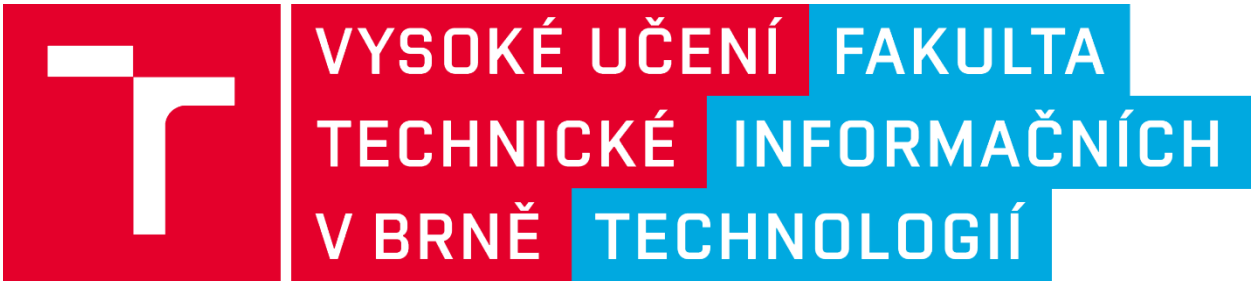


Protokol s riešením

Projekt - Signály a systémy

30.12.2021 **Leopold Nemček (xnemce07)**

Obsah

1.Používanie scriptu

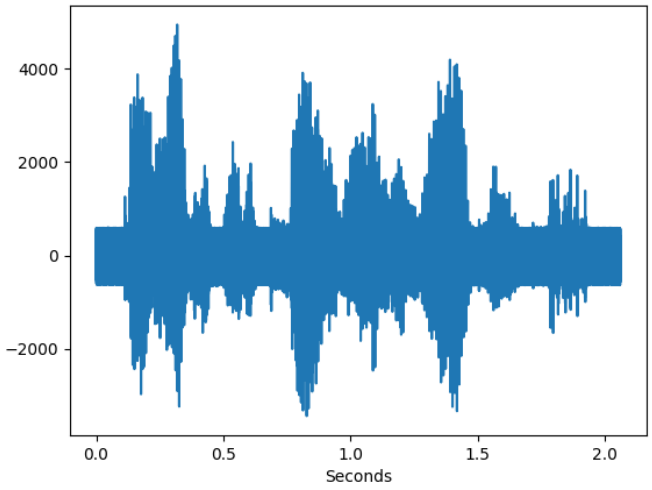
Programová časť riešenia sa nachádza v súbore src/main.py. Po spustení programu sa vypíše prompt, ktorý vyzve užívateľa na zadanie čísla úlohy, ktorej riešenie si chce zobraziť. Ak si chce zobraziť riešenie viacerých úloh, zadá čísla týchto úloh oddelené čiarkou, alebo ak si chce zobraziť riešenie všetkých úloh zadá “all”. Toto riešenie som využil preto, aby som sa nemusel “preklikávať” cez všetky zobrazované grafy pri kontrole neskorších úloh.

2.Riešenie

1. Úloha

Na načítanie vstupného signálu som využil funkciu *read* zo *scipy.io.wavfile,* ktorá mi vrátila jeho vzorkovaciu frekvenciu – 16000 vzoriek za sekundu a pole s dátami. Veľkosť tohto poľa (dĺžka vo vzorkách) je 32973 vzoriek a dĺžka v sekundách je 32973/16000 ~= 2.06 sekúnd. Jeho maximálna hodnota je 4937 a minimálna -3436.

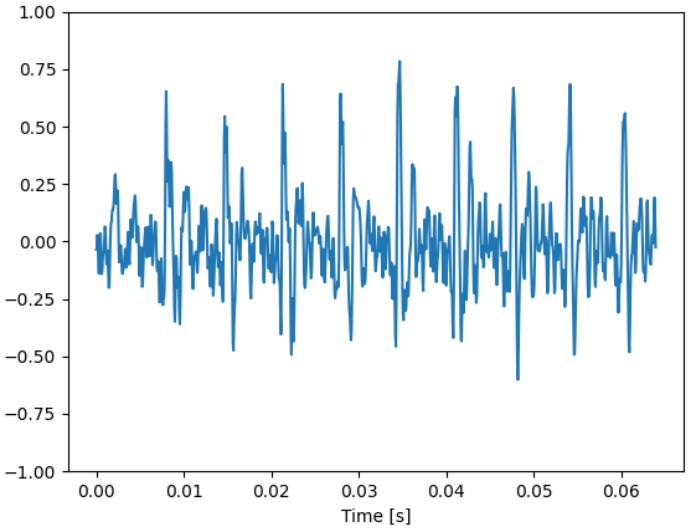
1: Vstupný signál



2. Úloha

Po normalizovaní a rozdelení signálu na rámce som sa pokúsil zamerať na prvé “o”, ktoré človek v nahrávke vysloví v slove “don’t” podľa grafu som si vyčítal, že by sa mohlo nachádzať približne v čase 0.2 sekundy – teda v 3. až 4. segmente. Tieto segmenty som si potom porovnal a 4. segment vyzeral omnoho preiodickejšie, zvolil som teda ten.

2: 4.segment z nahrávky



3. Úloha

Moja implementácia DFT je rozdelená do dvoch funkcií – *myDFTmat* vytvorí maticu bází podľa zadanej veľkosti a *myDFT* zavolá prvú funkciu a vynásobí zadané pole navrátenou maticou.

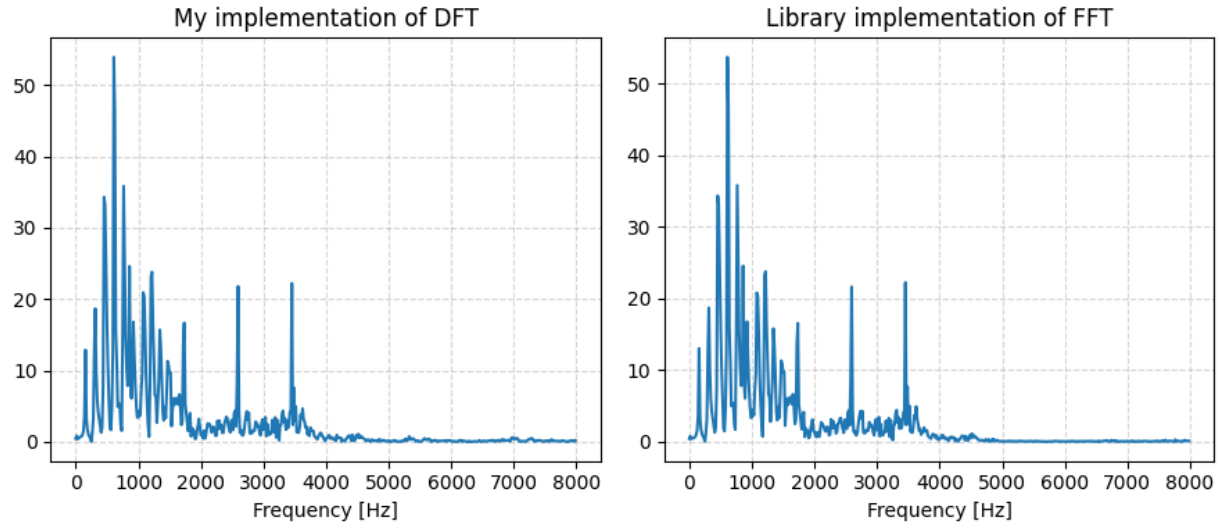
3: Moja implementácia DFT

```
def myDFTmat(N):
    Wn = cm.exp(-1j* 2 * pi / N )
    DFTmat = np.zeros((N,N), dtype='complex')
    for n in range(0,N-1):
        Wnn = pow(Wn,n)
        for k in range(0,n):
            res = pow(Wnn,k)
            DFTmat[n][k] = res
            DFTmat[k][n] = res
    return DFTmat

def myDFT(x):
    mat = myDFTmat(len(x))
    return np.matmul(mat,x)
```

Pre čo najnížší počet operácií pri vytváraní matice bází sa najprv predpočíta hodnota $e^{-j\frac{2\pi}{N}}$, ktorá sa potom umocňuje podľa pozície v tabuľke. Taktiež sa počíta iba polovica matice, keďže je matica bází symetrická.

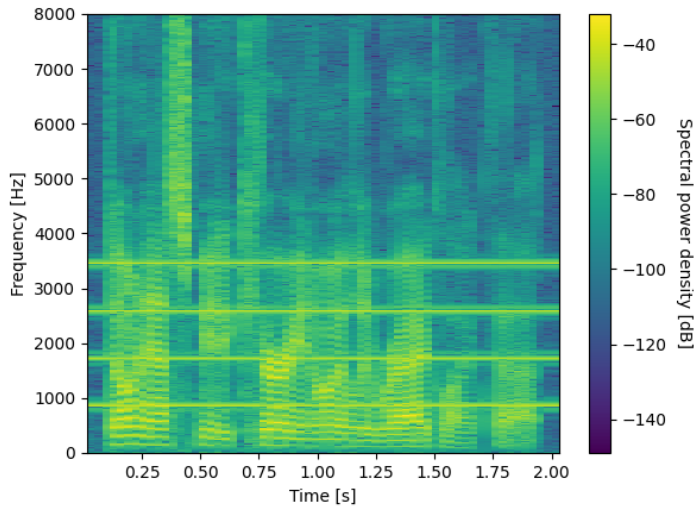
4: Porovnanie mojej implementácie DFT a FFT z knižnice numpy



4. Úloha

Na výpočet spektogramu som využil funkciu *spectogram* zo *scipy.signal.*

5: Logaritmický výkonový spektrogram



5. Úloha

V spektograme je vidno, že sú cez celú nahrávku 4 signály na stálych frekvenciách, to teda musia byť pridané cosinusovky. Frekvenciu rušivých cosinusoviek som odčítal zo spektogramu na frekvenciách 860Hz, 1720Hz, 2580Hz a 3440Hz.

6. Úloha

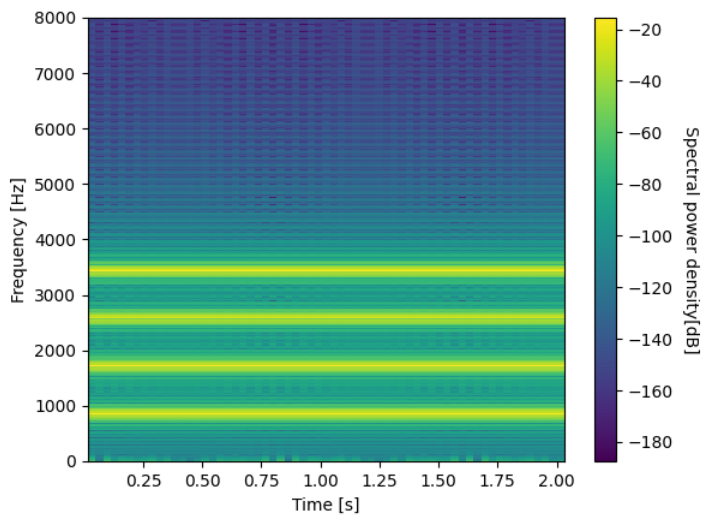
Výsledný signál som vygeneroval postupným pričítaním cosinusoviek s odčítanými hodnotami. Každú cosinusovku som vygeneroval pomocou vzorca $y[n] = cos\left(\frac{f}{Fs} * 2\pi * x[n]\right)$

6: Generovanie rušivého signálu (freqs - pole s frekvenciami rušivých cosinusoviek)

```
fx = np.arange(0, len(data), 1)
fy = 0

for freq in freqs:
    fy += np.cos(freq/fs * 2 * np.pi * fx)
```

7: Spektogram rušivého signálu - môžeme vidieť, že frekvencie sú zhodné s rušivými frekvenciami z predchádzajúcej úlohy



Pri ukladaní do súboru som výsledný signál vydelil 50, pretože bol príliš hlasný, inak som si však posluchom potvrdil, že som rušivé frekvencie určil správne.

7. Úloha

Pre vytvorenie filrtu som zvolil 3. metódu – Vytvorenie 4 pásmových zádrží. Na to som využil funkcie *buttord* a *butter* zo *scipy.signal.* Najlepšie výsledky sa mi podarilo docieliť so šírkou záverného pásma 20Hz a šírkou prechodu do prepustného pásma 50Hz. Po vytvorení každej zádrže som skonvoluoval obe zložky, aby som mal všetky zádrže v jednom filtri.

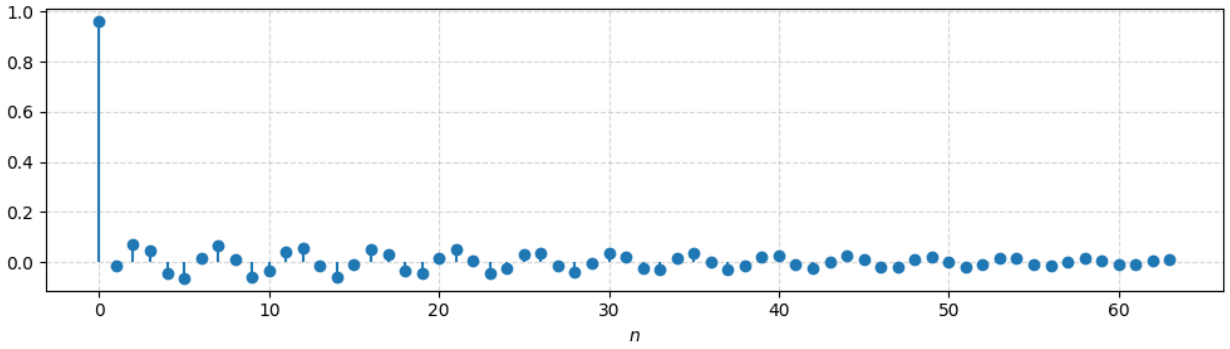
8: Implementácia vytvárania filtru

```
wp_segment = 50
ws_segment = 10

filt_a = 1
filt_b = 1

for freq in freqs:
    N, wn = signal.buttord([freq-wp_segment, freq+wp_segment],
                           [freq-ws_segment, freq+ws_segment], 3, 40, False, fs)
    b, a = signal.butter(N, wn, 'bandstop', False, 'ba', fs)
    filt_a = np.convolve(filt_a, a)
    filt_b = np.convolve(filt_b, b)
```

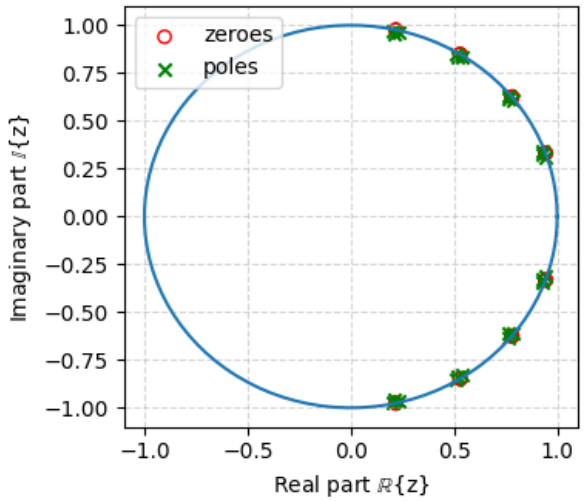
9: Impulzná odozva navrhnutého filtru



8. Úloha

Na výpočet nulových bodov a pólov som využil funkciu *tf2zpk* zo *scipy.signal*.

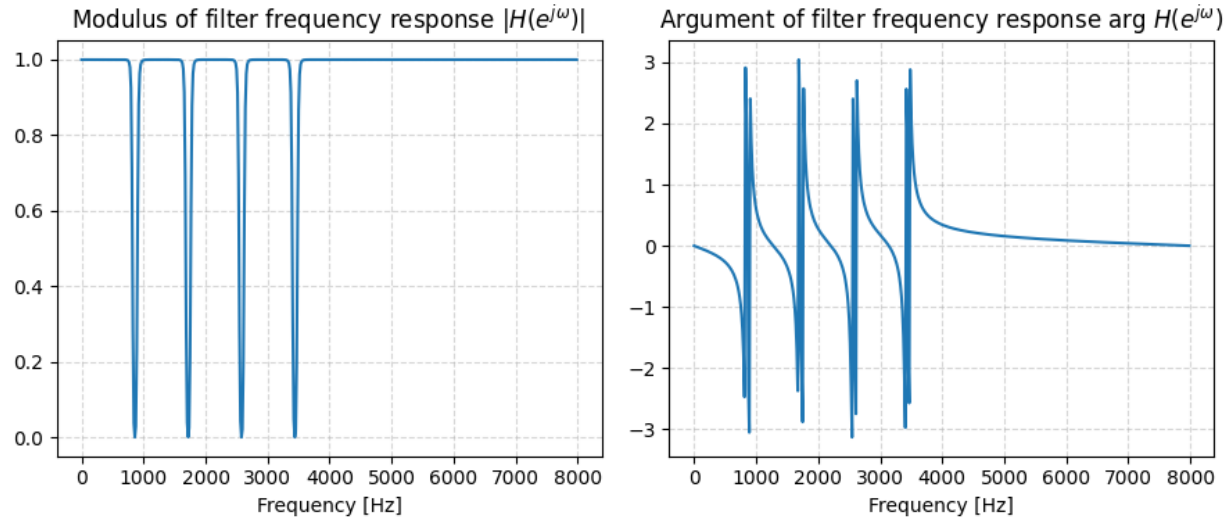
10: Nuly a póly navrhnutého filtru



9. Úloha

Na výpočet frekvenčnej charakteristiky filtru som využil funkciu *freqz* zo *scipy.signal*.

11: Frekvenčná charakteristika navrhnutého filtru

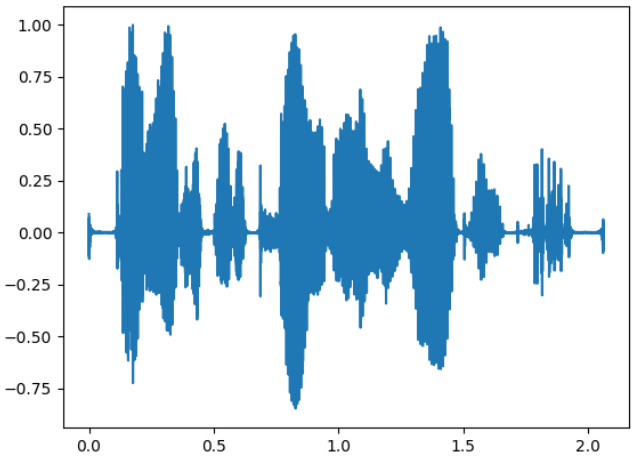


Na obrázku môžeme vidieť, že filter skutočne potláča odčítané frekvencie.

10. Úloha

Po odskúšaní funkcií *lfilter* a *filtfilt* zo *scipy.signal* som sa rozhodol pre použitie *filtfilt*, ktorá signál prefiltruje 2-krát, raz odpredu a raz odzadu, kvôli lepšiemu výsledku. Dáta som po prefiltrovaní pre istotu normalizoval.

12: Výsledný signál



3. Záver

Až na veľmi jemné pípnutie na začiatku a konci nahrávky sa mi rušivý signál podarilo odstrániť a preto si myslím, že sa mi filter podarilo navrhnúť správne. Pípnutia som mohol síce stlmiť po prefiltrovaní, keďže na okrajoch nahrávky je inak ticho, no rozhodol som sa ich ponechať kvôli “dvôveryhodnosti” výsledku. Mslím si, že v nahrávke zostali kvôli tomu, ako digitálne filtre fungujú, teda, že používajú predošlé vzorky signálu – ktoré však na začiatku signálu nie sú a sú za ne teda doplnené 0. Ak by som chcel tieto pípnutia odstrániť bez “umelého” tlmenia okrajov alebo skracovania nahrávky, skúsil by som pred filtráciou pridať na začiatok a koniec niekoľko vzoriek nájdeného rušivého signálu podľa dĺžky navrhnutého filtru, signál by som prefiltroval a nakoniec by som pridané okraje opäť odstránil.