VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Protokol k projektu

ISS - Signály a systémy

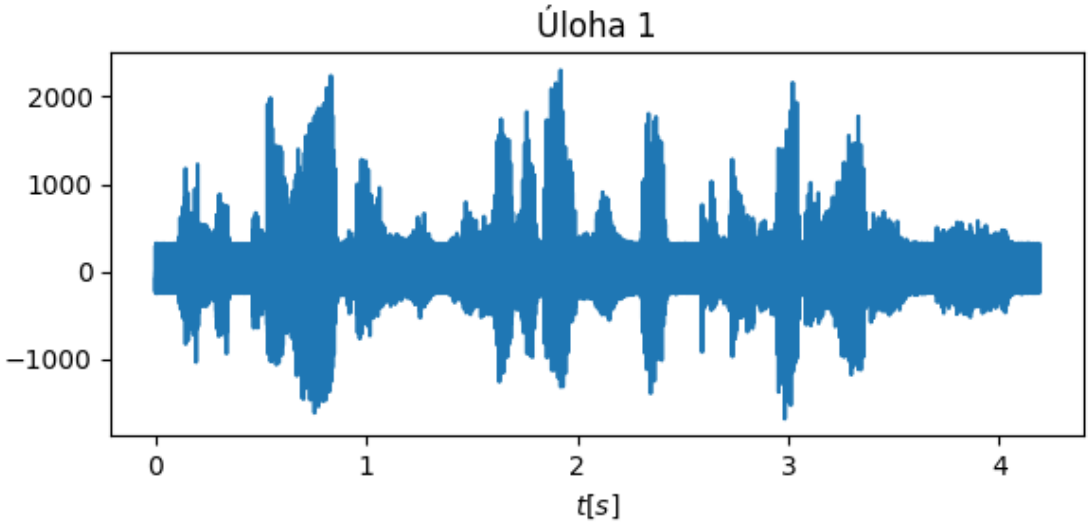
23.12.2021 Lukáš Kaprál (xkapra00)

Obsah

1 Odpovědi na otázky - Standardní zadání

1.1 Základy

Prvně si signál načtu pomocí funkce **wavfile.read()**. Vzorkovací frekvence se uloží do proměnné **fs** a data signálu se uloží do proměnné **data**. Poté si jeho délku ve vzorcích můžu vytisknout jako **data.size**, v mém případě je počet vzorků **67072**, a jeho délku v sekundách jako **data.size/fs**, délka nahrávky je **4.192s**. Minimální a maximální hodnoty lze zjistit pomocí funkcí **min(data)** (**-1682**) a **max(data)** (**2302**)

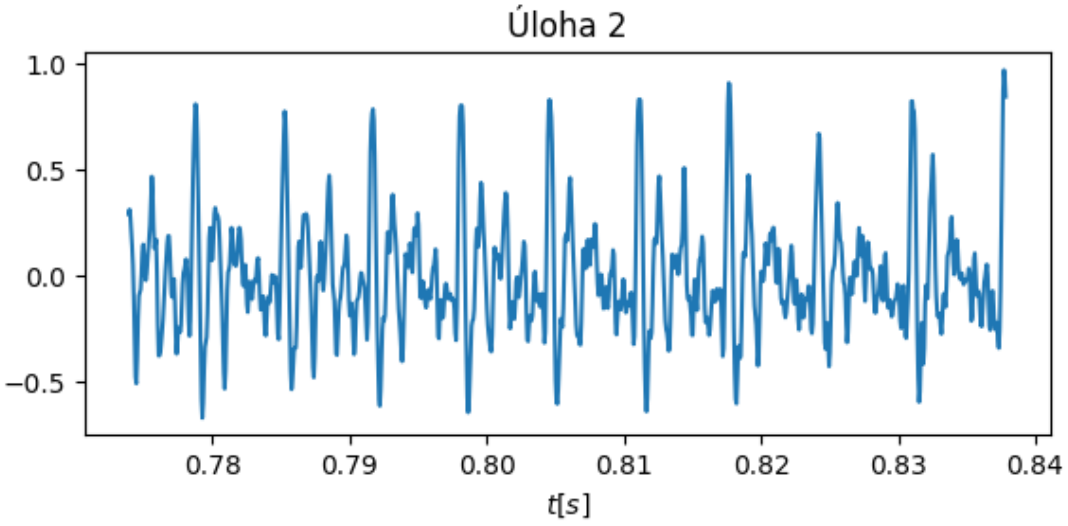


Obrázek 1: Graf vstupního signálu

1.2 Předzpracování rámce

Před ustředněním si spočítám průměrnou hodnotu tak, že projdu hodnoty všech vzorků, přičítám je k proměnné **avg** a na konec vydělím proměnnou **avg** počtem vzorků. Poté od proměnné **data** pouze odečtu hodnotu proměnné **avg**. Pro normalizaci vydělím **data** hodnotou **max(data)**.

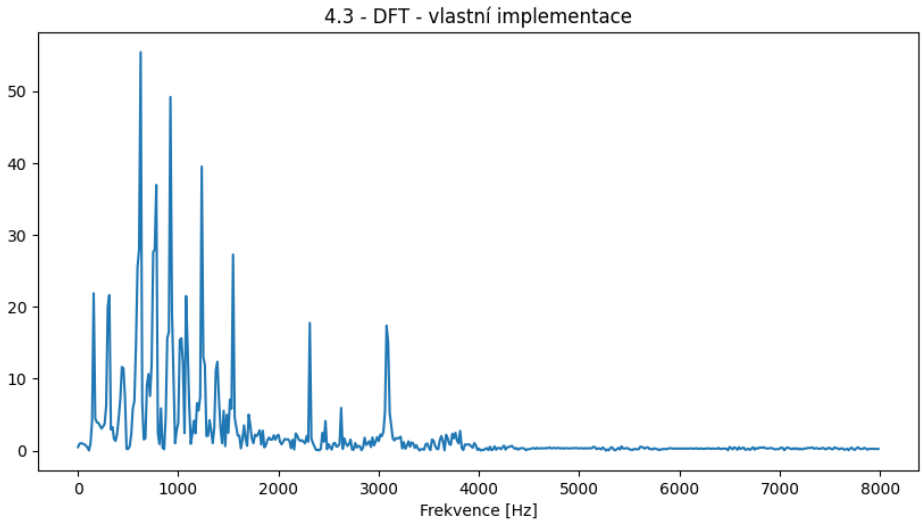
Pro rozdělení dat do rámců si v cyklu si projdu všechny vzorky, načtu si jich vždy 1024, tyto vzorky poté přidám do výsledného pole. Jako další krok se vrátím o 512 vzorků nazpět. Takto projdu všechny hodnoty a na konci mám pole, ve kterém mám 130 rámců, každý po 1024 vzorcích.



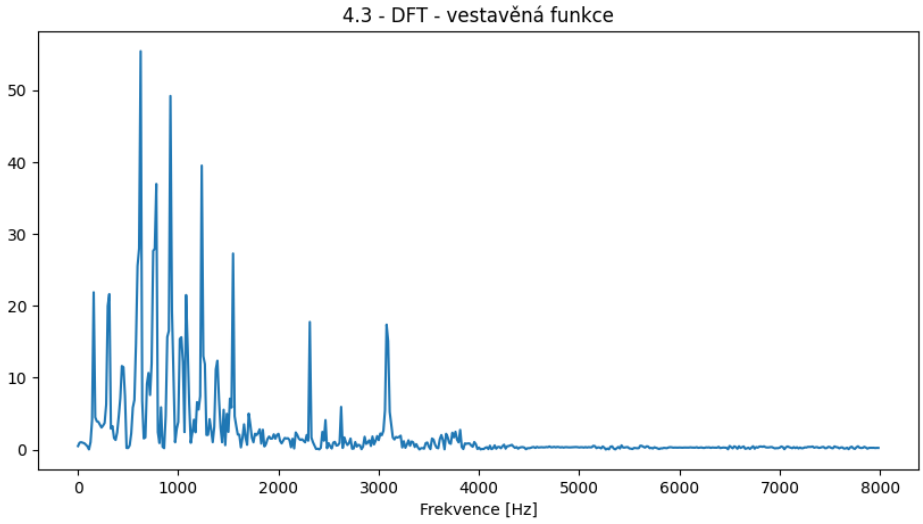
Obrázek 2: Graf znělého rámce

1.3 DFT

Pro DFT využívám 2 pomocné funkce. První je **gen_basis()**, která pomocí vzorečku $\mathbf{e}^{-j*2*\pi*n*\frac{k}{1024}}$ vypočítá bázi pro každý vzorek dat. Druhá pomocná funkce je **dft_with_basis()**, která vynásobí bázi a vektor signálu. Ve finální funkci si v cyklu projdu opět všechny hodnoty dat, použiji vždyk funkce **gen_basis()** a **dft_with_basis()** a výslednou hodnotu opět uložím do pole. Jelikož je graf DFT symetrický, vezmu pouze první polovinu a tu zobrazím. Pro porovnání zobrazím i graf pomocí vestavěné funkce **numpy.fft.fft**. Pro jistotu pomocí funkce **numpy.allclose()** zjistím, zda jsou výsledné hodnoty přibližně stejné. Funkce
`bool = numpyp.allclose(matr, matr2, rtol=1e-05, atol=1e-08)`, kde **matr** jsou hodnoty vlastní funkce a **matr2** jsou hodnoty vestavěné funkce, poté vrací **True**.



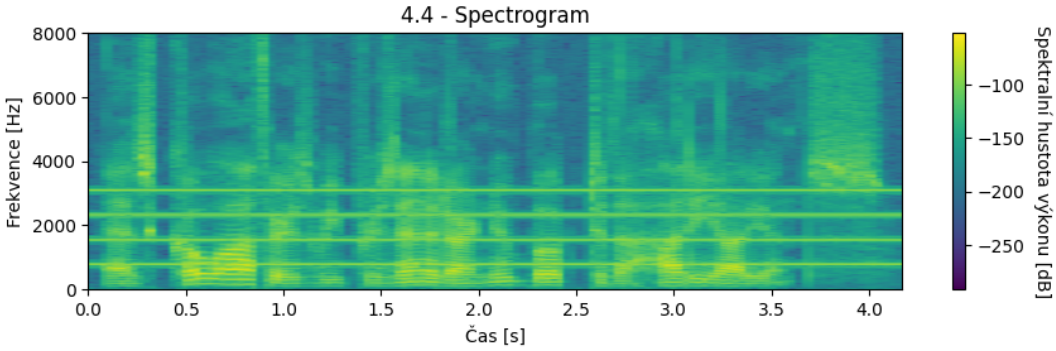
Obrázek 3: Graf vlastní implementace DFT



Obrázek 4: Graf vestavěné funkce DFT

1.4 Spektrogram

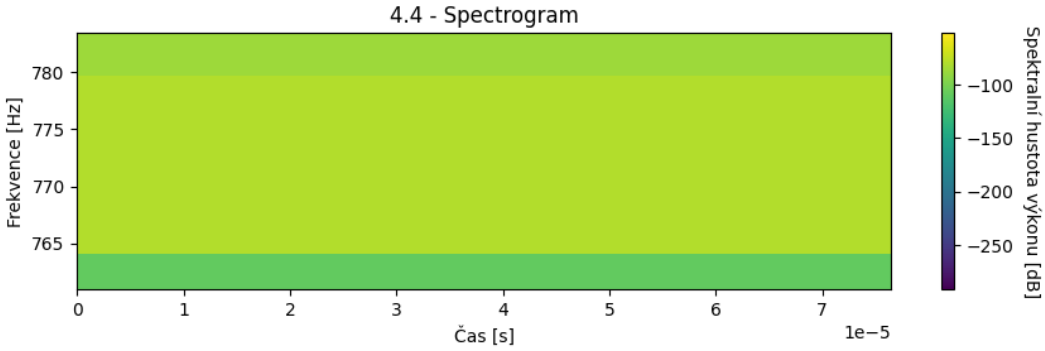
Pro rozdělení signálu do rámců po 1024 vzorcích, s překrytím 512 a následým provedením spektra nad každým rámcem, jsem použil funci **spectrogram()**, s argumenty **data**, neboli hodnotami jednotlivých vzorků, $\frac{fs}{2}$, jelikož jsme měli pracovat pouze s polovinou vzorkovací frekvence. Poté jsem pro rozdělení použil volitelný parametr **nperseg = 1024**, který rozdělil data do rámců po 1024 vzorcích. Pro překrytí sloužil poslední parametr **noverlap = 512**. Tato funkce mi, do proměnné **sgr**, vrátila spectrogram původních dat. Stačilo už pouze vzít tento spectrogram, vynásobit jej pomocí vzorce $\mathbf{10}\log_{10}(|\mathbf{sgr}|^2)$ a výsledný spectrogram zobrazit.



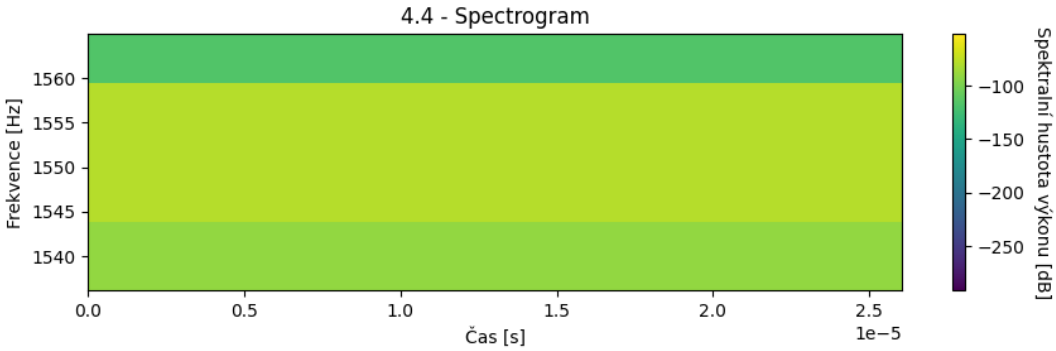
Obrázek 5: Graf spectrogramu

1.5 Určení rušivých frekvencí

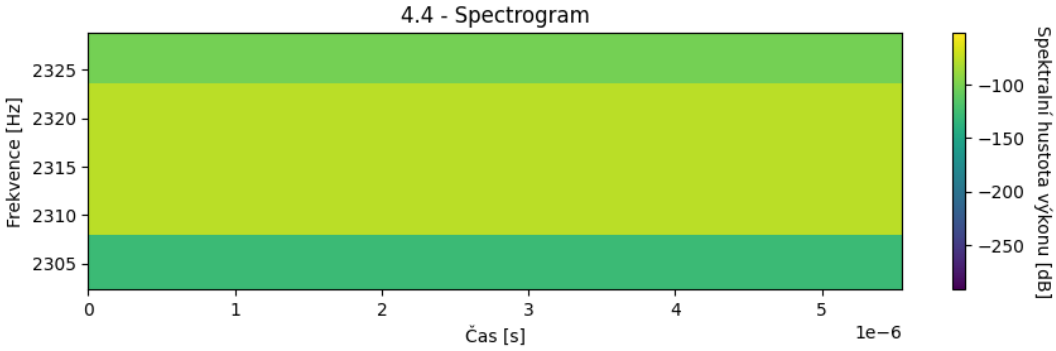
Frekvence jsem určil ručně pomocí přiblížení na rušivou frekvenci. Tímto jsem dostal přibližné frekvence. Poté jsem pro jistotu vytvořil 2 pole, do jednoho jsem zapsal své přibližné hodnoty, a do druhého očekávané hodnoty. Pole **real_values** obsahuje hodnoty 775, 1555, 2320, 3090. Očekávané hodnoty jsou 775, 1550, 2325, 3100.



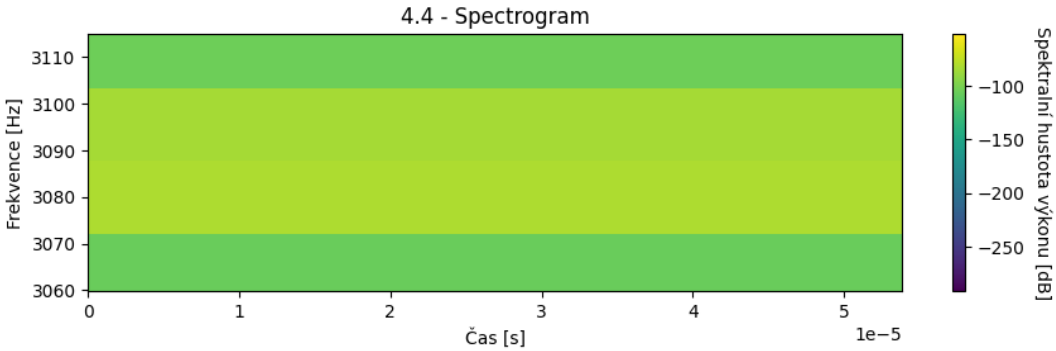
Obrázek 6: Frekvence 1. rušivé komponenty



Obrázek 7: Frekvence 2. rušivé komponenty



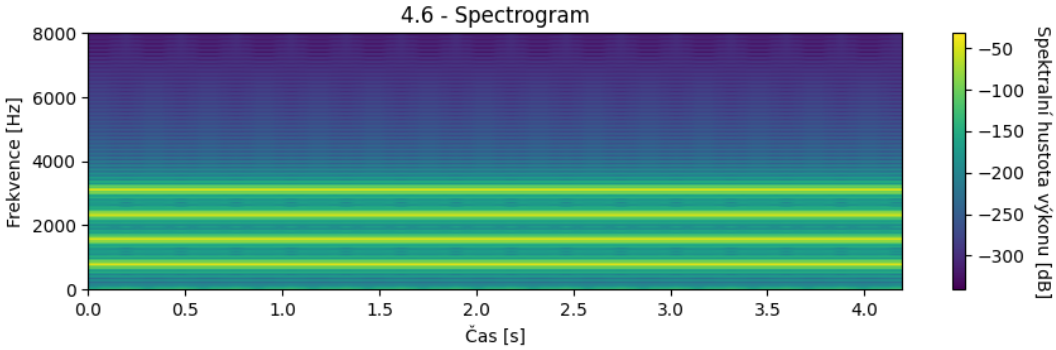
Obrázek 8: Frekvence 3. rušivé komponenty



Obrázek 9: Frekvence 4. rušivé komponenty

1.6 Generování signálu

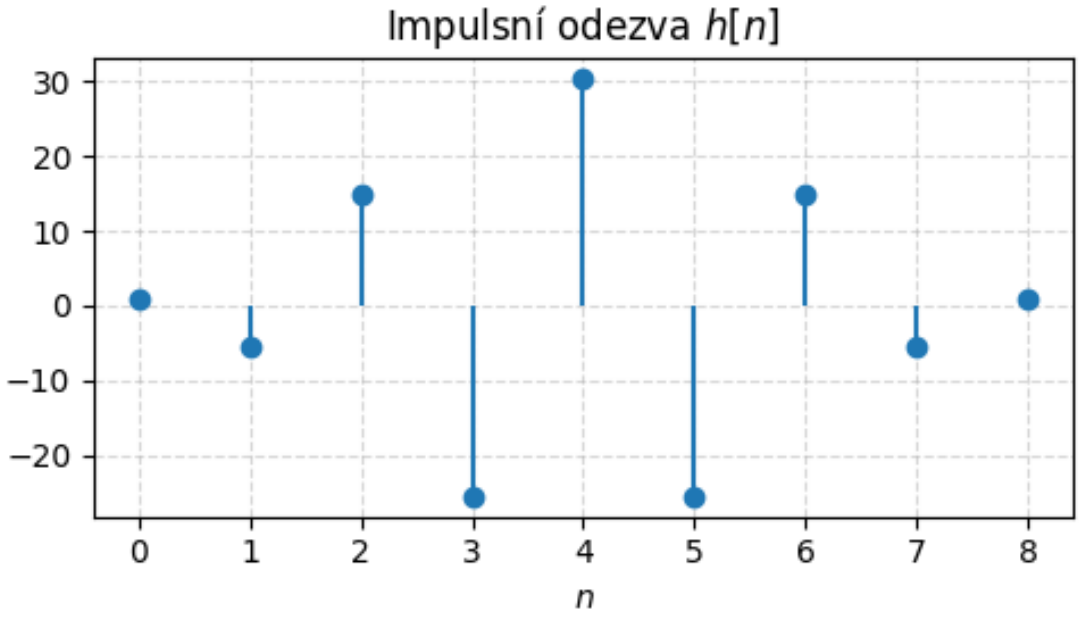
Na začátek jsem si vytvořil pole, s časovými úsekami (0/16000 - 67071/16000). Jako další krok jsem si vytvořil čtyři cosinusovky, každou pomocí vzorečku **cos(2π*(_frekvence_)*(pole hodnot))**. Dále jsem do nového pole **out** vložil součet těchto cosinusovek, čímž jsem si vytvořil finální signál. Nakonec stačilo opět použít funcki **spectrogram**, tentokrát bez parametru **noverlap**. Výsledný spectrogram jsem ještě upravil pomocí vzorce $\mathbf{10}\log_{10}(|\mathbf{sgr}|^2)$ a poté jsem jej zobrazil.



Obrázek 10: Spectrogram signálu se směsí 4 cosinusovek

1.7 Čistící filtr

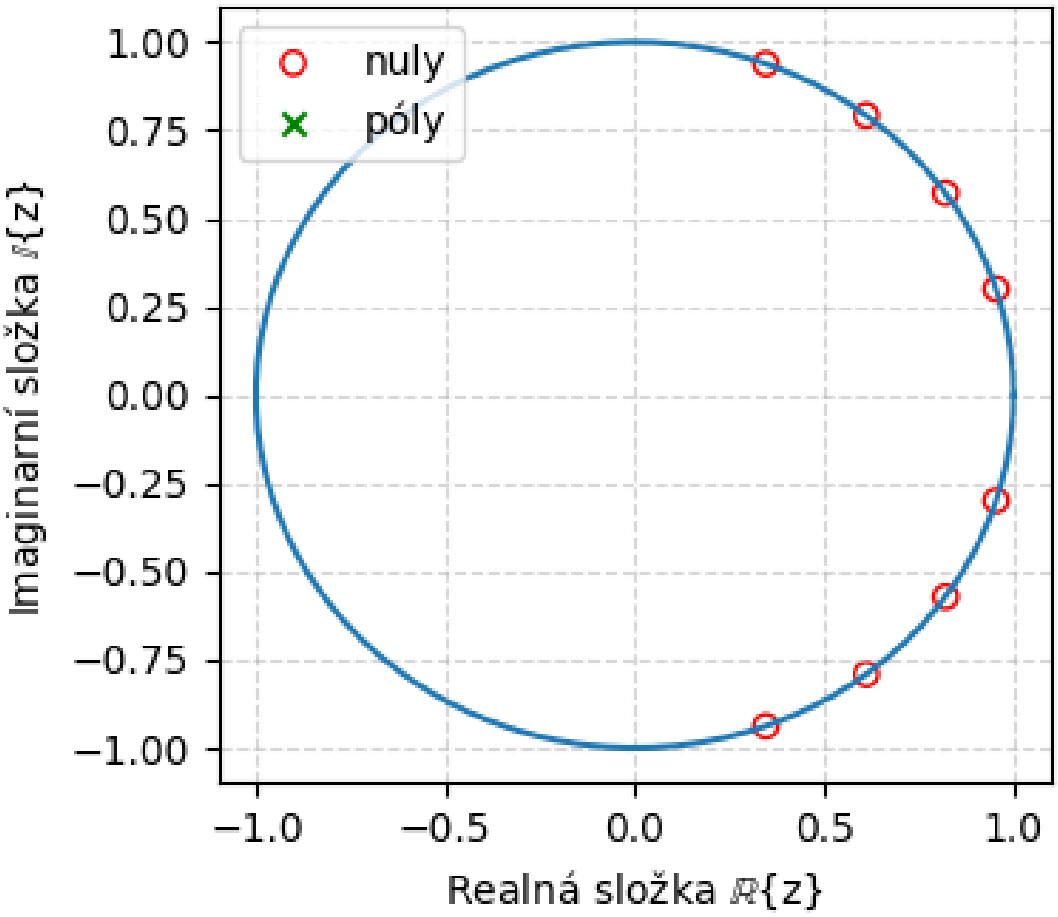
Pro filtr jsem zvolil výrobu filtru v z-rovině. Začal jsem tím, že jsem si do pole uložil frekvence, na kterých se nachází rušivé komponenty. Poté jsem každou hodnotu z tohoto pole upravil podle vzorečku $\omega_k = 2\pi \frac{f_k}{F_s}$ a následně přes $n_k = e^{j\omega_k}$. Tímto jsem dostal pole čtyř bodů, ležících na jednotkové kružnici. K těmto bodům jsem pomocí funkce **numpy.conj()** našel komplexně sdružené body a opět jsem si je uložil do pole. Nakonec jsem na všechny tyto body použil funkci **numpy.poly**(), čímž jsem získal koeficienty filtru, které jsou: **1, -5.46383462, 14.77312342, -25.39058723, 30.19615631, -25.39058723, 14.77312342, -5.46383462, 1**. Po vyfiltrování došlo ke zkreslení signálu, což může být tím, že při filtrování signálu dochází k zesílení vysokých frekvencí natolik, že po normalizování bude reálná řeč příliš slabá.



Obrázek 11: Impulzní odezva filtru

1.8 Nulové body a póly

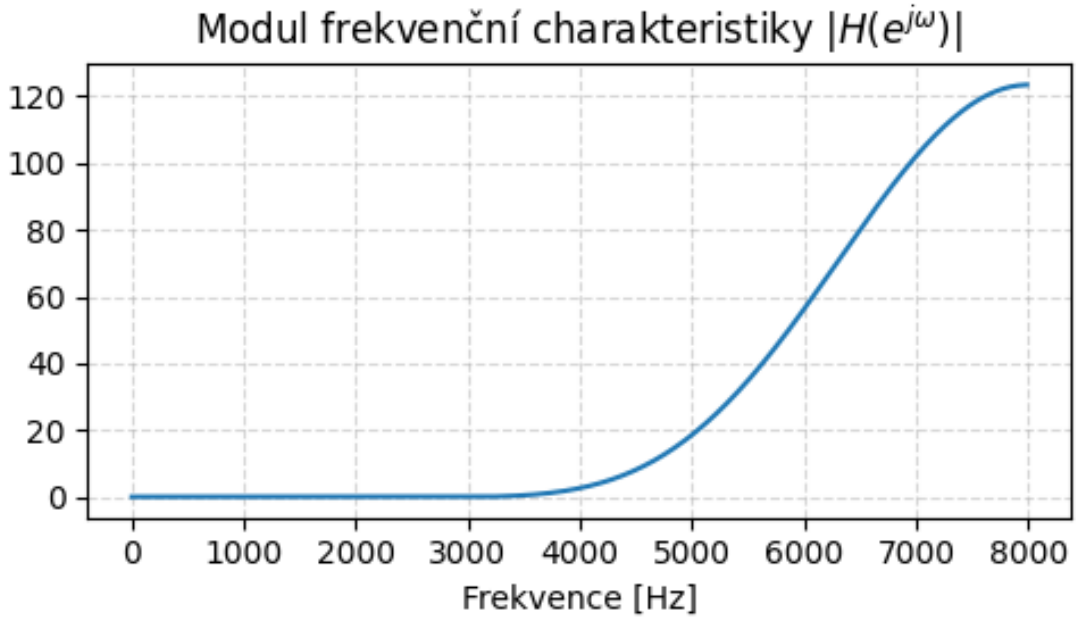
Pro získání nulových bodů a pólů jsem využil funkci **signal.tf2zpk**(). Jelikož jsem pro filtrování zvolil metodu **4.7.1**, vrátila mi funkce prázdé pole pólů. Pro zobrazení jsem využil funkci ze stránky **Jupyter/Zmolikova - filtrace**, ze sekce **Filtrace**, příklad **In[15]**.



Obrázek 12: Nulové body filtru

1.9 Frekvenční charakteristika

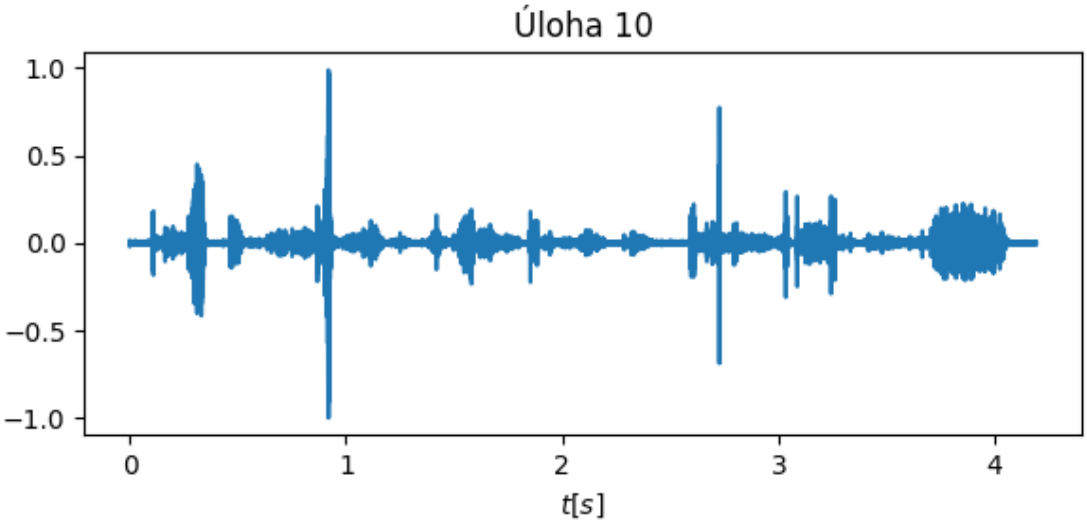
Na výpočet frekvenční charakteristiky jsem použil funcki **signal.freqz()** s argumenty **filter**, což je pole koeficientů filtru, a [1], což při dělení vlastně eliminuje jmenovatele. Tato funkce mi vrátí 2 hodnoty do proměnných w a H. Do proměnné w uloží frekvence, na kterých bylo vypočteno H. Proměnná H poté obsahuje frekvenční odezvu. Stačilo už pouze zobrazit graf. Pro zobrazení jsem použil kód ze stránky **Jupyter/Zmolikova - filtrace**, ze sekce **Filtrace**, příklad **In[12]**.



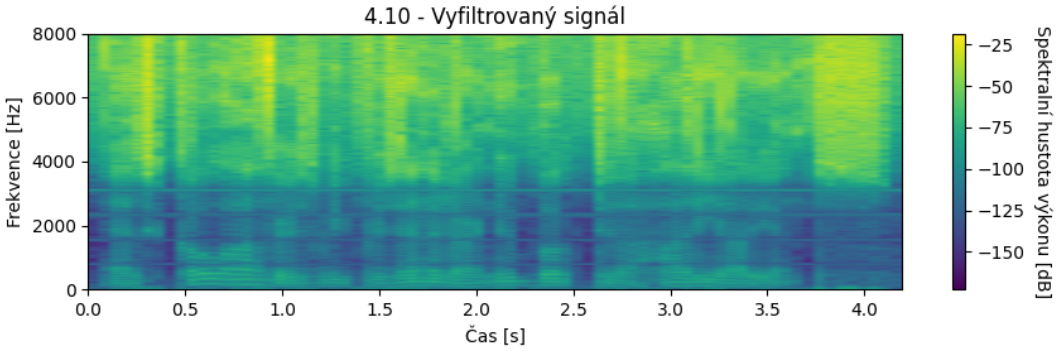
Obrázek 13: Frekvenční charakteristika filtru

1.10 Filtrace

Pro finální filtraci jsem si prvně načetl data originální nahrávky, koeficienty filtru a vzorkovací frekvenci. Jako první jsem opět ustřednil a normalizoval originální signál pomocí funkce, popsané v sekci Předzpracování rámce. Poté jsem použil funkci **signal.lfilter()** s argumenty **filter**, neboli koeficienty filtru, **[1]** a **data**, což jsou hodnoty originální nahrávky. Tato funkce mi vrátila vyfiltrovaný signál, který jsem poté opět ustřednil a normalizoval. Prvně jsem si zobrazil graf, abych si ověřil, že je v dynamickém rozsahu -1 až 1. Poté stačilo pouze vytvořit soubor s názvem **clean_z.wav** pomocí funkce **wavfile.write** s argumenty **"../audio/clean_z.wav", 16000, final**, kde **16000** je vzorkovací frekvence a **final** je výsledný, vyfiltrovaný signál.



Obrázek 14: Graf vyfiltrovaného signálu



Obrázek 15: Spectrogram vyfiltrovaného signálu

2 Závěr

Z originálního signálu zmizel rušivý signál, avšak došlo ke zhoršení kvality zvuku. Mohlo by to být důsledkem důvodu popsaného v sekci Čistící filtr.

3 Zdroje

Většina mých zdojů pochází přímo z definicí funkcí knihoven **numpy** a **scipy.signal**. Největší části projektu jsem převzal přímo **příkladů Katky Žmolíkové**.