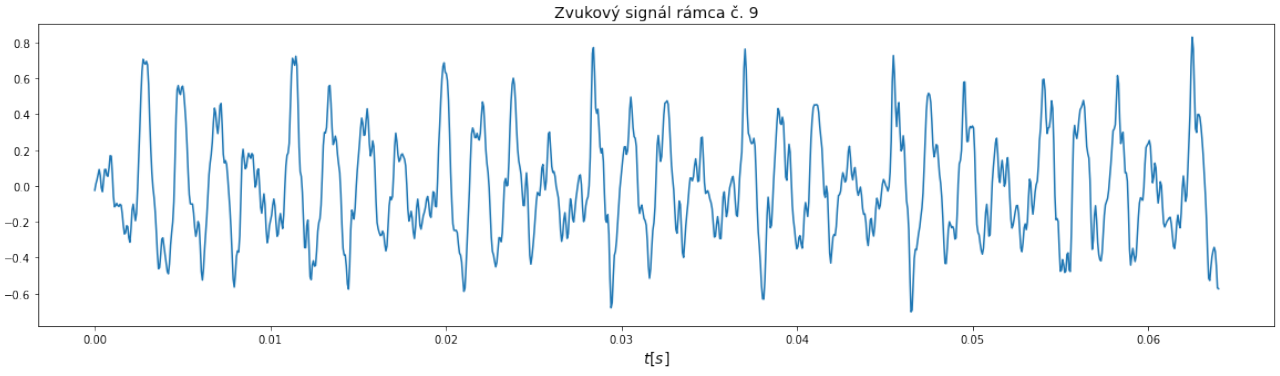
Dĺžka vo vzorkách	43930
Dĺžka v sekundách	2.745625
Minimálna hodnota	-2871
Maximálna hodnota	2884



2. úloha - predspracovanie a rámce

Zo vstupného signálu som získal priemernú hodnotu a maximum z absolútnych hodnôt a následne som pomocou týchto hodnôt signál ustrednil a normalizoval. Na zistenie počtu rámcov, ktoré s dĺžkou 1024 vzoriek a prekrytím 512 vzoriek môžem na danom signále vytvoriť som použil následovný vzorec [2]

$$n = \left\lfloor \frac{Fs}{1024} \right\rfloor * 2 = \left\lfloor \frac{43930}{1024} \right\rfloor * 2 = 42 * 2 = 84$$

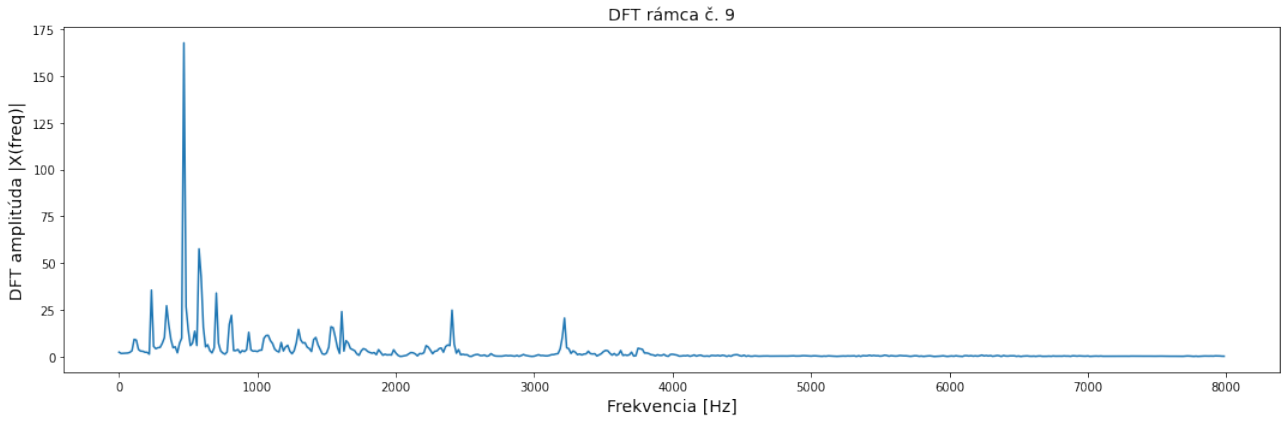


3. úloha - DFT

Na výpočet diskrétnej fourierovej transformácie som si vytvoril funkciu *dft(signal)* a samotný výpočet je vykonaný následujúcim vzorcom a kódom:[1][2]

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i 2\pi k n / N}$$

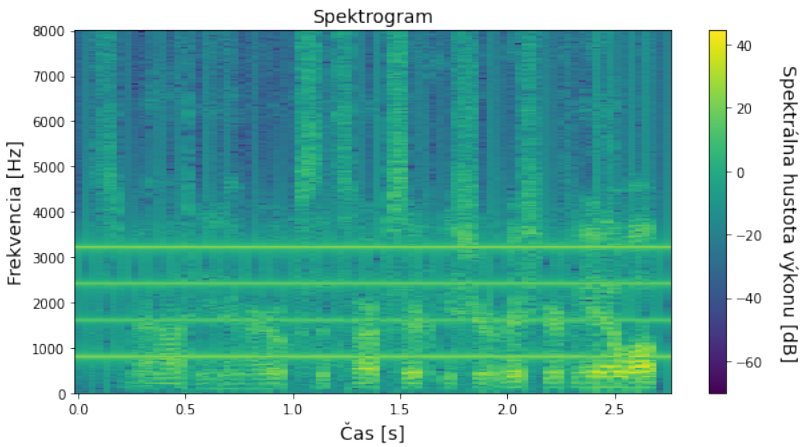
```
def dft(signal):
    N = len(signal)
    n = np.arange(N)
    k = np.arange(N)[:, None]
    M = np.exp((-2j * np.pi * k * n) / N)

dft = np.dot(M, signal)
return dft
```



4. úloha - spektrogram

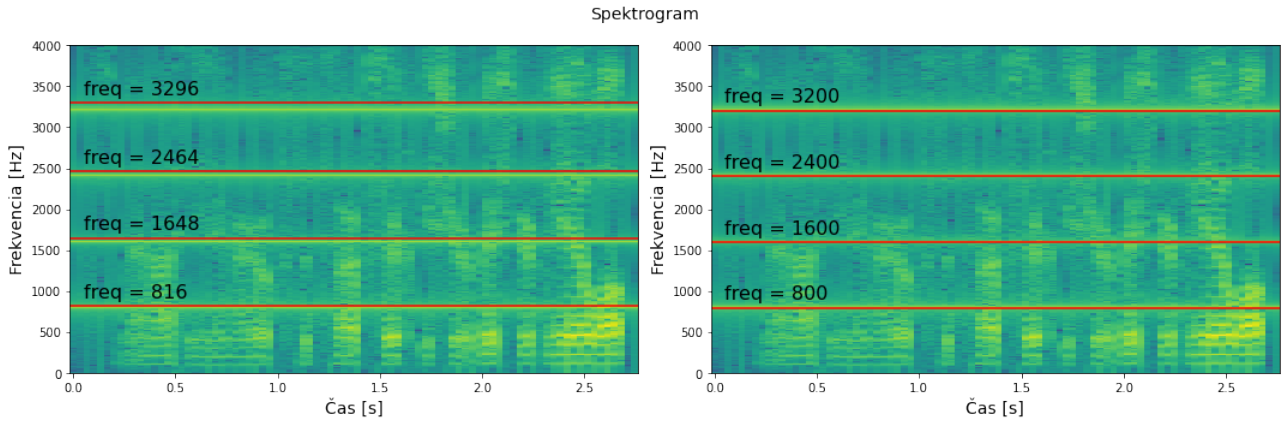
Pre vytvorenie spektrogramu som musel najprv musel aplikovať diskrétnu fourierovu transformáciu na všetky rámce a následne získať logaritmickú spektrálnu hustotu výkonu, čo som urobil pomocou následujúceho vzorca[2][3]

$$G[k] = 10 * \log_{10} |X[k]|^2$$



5. úloha - určenie rušivých frekvencií

Určiť rušivé frekvencie som sa snažil najprv pomocou funkcie, avšak to nebolo úplne presné. Nakoniec som sa teda rozhodol pre ”ručné” odčítanie rušivých hodnôt z grafu. Na obrázku nižšie môžete vidieť určenie rušivých hodnôt pomocou funkcie na ľavej strane a na pravej strane určenie odčítaním z grafu. Hodnota najnižšej frekvencie určená pomocou funckie nie je úplne presná a vyššie frekvencie nie sú jej násobkom. Ja som za hodnotu najnižšej frekvencie zvolil hodnotu 800Hz, ktorá mi prišla za najvhodnejšiu, a násobky frekvencie 800Hz. (Pre lepšiu vizualizáciu na menšom priestore som obmedzil rozsah osy y na 4kHz.)

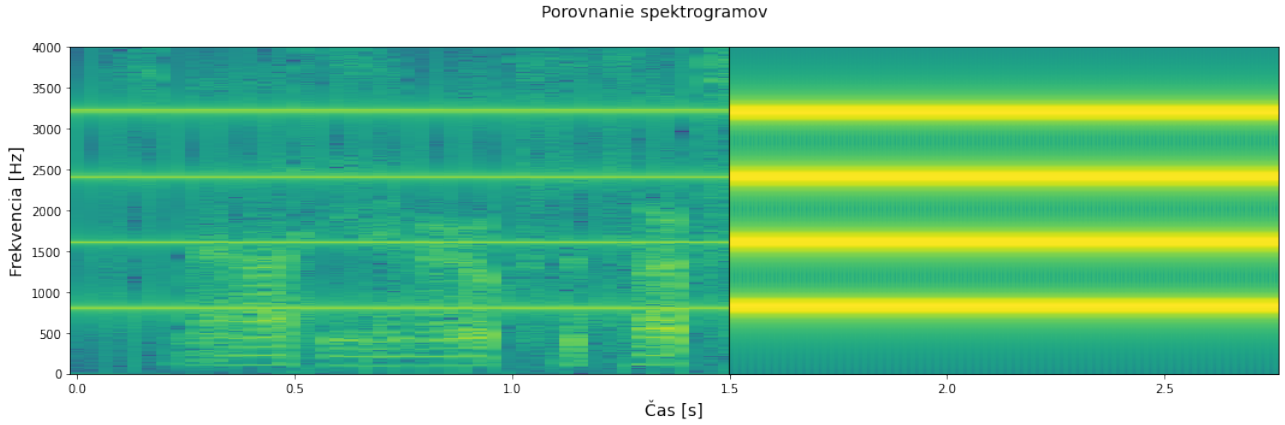


6. úloha - generovanie signálu

Signál som vygeneroval ako súčet kosínusoviek. Argument *800n* reprezentuje frekvenciu v závislosti na n, tak aby sa vygenerovali kosínusovky s frekvenciami *800, 1600, 2400* a *3200*, a argument *t* reprezentuje pozíciu v rovnomerne rozloženom čase s dĺžkou 2.745625s, tak ako pôvodna nahrávka, s počtom vzoriek 43930, rovnako ako pôvodna nahrávka. [2]

$$\sum_{n=1}^{4} \cos(800n2\pi t)$$

Vygenerovaný signál som porovnal podľa vypočutia a porovnaním spektrogramu pôvodnej nahrávky a vygenerovaného signálu, a overil tak zhodu s rušivým signálom v pôvodnej nahrávke. (Pôvodný signál vľavo, vygenerovaný signál vpravo.)

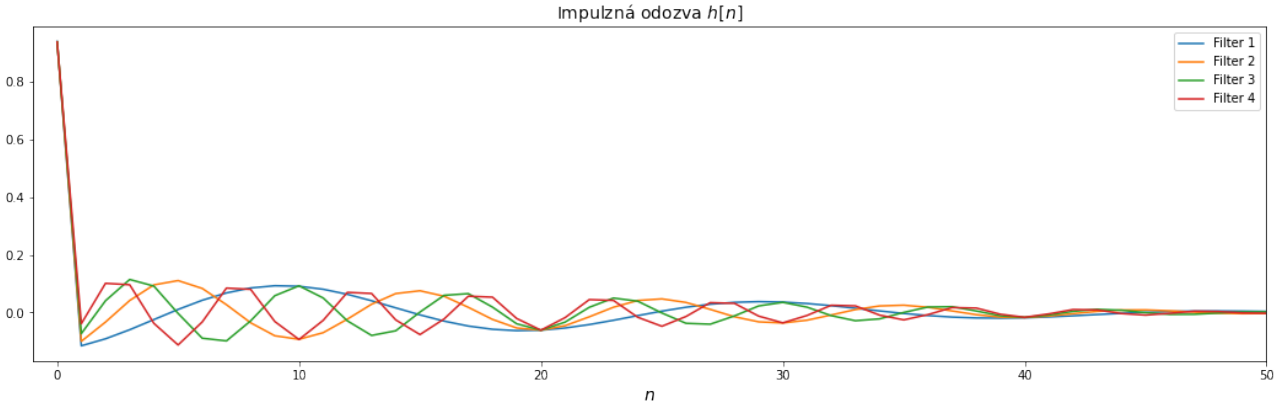


7. úloha - čistiaci filter

Filtrovať som sa rozhodol pomocou 4 pásmových zádrží. Na návrh a generovanie čistiaceho filtru som si vytvoril funkciu *makeButter(block_fq)*, kde block_fq je frekvencia, ktorú chceme blokovať. Vrámci funkcie som filter nastavil tak, aby dĺžka záverného pásma bola 30Hz, teda 15Hz doľava aj doprava od *block_fq*, dĺžku prechodu z a do prepustného pásma som nastavil na 50Hz, zvlnenie v prepustnom pásme na 3dB a potlačnenie v závernom pásme na -40dB. Návrh a generovanie filtra je vykonávané pomocou funkcií *scipy.signal.buttord()* a *scipy.signal.butter()*. Na získanie koeficientov jednotlivých filtrov som použil funkciu *scipy.signal.sos2tf()*. Filtre som aplikoval na dirakov impulz a zostavil graf. [2]

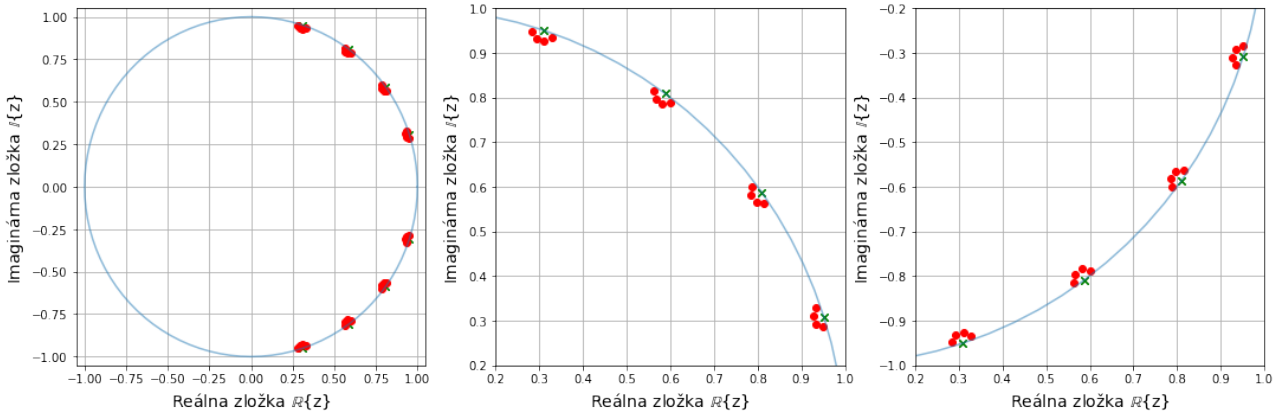
```
def makeButter(block_fq):
    nyq = sampleRate * 0.5
    passband = [(block_fq - 15 - 50) / nyq, (block_fq + 15 + 50) / nyq]
    stopband = [(block_fq - 15) / nyq, (block_fq + 15) / nyq]
    N, Wn = signal.buttord(passband, stopband, 3, 40)
    return signal.butter(N, Wn, 'bandstop', output='sos')
```

Koeficienty b filtru 1	0.9376659672932826, −7.134324696969703, 24.106495792741207, −47.21620244821185, 58.61281672986276, −47.21620244821187, 24.10649579274122, −7.134324696969707, 0.9376659672932834
Koeficienty a filtru 1	1.0, -7.486194651568064, 24.88927933395864, -47.968394019418064, 58.594877927991284, -46.449231776622746, 23.33776552176166, −6.797233842754276, 0.8792174662201675
Koeficienty b filtru 2	0.9365192092192571, -6.061406027422333, 18.457725034101873, -34.05384684128202, 41.46194517787623, -34.05384684128202, 18.45772503410187, -6.061406027422333, 0.9365192092192571
Koeficienty a filtru 2	1.0, -6.366167983983941, 19.068250334224395, -34.60554570722893, 41.447369666071125, -33.48910936781583, 17.857745434986157, -5.769682678379991, 0.8770682292368006
Koeficienty b filtru 3	0.9360530889741969, -4.401694351580104, 11.506157188253248, -19.288375610169574, 22.928087192525368, -19.28837561016957, 11.506157188253242, -4.401694351580101, 0.936053088974196
Koeficienty a filtru 3	1.0, -4.624724658045192, 11.889116448215253, -19.602116661116902, 22.918350682583522, -18.965021819327966, 11.128845230803215, -4.188276785009309, 0.8761953853782823
Koeficienty b filtru 4	0.9357526805084607, -2.3133717205689512, 5.887683466099695, -7.823793327840285, 10.040401126598603, -7.823793327840285, 5.887683466099695, -2.3133717205689512, 0.9357526805084607
Koeficienty a filtru 4	1.0, -2.4311704253883994, 6.083890342757531, -7.951062754814336, 10.034700094474697, -7.691422584894459, 5.693049903503759, -2.2006743317212756, 0.8756330790789217



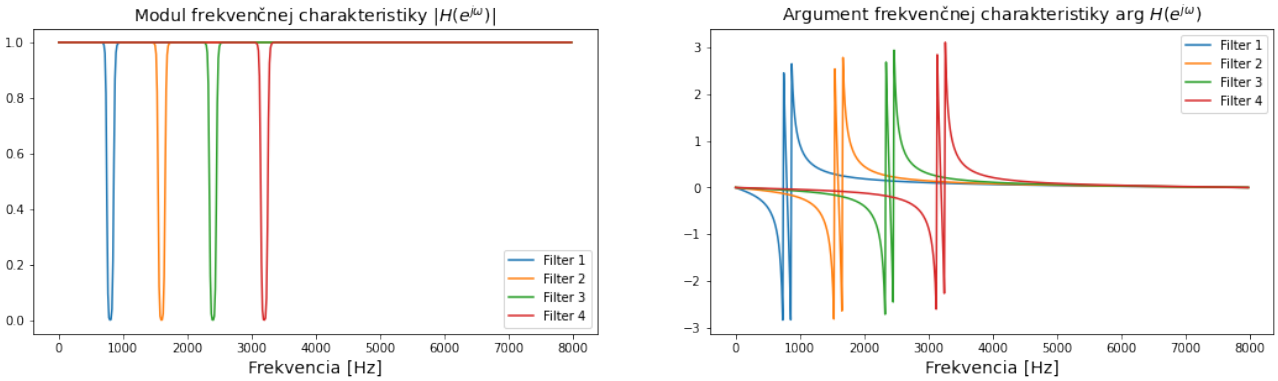
8. úloha - nulové body a póly

Nulové body a póly som získal pomocou funkcie *scipy.signal.sos2zpk()* a následne ich vykreslil na graf. V grafe naľavo môžete vidieť celý rozsah, graf v strede je priblíženie na prvý kvadrant a graf vpravo je priblíženie na štvrtý kvadrant. Zelené krížiky reprezentujú nuly a červené bodky reprezentujú póly. [2][3]



9. úloha - Frekvenčná charakteristika

Pomocou funkcie *signal.sosfreqz()* knižnice *scipy* som vyzobrazil správanie pre jednotlivé frekvencie. Z grafu maginutúdy frekvenčnej charakteristiky môžeme zistiť, ako filter potlačí alebo posíli rôzne frekvenčné zložky v signáli. Pomocou grafu argumentu frekvenčnej charakteristiky môžeme zasa vidieť ako sa jednotlivé frekvečné zložky opozdia alebo predbehnú. [2][3]



10. úloha - Filtrácia

V poslednej úlohe som už iba aplikoval vytvorený filter na vstupný signál *xkrama06.wav* pomocou funkcie *signal.sosfilt()* z knižnice *scipy*. Overil som či je filter stabilný, a kedže je, uložil som ho ako *clean_bandstop.wav*. [2]

Záver

Vrámci projektu som získal mnohé informácie o tom ako pracovať so zvukom pomocou progamovacieho jazyku *Python* a naučil sa lepšie využívať knižnice *numpy, scipy*. Generácia 4 kosínusoviek a filtrácia zvuku prebehla úspešne a výsledok je krásne počuť vo výstupnom súbore. [1]

Využité zdroje

[1] Discrete Fourier Transform (DFT). 2020. Dostupné z: https://towardsdatascience.com/fast-fourier-transform-937926e591cb

[2] Numpy and Scipy Documentation. 2008-2021. Dostupné z: https://docs.scipy.org/doc/

[3] Žmolíková, K.: Jupyter Notebook python zápisky. 2021. Dostupné z: https://nbviewer.org/github/zmolikova/ISS_project_study_phase/blob/master/Zvuk_spektra_filtrace.ipynb#