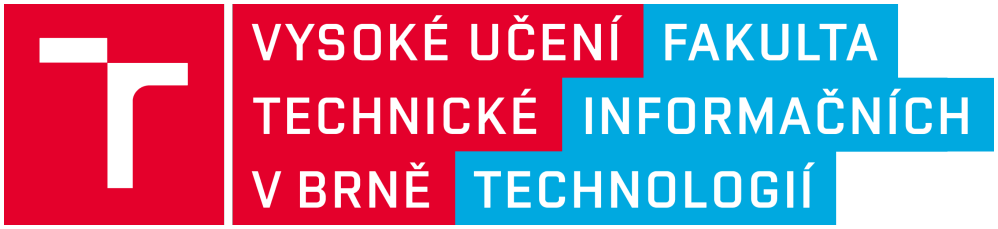


Signály a systémy

Projekt - analýza vlivu roušky na řeč

Tereza Burianová (xburia28)

4. ledna 2021

Detaily implementace

Projekt je napsán v jazyce Python v Jupyter Notebook, který je součástí distribuce Anaconda. Dále bylo třeba pomocí Anaconda Prompt a příkazu *"pip install soundfile"* doinstalovat modul SoundFile.

1. úloha - vytvoření nahrávek tónů

Audionahrávky byly vytvořeny přes mobilní telefon v aplikaci, která umožňuje nastavení formátu souboru a vzorkovací frekvence. Přímo v kódu jsem pomocí "list slicing"vybrala z každé nahrávky 1 sekundu, kterou jsem následně posouvala podle podobnosti základní frekvence ze 4. úlohy.

Název souboru	Délka ve vzorcích	Délka v sekundách
maskoff_tone.wav	76800	00:00:04.80
maskon_tone.wav	78400	00:00:04.90

2. úloha - vytvoření nahrávek vět

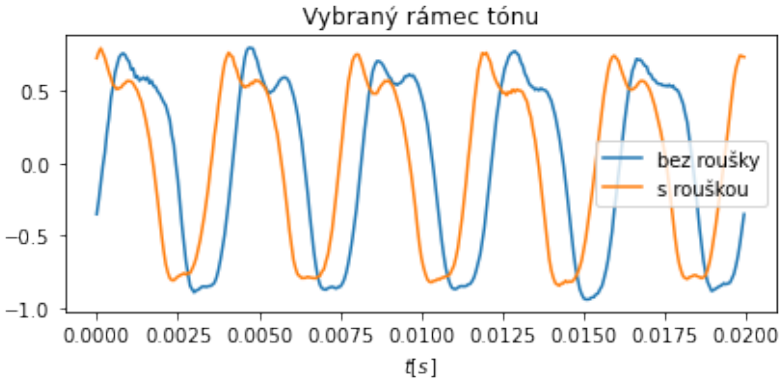
Zvolila jsem větu "Ahoj, jak se máš?", která byla dostatečně krátká k tomu, aby se nahrávky co nejvíce podobaly. Přímo v kódu jsem upravila oba signály tak, aby začínaly a končily stejně.

Název souboru	Délka ve vzorcích	Délka v sekundách
maskoff_sentence.wav	48000	00:00:03.00
maskon_sentence.wav	54400	00:00:03.40

3. úloha - vytvoření úseků a rámců

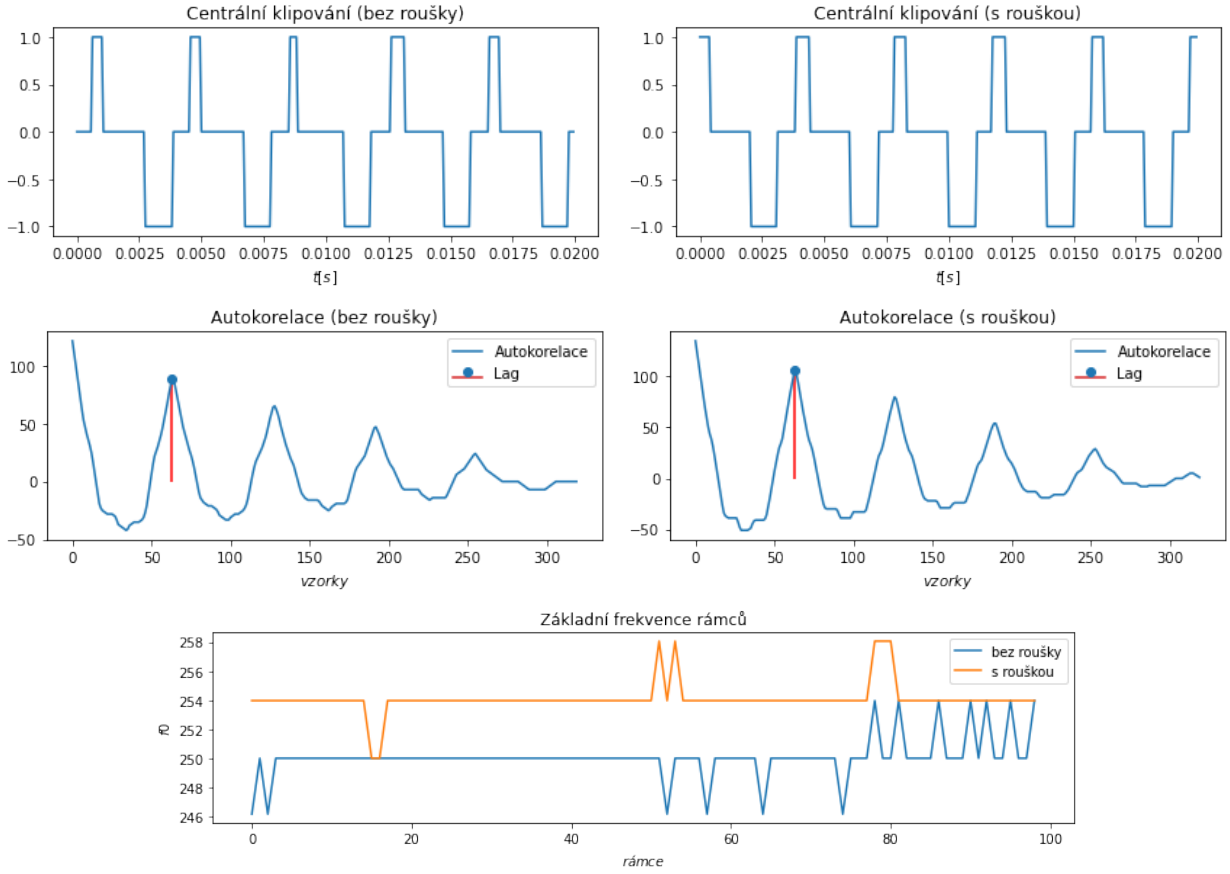
Oba signály tónu byly ustředněny a normalizovány pomocí funkcí numpy.mean(x) a numpy.abs(x).max(). Následně jsem celou sekundu rozdělila na rámce o délce 20 ms, které se do poloviny překrývají. Počet vzorků v rámci jsem vypočítala podle následujícího vztahu:

$$ramec = \frac{F_s}{\frac{delka\_nahravky[ms]}{delka\_ramce[ms]}} = \frac{16000}{\frac{1000}{20}} = 320$$



4. úloha - autokorelace a základní frekvence nahrávek

Nejprve bylo podle algoritmu popsaného v zadání aplikováno na všechny rámce centrální klipování, následně mohla být na takto upravený rámec aplikována autokorelace. Z takto upraveného signálu je možné získat hodnotu lag-u a základní frekvenci rámců obou nahrávek.



Střední hodnota a rozptyl se liší následovně:

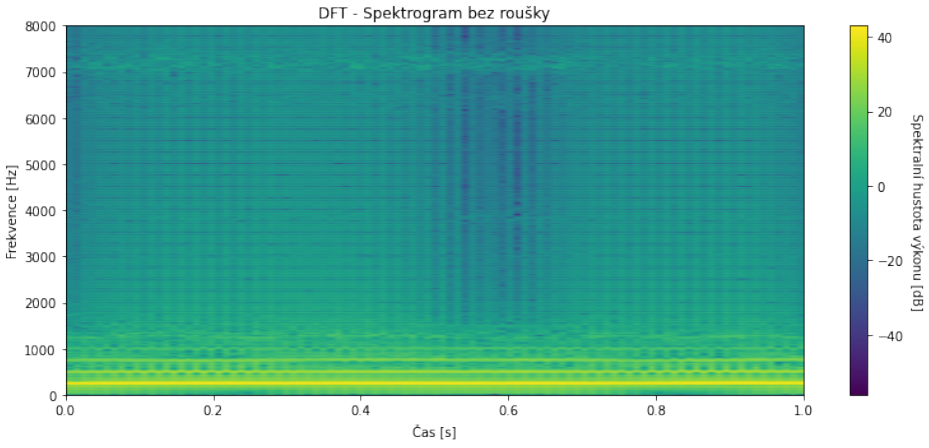
	Střední hodnota základní frekvence	**Rozptyl základní frekvence**	**Střední hodnota lag-u**
Bez roušky	250,05	2,01	63,99
S rouškou	254,09	1,15	62,97

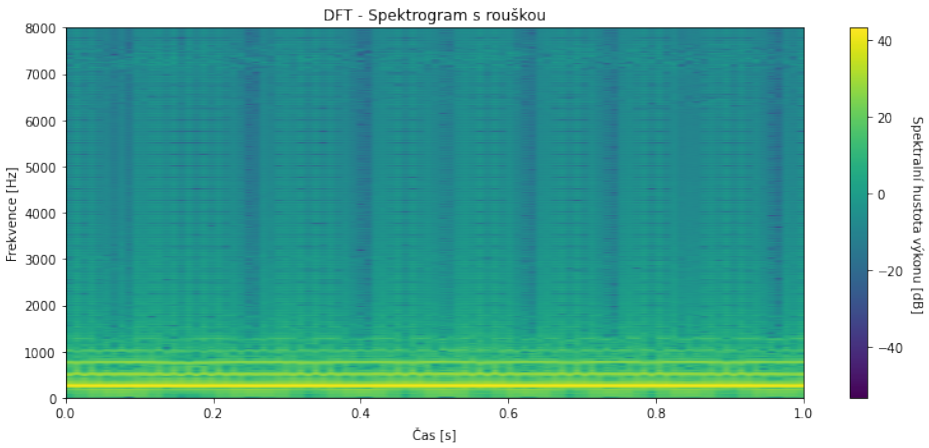
Na doporučení vyučujícího jsem kromě hodnot pro základní frekvenci vypočítala i střední hodnotu lag-u, která mi pomohla se rozhodnout, zda je podobnost dostačující. Základní frekvence se vypočítá podle vztahu $f_0 = \frac{F_s}{lag}$, takže by k menším rozdílům ve grafu při stejných hodnotách lag-ů napomohlo navýšení vzorkovací frekvence na více než aktuálních 16 000 (tedy například 48 000).

5. úloha - diskrétní Fourierova transformace

DFT jsem vypočítala podle vzorce $X_k = \sum_{n=0}^{N-1} x_n * e^{-i2\pi kn/N}$. Hodnoty jsem následně upravila podle vzorce $P[k] = 10 * \log_{10} |X[k]|^2$ uvedeného v zadání a zobrazila jsem je jako spektrogram. Vzorec jsem implementovala následujícím způsobem (příklad uveden pro hodnoty bez roušky):

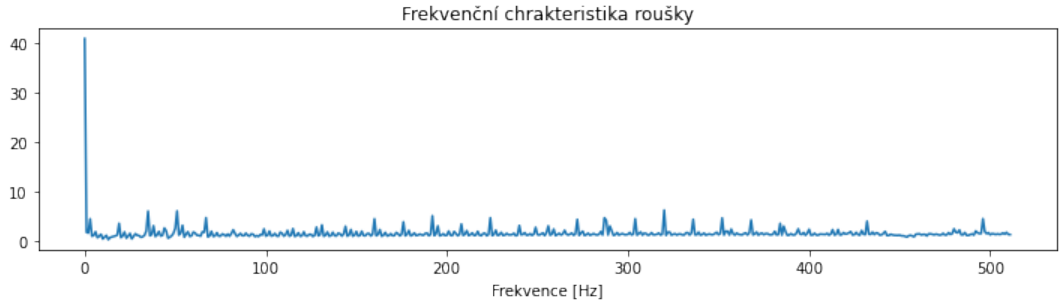
```
DFT = []
DFT_log = []
for frame in range(99): # jednotlivé rámce
    frames_toneoff_3[frame] = np.pad(frames_toneoff_3[frame],
    (0, 1024 - len(frames_toneoff_3[frame])), 'constant') # zero padding
    N = len(frames_toneoff_3[frame])
    k = 0
    Xlist = []
    for k in range(N): # pro všechna X[k] v rámci
        X = 0 # X[k] = X
        for n in range(N): # suma
            M = np.exp(-2j * np.pi * k * n / N)
            X += frames_toneoff_3[frame][n] * M
        Xlist.append(X)
    DFT.append(Xlist)
    DFT_log.append(Xlist)
    Gval = 10 * np.log10(np.abs(DFT[frame])**2) # hodnoty do spektrogramu
    DFT_log[frame] = Gval
    DFT[frame] = DFT[frame][:512]
    DFT_log[frame] = DFT_log[frame][:512]
```





6. úloha - frekvenční charakteristika filtru

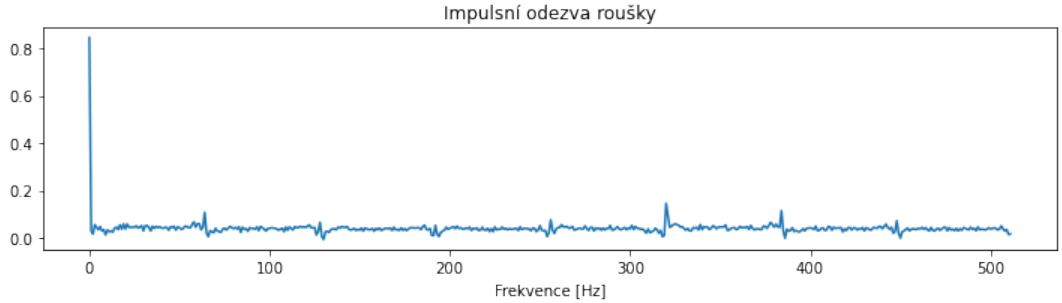
Pro výpočet frekvenční charakteristiky roušky může být použit vzorec $H(e^{j\omega}) = \sum_{k=0}^{\infty} h[k]e^{-j\omega k}$, v případě tohoto projektu ale může být využita předchozí implementace DFT. Výpočet je proveden jako $\frac{DFT(mask\_on)}{DFT(mask\_off)}$ a následně je jeho absolutní hodnota zprůměrována přes všechny rámce. Vytvořený filtr bude po úpravě do časové oblasti již možno využít k simulaci roušky na nahrávce bez roušky.



7. úloha - impulsní odezva

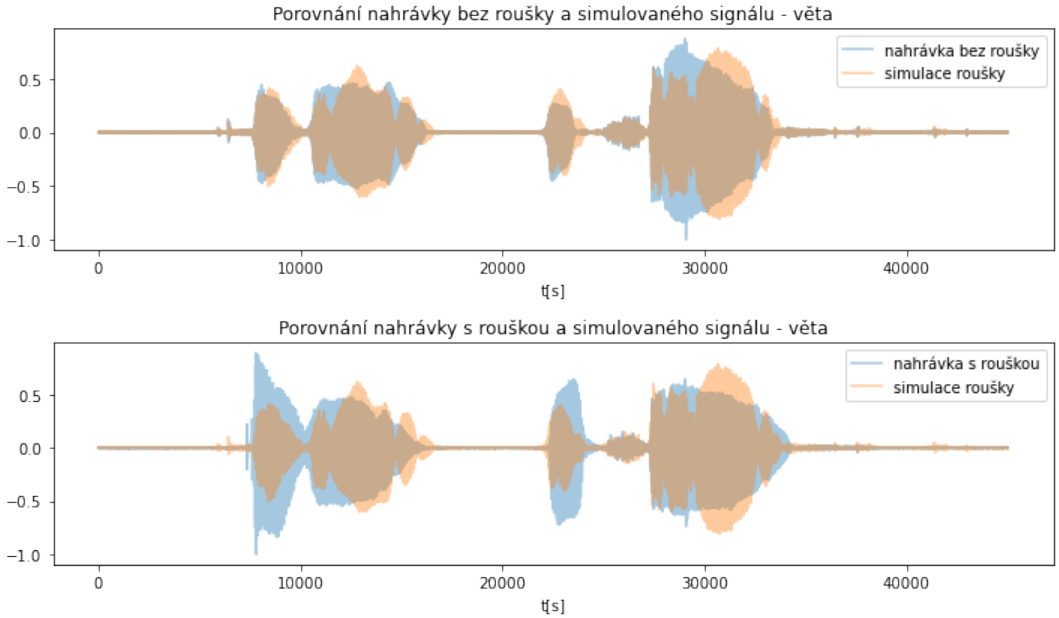
Převod frekvenční charakteristiky na impulsní odezvu může být proveden pomocí inverzní diskrétní Fourierovy transformace. Algoritmus jsem implementovala podle vzorce $x_n = \frac{1}{N} * \sum_{k=0}^{N-1} N_k * e^{i2\pi kn/N}$:

```
IDFT = []
char = np.pad(char, (0, 1024 - len(char)), 'constant') # zero padding
N = len(char)
for k in range(N): # pro všechna x
    X = 0 # X[k] = X
    for n in range(N): # suma
        M = np.exp(2j * np.pi * k * n / N)
        X += char[n] * M
    X = (1/N)*X
    IDFT.append(X)
IDFT = IDFT[:512]
```



8. úloha - aplikace filtru

Filtr byl na nahrávku věty bez roušky aplikován pomocí funkce scipy.signal.lfilter a výsledky jsou následující:



Simulovaný signál je podobný nahrávce bez roušky, ale má větší výkyvy. Nahrávka s rouškou má, oproti té bez roušky i simulaci, v některých částech vyšší hodnoty. Hodnoty se ale podle mého názoru podobají dostatečně pro získání výsledků.

9. úloha - závěr

I přesto, že se simulovaný signál zcela nepodobá nahrávce s rouškou, je podobnost výsledných audio souborů dostačující. Simulace zní oproti normalizované nahrávce bez roušky „tlumeněji“, což se dá očekávat. Rozdíl v signálech je zřejmě způsoben kvalitou nahrávek, které byly nahrávány mobilním telefonem a udržení stejné hlasitosti, tónu a také vlivu roušky u obou nahrávek tónu i věty na řeč není jednoduché. Kvalitnější nahrávky by jistě daly lepší výsledky.