1. & 2. Úkol

Jako nahrávací program byl použit Audacity[1]. Délka a počet vzorků byly zjištěny pomocí nastroje soxi[8].

Název	Délka [s]	Počet vzorků	Bitová šířka	Vzorkovací frekvence
maskoff_tone.wav	1.0	16000	16 bitů	16 kHz
maskon_tone.wav	1.0	16000	16 bitů	16 kHz
maskoff_sentence.wav	03.51	56669	16 bitů	16 kHz
maskon_sentence.wav	03.54	56161	16 bitů	16 kHz

3. Úkol

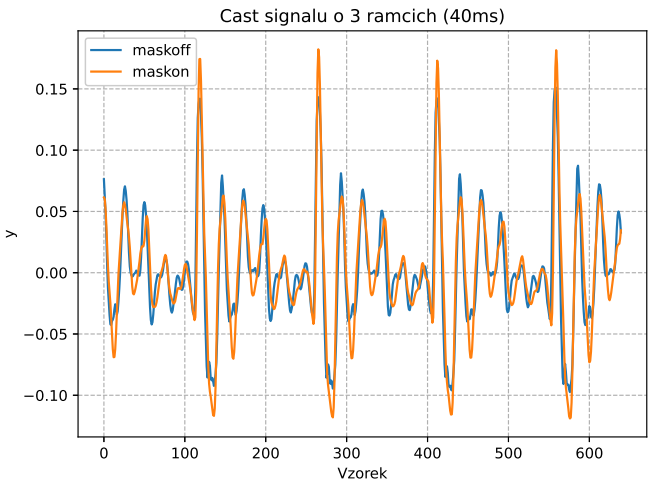
K načtení signálů byla použita funkce z knihovny `scipy`[5]. Z nahrávek byly ručně vybrány sekundové rámce kde se signál nejvíce podobá.

```
1    off_tone = wavfile.read('../audio/maskoff_tone.wav')[1]
2    off_tone = off_tone[:16000]
```

`Ustřednění`, neboli odstranění stejnosměrné složky[2] proběhlo odečtením střední hodnoty z obou signálů. `Normalizace` [10] proběhla dělením vstupních signálů hodnotou 2^{15}. `Velikost rámce` ve vzorcích se získala pomocí vzorce $framesize = duration * Fs$. V tomto případě to je $0.02 * 16000$ a velikost vzorce je 320.

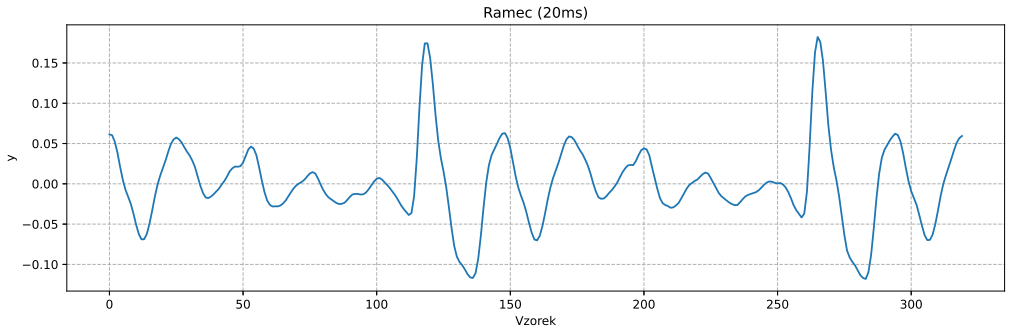
```
1    # Normalizace
2    on_tone_uprav = on_tone / 2**15
3    off_tone_uprav = off_tone / 2**15
4    # Ustredneni
5    on_tone_uprav -= np.mean(on_tone_uprav)
6    off_tone_uprav -= np.mean(off_tone_uprav)
7    # Rozdeleni na ramce
8    framesize= int(0.020 * 16000)
9    rows, cols = (99, framesize)
```

Signály byly `rozděleny` na 99 rámců po 320 vzorcích, které se překrývají po 10 milisekundách.

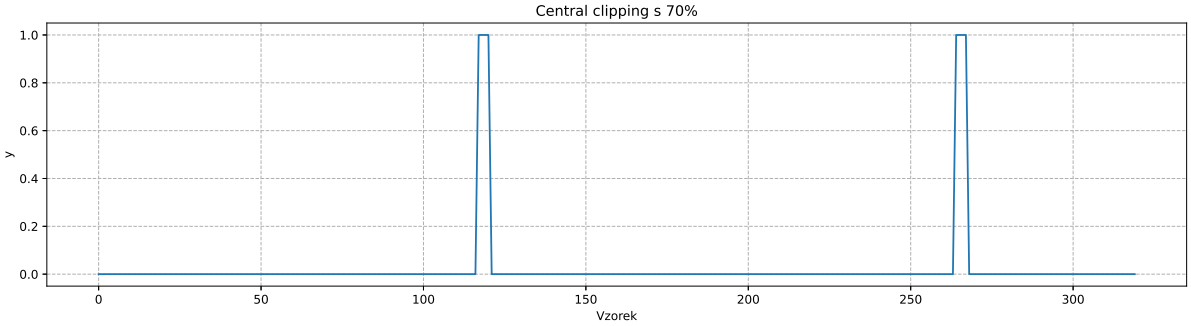


Obrázek 1: Porovnání 3 rámců upravených signálů

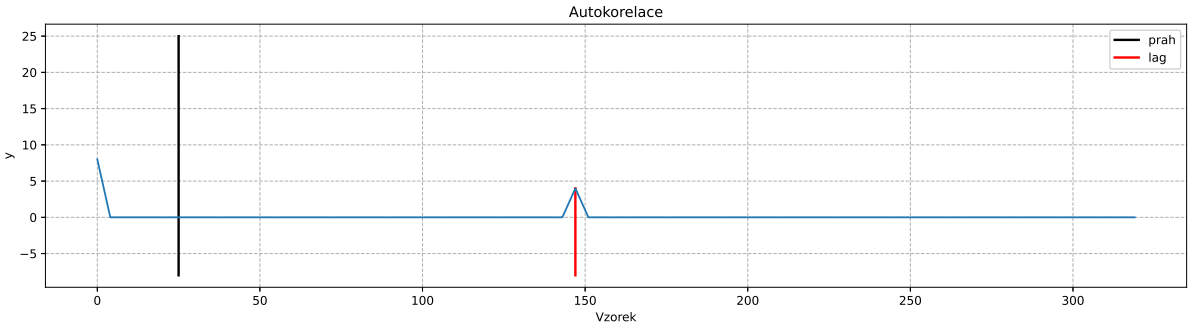
4. Úkol



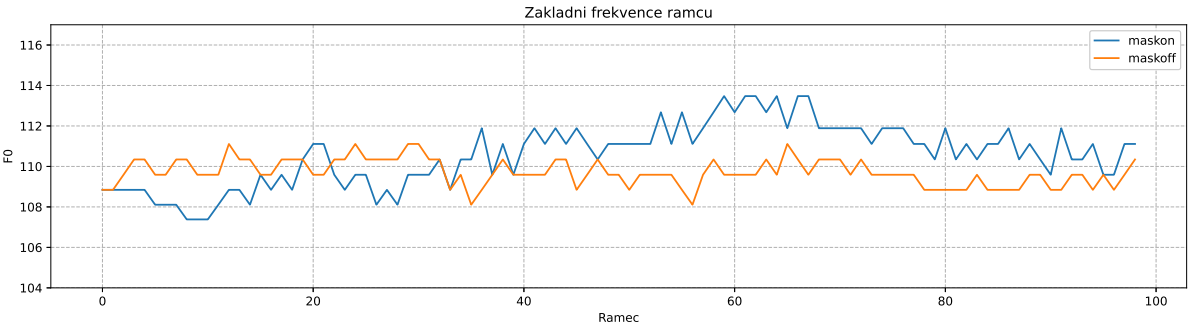
Obrázek 2: Normalizovaný a ustředněný rámec



Obrázek 3: Rámec upravený pomocí 70% centre clipping



Obrázek 4: Autokorelace



Obrázek 5: Základní frekvence rámců

Výsledky:

Název	Rozptyl	Střední hodnota
maskoff_tone.wav	2.1932854	110.487810
maskon_tone.wav	0.4512025	109.669025

Pro každý rámec bylo zjištěno jeho maximum pro využití při center clippingu [7]. Každý vzorek rámce byl podle této hodnoty kontrolován pro aplikaci center clippingu.

```
    ramecOnMax = (max(np.absolute(on_tone_ramce[ramecCntr]))*0.7)
    if on_tone_ramce[ramecCntr][vzorekCntr] > ramecOnMax:
        on_tone_clipped[ramecCntr][vzorekCntr] = 1
    elif on_tone_ramce[ramecCntr][vzorekCntr] < -ramecOnMax:
        on_tone_clipped[ramecCntr][vzorekCntr] = -1
    else:
        on_tone_clipped[ramecCntr][vzorekCntr] = 0
```

Autokorelační metoda je implentována dle vzorečku[7]:

$$R(m) = \sum_{n=o}^{N-1-m} s(n)s(n+m)$$

Implementace autokorelace:

```
    for ramecCntr in range(rows):
        N=320
        for vzorekCntr in range(cols):
            m = vzorekCntr
            horniIndex=N-1-m
            #Suma
            for n in range(horniIndex):
                hodnota = on_tone_clipped[ramecCntr][n]*on_tone_clipped[ramecCntr][n+m]
                on_tone_correlated[ramecCntr][m] += hodnota
                hodnota = off_tone_clipped[ramecCntr][n]*off_tone_clipped[ramecCntr][n+m]
                off_tone_correlated[ramecCntr][m] += hodnota
```

Dle vztahu $lag = argmax(R(m))$ [7] byly zjištěny lagy pro každý rámec.

```
    for ramecCntr in range(rows):
        for vzorekCntr in range(prah,cols):
            if on_tone_correlated[ramecCntr][vzorekCntr] >  on_tone_correlated[ramecCntr][lags_on[ramecCntr]]:
                lags_on[ramecCntr] = vzorekCntr
            if off_tone_correlated[ramecCntr][vzorekCntr] >  off_tone_correlated[ramecCntr][lags_off[ramecCntr]]:
                lags_off[ramecCntr] = vzorekCntr
```

`Odpověd na otázku c:` Místo skutečného lagu je často detekován poloviční či několinásobný lag. Velikost změny při chybě by se dala ovlivnit desetinným vzorkováním, kdy signál nadvzorkujeme a následně filtrujeme[7]. Dále se pomocí zjištěných lagů vypočítala základní frekvence pro každý rámec:

```
    for ramecCntr in range(rows):
        fzero_on[ramecCntr] = 16000/lags_on[ramecCntr]
        fzero_off[ramecCntr] = 16000/lags_off[ramecCntr]
```

Pro výpočet střední hodnoty a rozptylu jsou použity funkce np.var() a np.mean()

```
    stredHodnota_on=np.mean(fzero_on)
    stredHodnota_off=np.mean(fzero_off)
    rozptyl_on=np.var(fzero_on)
    rozptyl_off=np.var(fzero_off)
```

5. Úkol

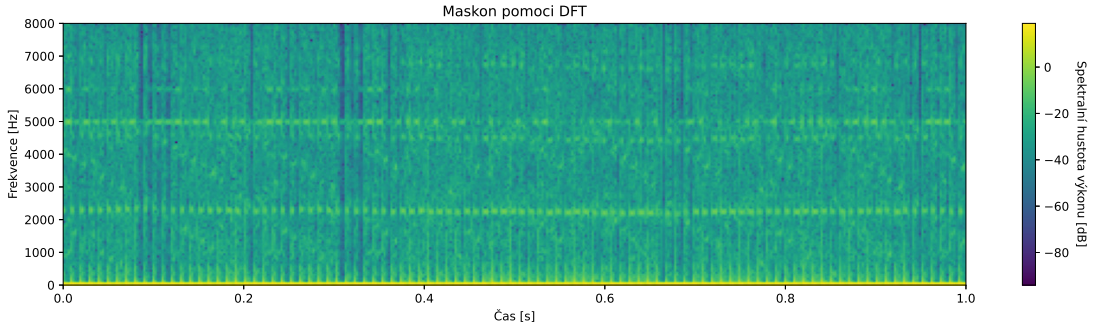
Vlastní implementace DFT[6] a použití funkce použitá na obě nahrávky:

```
    #FFT
    for ramecCntr in range(rows):
        on_tone_fft[ramecCntr]  = np.fft.fft(on_tone_ramce_padded[ramecCntr],n=N)
        off_tone_fft[ramecCntr]  = np.fft.fft(off_tone_ramce_padded[ramecCntr],n=N)
    #DFT
    for ramecCntr in range(rows):
        for vzorekCntr in range(N-1):
            k=vzorekCntr
            for n in range(N-1):
                on_tone_dft[ramecCntr][k] += on_tone_ramce_padded[ramecCntr][n]*np.e**(-2*1j*np.pi*(k/N)*n)
                off_tone_dft[ramecCntr][k] += off_tone_ramce_padded[ramecCntr][n]*np.e**(-2*1j*np.pi*(k/N)*n)
```

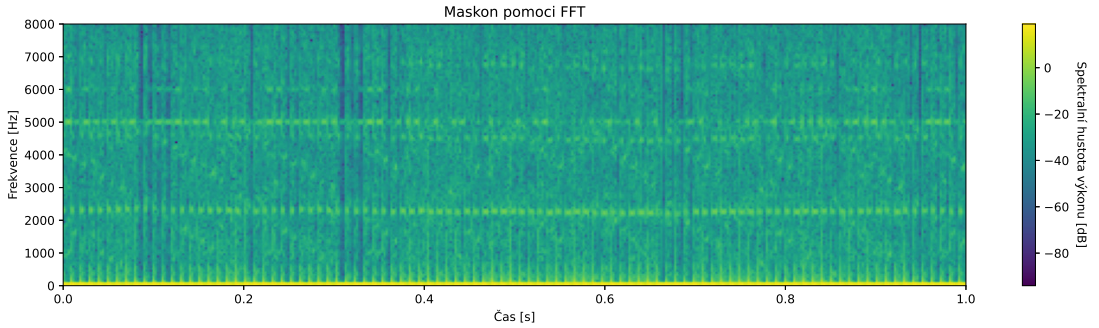
Vypočítané koeficienty pro každý rámec poté byly převedeny do 1D pole a koeficienty byly upraveny logaritmem. Kvůli dělení nulou byla ke každému koeficientu přičtena minimální hodnota.

```
off_tone_dft_full[vzorekCntr] = 10 * np.log10(np.abs((off_tone_dft_full[vzorekCntr])**2)+1e-20)
```

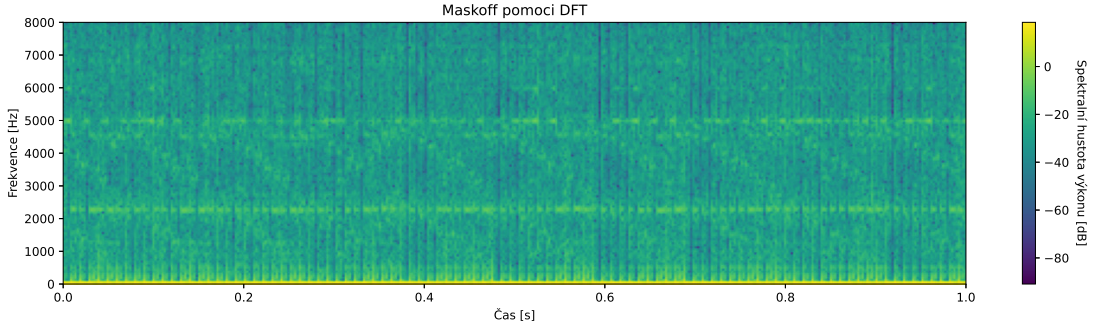
Kvůli symetrii budeme zobrazovat pouze polovinu (512) koeficientů. Porovnání vlastní implementace diskrétní Fourierovy Transformace s funkcí fft z knihovny numpy:



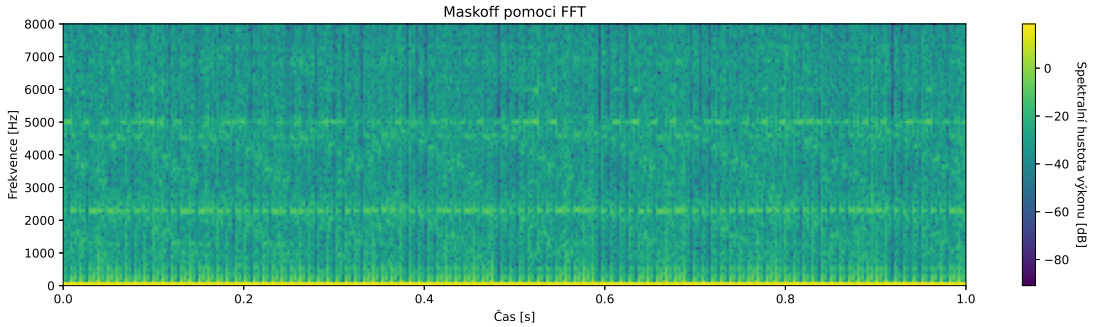
Obrázek 6: Spektogram pomocí Diskrétní Fourierovy Transformace nahrávky s rouškou



Obrázek 7: Spektogram pomocí Rychlé Fourierovy Transformace nahrávky s rouškou



Obrázek 8: Spektogram pomocí Diskrétní Fourierovy Transformace nahrávky bez roušky



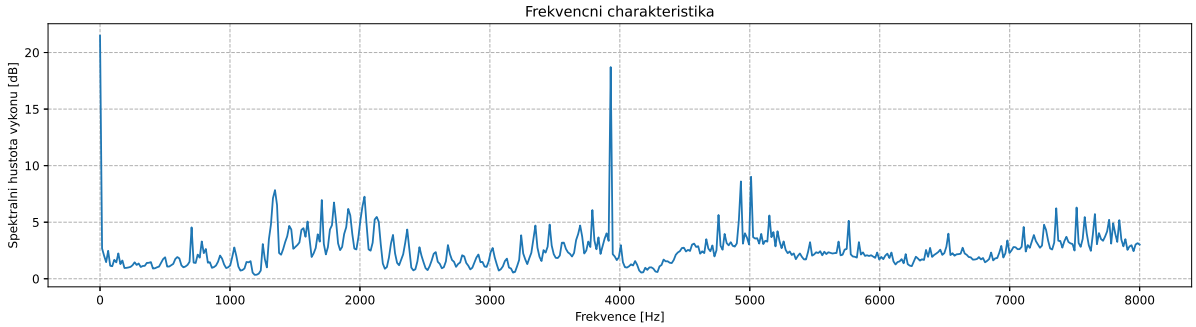
Obrázek 9: Spektogram pomocí Rychlé Fourierovy Transformace nahrávky bez roušky

6. Úkol

Vztah pro výpočet $H(e^{j\omega})$:

$$H(e^{j\omega}) = \frac{\sum_{k=0}^{Q} b_k e^{-jk\omega}}{1 + \sum_{k=1}^{P} a_k e^{-jk\omega}}$$

Diskrétní fourierovu transformaci nahrávek využijeme pro zjištění frekvenční charakteristiky filtru[9]. Podělíme v absolutní hodnotě příslušné koeficienty DFT s roškou a bez roušky mezi sebou. Výsledky zprůměrujeme přes všechny rámce. Poté vykreslíme frekvenční charakteristiku filtru jako výkonové spektrum.



Obrázek 10: Výkonové spektrum filtru

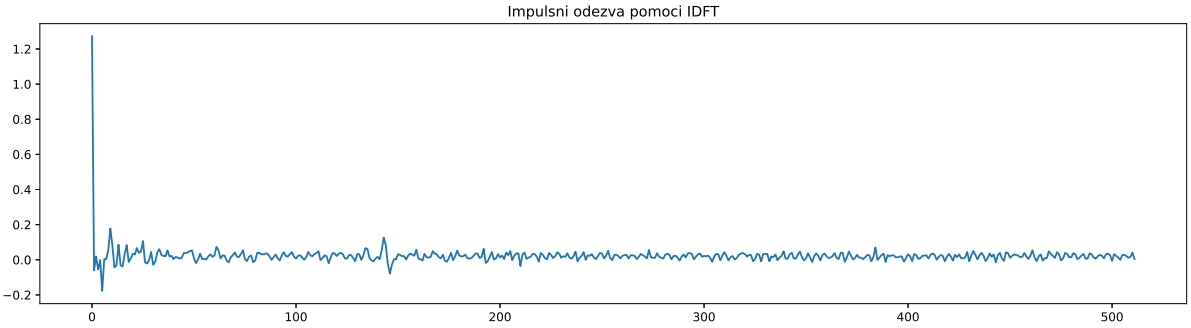
Dle frekvenční charakteristiky můžeme usoudit že po použití filtru bude výstupní signál silnější než vstupní[3]. Nejvíce při frekvencích kolem 2000, 5000 a 7500 Hz. Filtr je typu FIR. To znamená že jeho impulzní odezva je konečná. Tento filtr je vždy stabilní.

7. Úkol

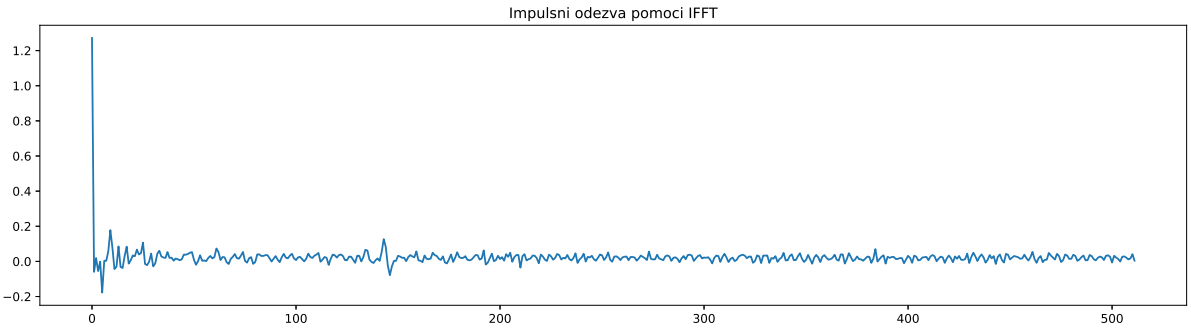
Impulzní odezva je získána díky inverzní diskrétní Fourierově transformaci. Implementace vlastní IDFT[6] a použití věstavěné funkce ifft knihovny numpy[4]:

```
    #IFFT
    impulzniodezva_ifft = np.fft.ifft(tone_freqchar,1024)[0:512]
    #IDFT
    impulzniodezva_idft = [0 for i in range(N)]
    for n in range(N):
        for k in range(N-1):
            impulzniodezva_idft[n] += tone_freqchar_padded[k]*np.e**(2*1j*np.pi*(k/N)*n)
        impulzniodezva_idft[n]=impulzniodezva_idft[n]/N
```

Porovnání vlastní implementace inverzní diskrétní Fourierovy Transformace s funkcí ifft z knihovny numpy:



Obrázek 11: Impulzní odezva pomocí inverzní diskrétní Fourierovy transformace

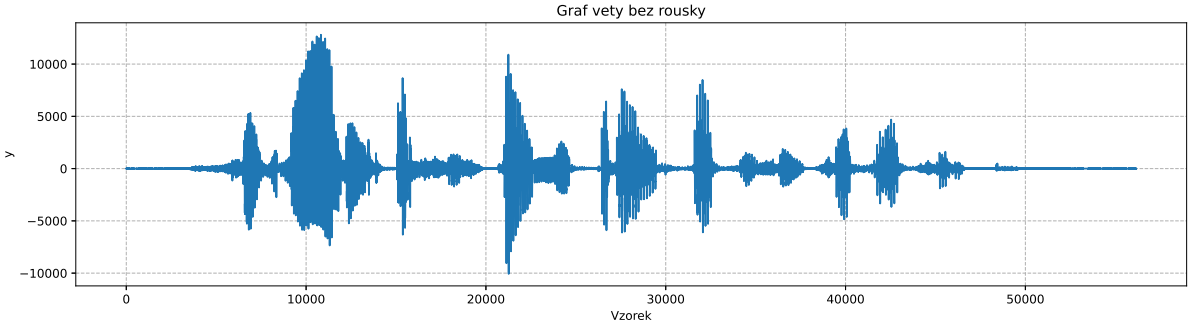


Obrázek 12: Impulzní odezva pomocí inverzní rychlé Fourierovy transformace

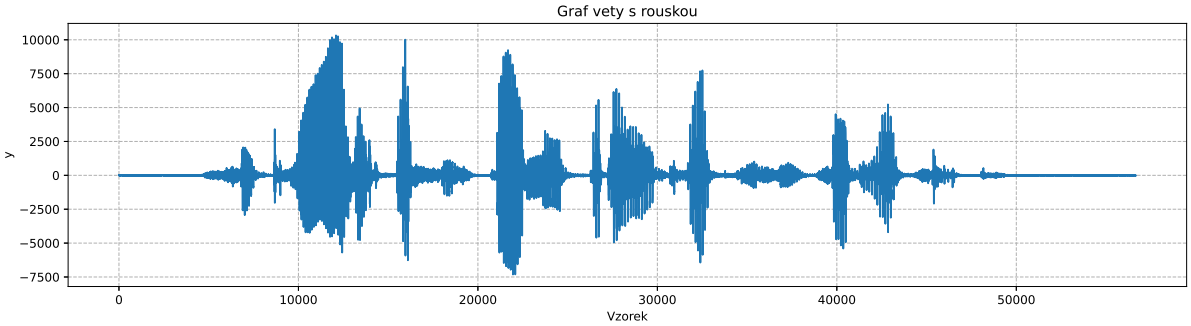
8. Úkol

Nyní si díky vypočítané impulzní odezvě můžeme vytvořit simulaci naší roušky na větě. Použijeme funkci lfilter[11].

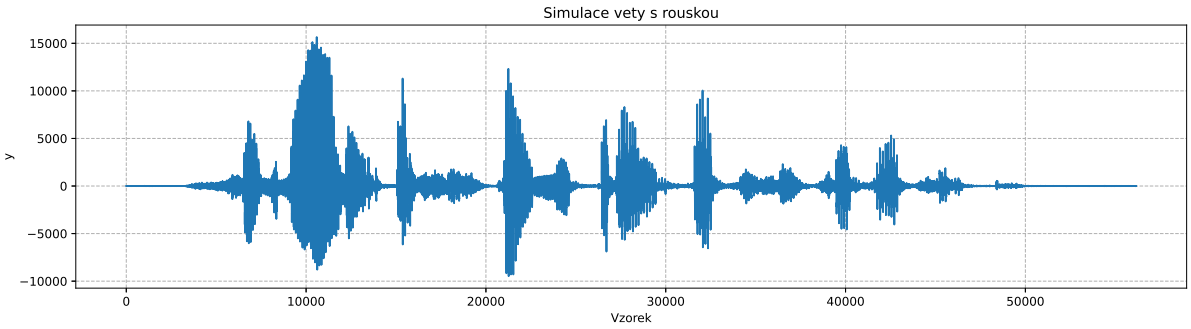
```
on_sentence_sim = [0 for i in range(len(off_sentence))]
on_sentence_sim = lfilter(impulzniodezva_ifft[0:512].real,[1],off_sentence)
```



Obrázek 13: Graf věty bez roušky



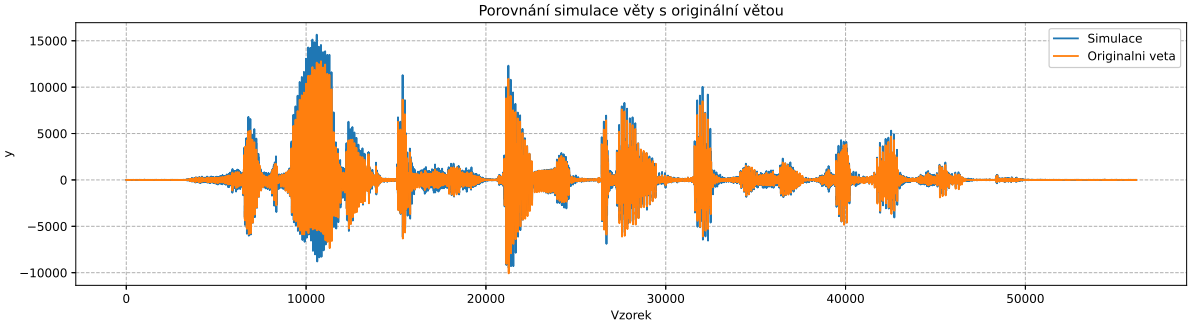
Obrázek 14: Graf věty s rouškou



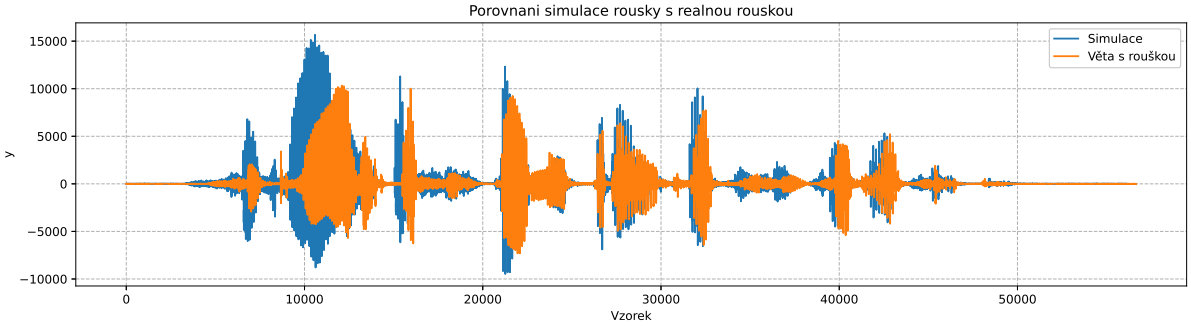
Obrázek 15: Graf věty se simulovanou rouškou

Při porovnání nahrávky bez roušky a simulace roušky si můžeme povšimnout že si signály jsou velice podobné. Simulace roušky originální signál na několika místech zesílilo. Zesílení si nejvíce můžeme všimnout kolem šesté desetiny sekundy. Zbytek signálu je zesílen velice mírně.

U porovnání nahrávky s reálnou a simulovanou rouškou lze vidět mnohem větší rozdíl. Jeden z důvodů je mírný časový posun některých úseků signálu, zejména v první polovině. Také simulovaná rouška zde funguje opačně než reálná. Simulace roušky zesiluje vstupní signál, kdežto reálná rouška ho zeslabí. Postupně se simulace roušky začíná více přibližovat reálné roušce. Přibližně v polovině třetí sekundy si jsou simulace a realita velice podobné.



Obrázek 16: Porovnání věty bez roušky a se simulovanou rouškou



Obrázek 17: Porovnání věty s rouškou a se simulovanou rouškou

9. Úkol - Závěr

Simulace nenaplnila mé očekávání, což byl slabší signál při simulaci. Použití filtru vstupní signál mírně zesílil, což neodpovídá situaci při použití reálné roušky. Odůvodnění toho může být špatná nahrávka tónu s rouškou, podle které se filtr vytvářel. Náhrávky mohly být ovlivněny náladou, únavou nebo jinou větnou melodií[7]. Nejvíce mě překvapil rozdíl časové náročnosti vlastní implementace DFT a FFT. Rychlá Fourierova transformace je několikanásobně rychlejší než diskrétní Fourierova transformace.

Zdroje

[1] Audacity Team: Systémy s diskrétním časem. Navštíveno 19. 12. 2020.
URL https://www.audacityteam.org/

[2] Bijota, B. J.: *Aplikace statistické analýzy řeči pacientů s Parkinsonovou nemocí.* Diplomová práce, Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, Brno, 2016.

[3] Máša, P.: Frekvenční charakteristika. online, navštíveno 23. 12. 2020.
URL http://skola.hellebrand.cz/text1011/au/soustavy-freqchar.pdf

[4] The SciPy community: Knihovna NumPy. Navštíveno 21. 12. 2020.
URL https://numpy.org/doc/stable/index.html

[5] The SciPy community: Knihovna SciPy. Navštíveno 20. 12. 2020.
URL https://docs.scipy.org/doc/scipy/reference/index.html

[6] Černocký, H.: Spektrální analýza a diskrétní Fourierova transformace. online, navštíveno 21. 12. 2020.
URL http://www.fit.vutbr.cz/study/courses/ISS/public/NEW_PRED/02_spectrum/spectrum.pdf

[7] Černocký, H.: Zpracování řečových signálů — studijní opora. online, navštíveno 21. 12. 2020.
URL https://www.fit.vutbr.cz/study/courses/ZRE/public/opora/zre_opora.pdf

[8] Černocký, J.: Systémy s diskrétním časem. online, navštíveno 22. 12. 2020.
URL http://www.fit.vutbr.cz/study/courses/ISS/public/pred/11_disk_syst/disk_syst.pdf

[9] Černocký, J.: Systémy s diskrétním časem. online, navštíveno 22. 12. 2020.
URL http://www.fit.vutbr.cz/study/courses/ISS/public/pred/11_disk_syst/disk_syst.pdf

[10] Žmolíková, I. K.: Obecne python tipy. online, navštíveno 20. 12. 2020.
URL https://nbviewer.jupyter.org/github/zmolikova/ISS_project_study_phase/blob/master/Obecne_Python_tipy.ipynb

[11] Žmolíková, I. K.: Zvuk, spektra, filtrace. online, navštíveno 23. 12. 2020.
URL https://nbviewer.jupyter.org/github/zmolikova/ISS_project_study_phase/blob/master/Zvuk_spektra_filtrace.ipynb#Filtrace