ISS Projekt

Ondřej Sabela (xsabel03)

2020/21

Projekt je implementován v jazyce Python (v Jupyter Notebooku) na systému Windows 10.

Ondřej Sabela (xsabel03)

1. – 2.

Pro nahrání zvuku jsem použil notebook a aplikaci Audacity.

Název souboru	Počet vzorků	Doba trvání
maskoff_tone	51877	03.24
maskon_tone	67420	04.21
maskoff_sentence	48736	03.05
maskon_sentence	56420	03.53

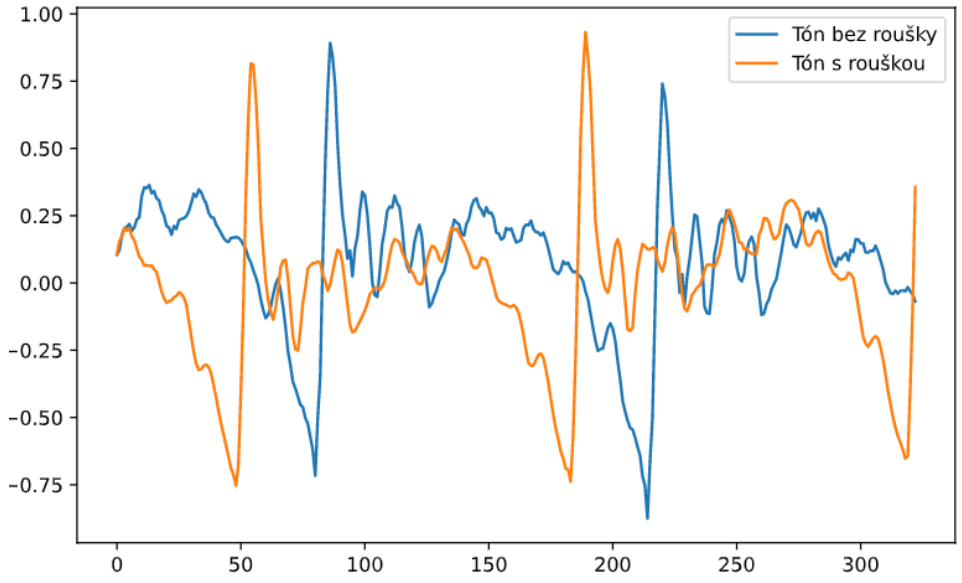
3. Rámce

Zvukové soubory jsou čteny funkcí wavfile.read.

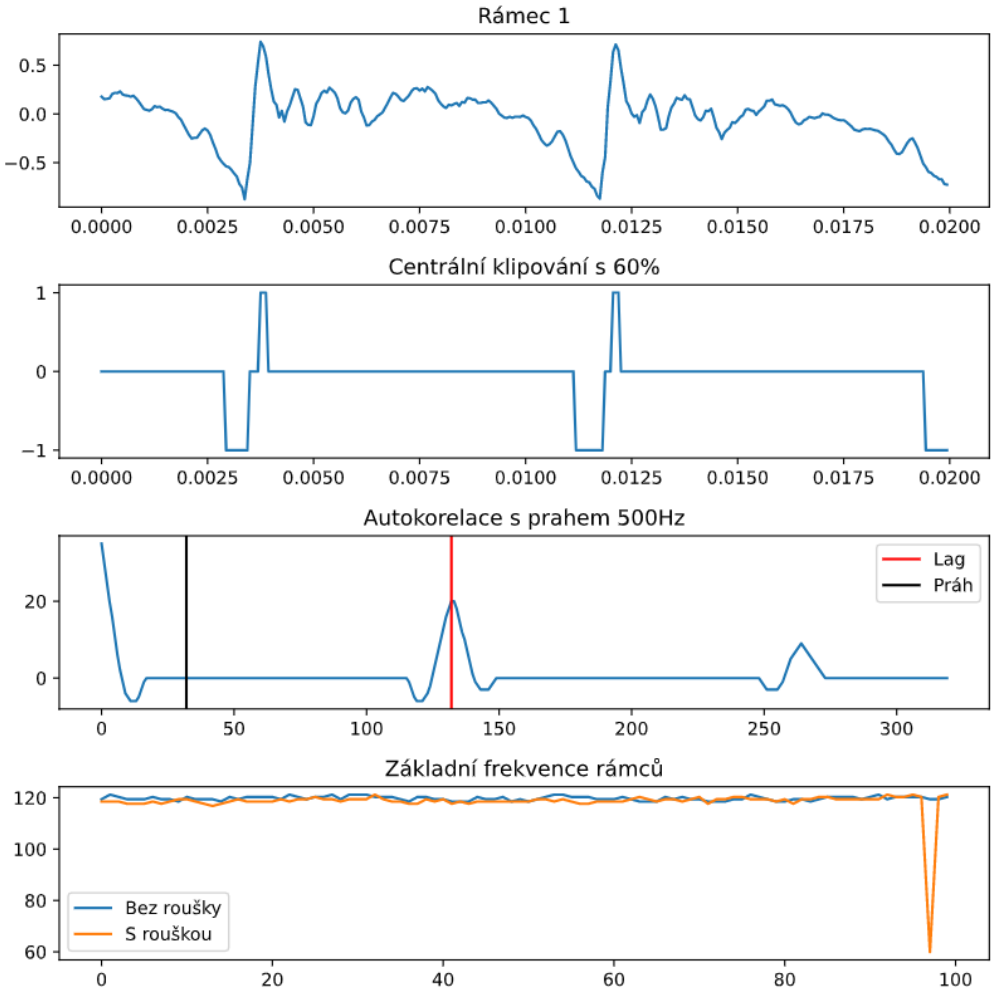
Vzorec pro výpočet velikosti rámce: (Fs/N)*2
Fs (vzorkovací frekvence) = 1600
N (počet rámců) = 100

Dosazení: (16000/100)*2 = 320

1sekundové sekce jsem z nahrávek vybral ručně.



4. Výška tónu



a) Autokorelaci jsem implementoval podle vzorce pro korelační koeficienty ze studijní opory k předmětu ZRE, sekce 7.2.1[1]

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

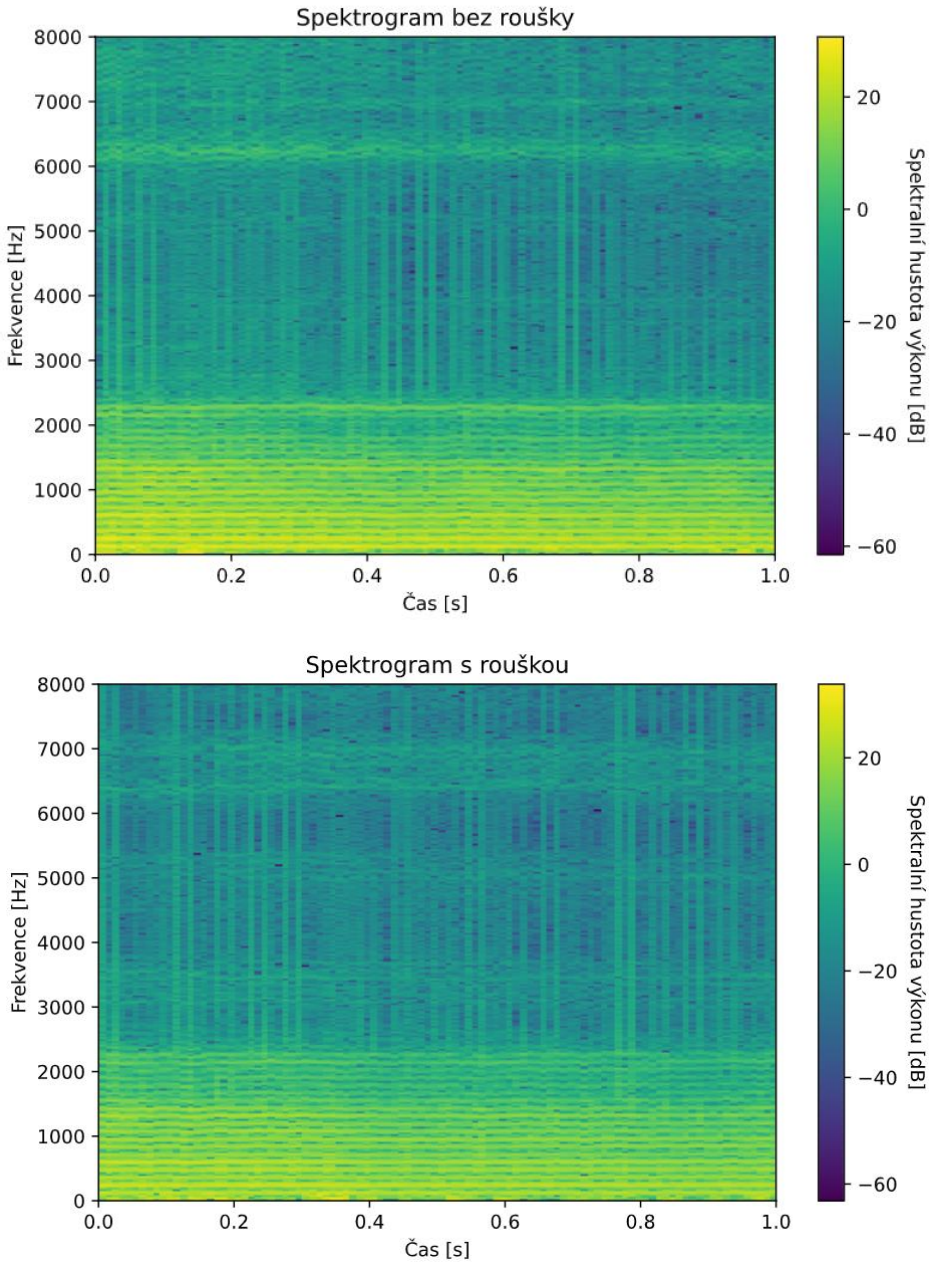
b)

Nahrávka	Střední hodnota zákl. frekv.	Rozptyl
Bez rouškы	119.7567354236639	0.6253005171279074
S rouškou	118.41547567653107	35.37877442337423

c) Velikost minimální změny ve frekvenci by se dala zmenšit, kdybychom použili delší rámce, nebo signál (třeba jen simulovaně) nadvzrokovali.

[1] https://www.fit.vutbr.cz/study/courses/ZRE/public/opora/zre_opora.pdf

5. DFT



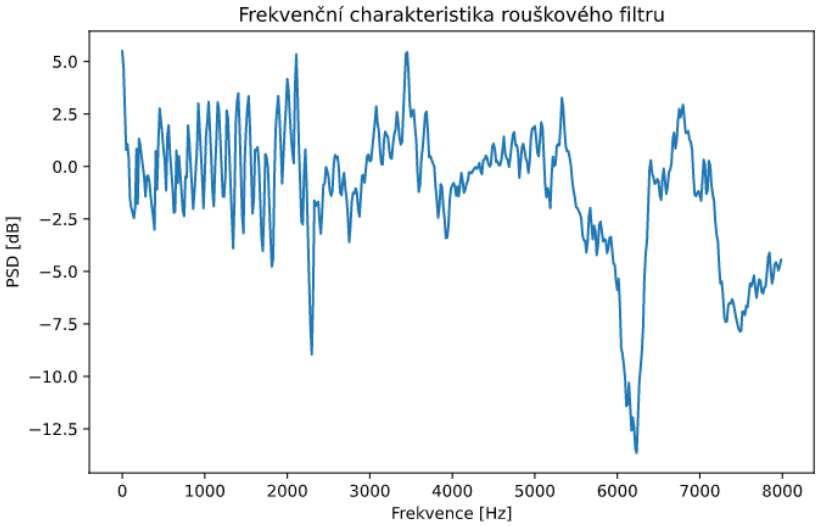
Spektrogramy jsou velmi podobné těm v zadání. Pro jejich vykreslení jsem použil funkci *imshow* z knihovny *numpy*.

Implementace DFT:

```
def dft(source, N):
    result = []
    padded = np.pad(source,(0,N-len(source)))
    for k in range(N):
        sum = np.complex(0,0)
        for n in range(N):
            exponent = -2j * np.pi * n * (k/float(N))
            sum += padded[n] * np.exp(exponent)
        result.append(20*np.log10(np.abs(sum)))

    return result
```

6. Frekvenční charakteristika

$$H(e^{j\omega}) = \frac{B(e^{j\omega})}{A(e^{j\omega})}$$

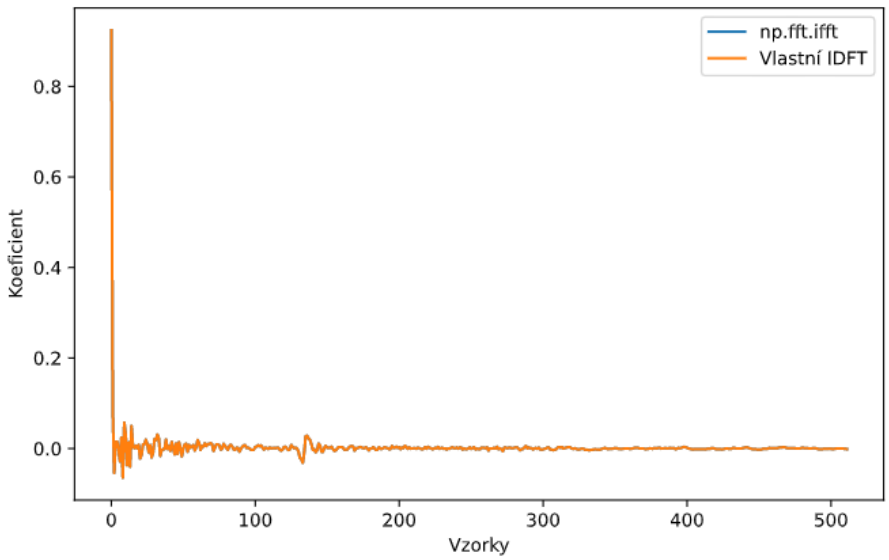


Komentář: Filtr je typu FIR. Některé frekvence dokonce zesiluje, což odporuje logice roušky, která by měla jen zeslabovat hlas. Zesílení je pravděpodobně způsobeno nestejnou hlasitostí mého hlasu v nahrávkách s rouškou a bez roušky.

7. Filtrace

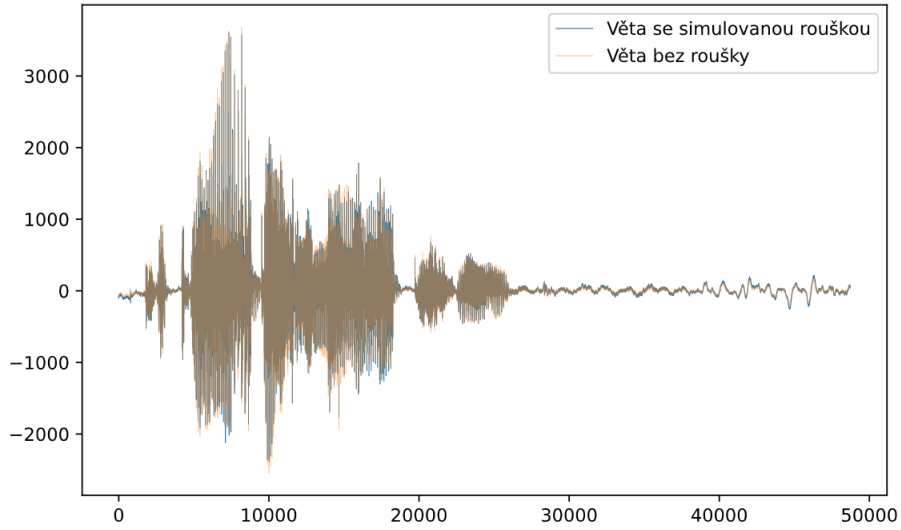
Implementace IDFT:

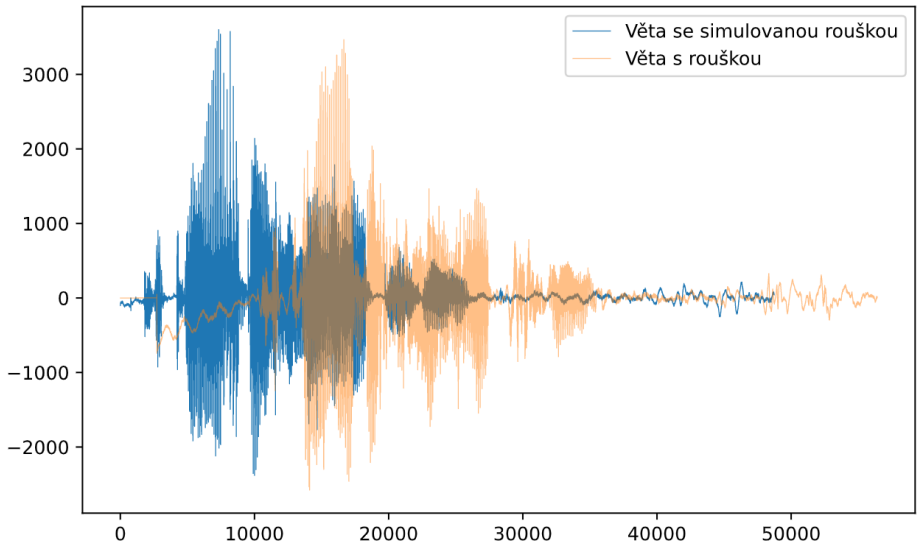
```
def idft(source, N):
    result = []
    for k in range(N):
        sum = np.complex(0,0)
        for n in range(N):
            exponent = 2j * np.pi * n * (k/float(N))
            sum += (source[n] * np.exp(exponent)) / N
        result.append(sum)

    return result
```



8. Simulace roušky

K filtraci jsem použil funkci *signal.lfilter* z knihovny *scipy*.



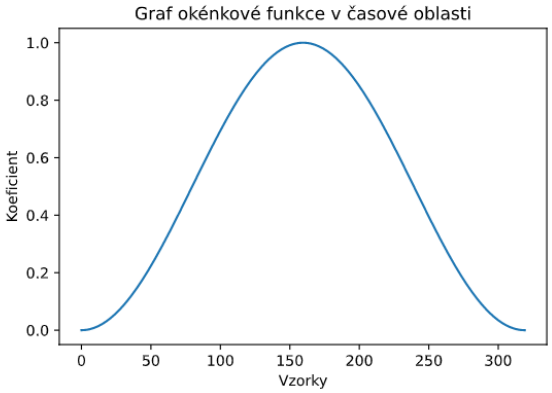


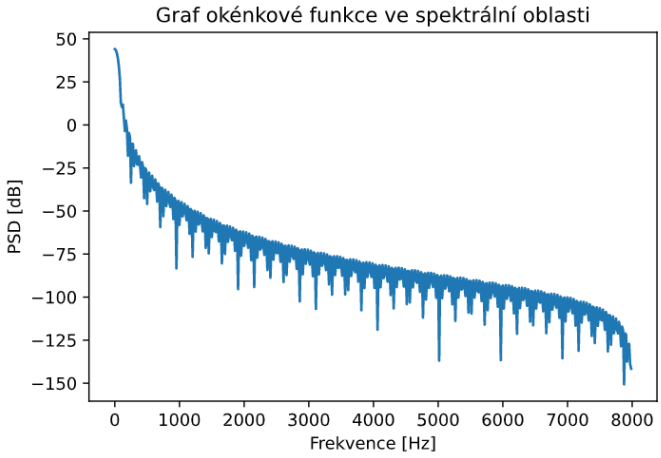
Signály jsou si velmi podobné už jen proto, že původní signály si byly podobné a rouška, kterou jsem použil, neměla moc silný efekt.

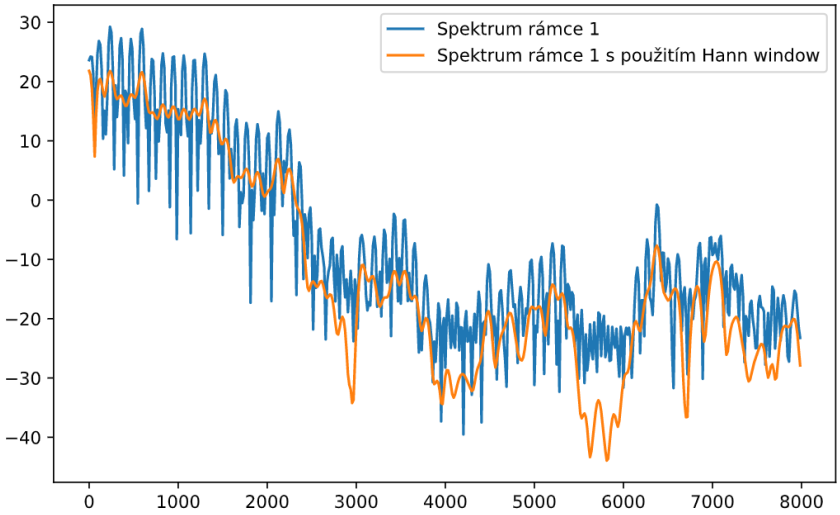
Liší se v místech, kde se mi nepodařilo udělat tón/větu tak přesné. Nejvíce podobné jsou v místech, kde je tón nejstabilnější. Nejvíce patrné je zmenšení hlasitosti celého signálu.

11. Okénková funkce

Vybral jsem okénkovou funkci Hann window.

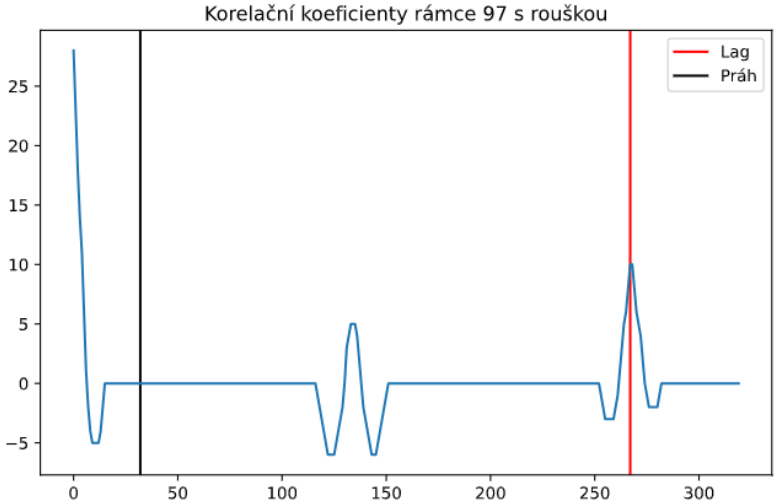






Okénková funkce je užitečná, protože Fourierova transformace předpokládá periodický signál a na začátku a konci rámce se proto v časové oblasti vytvoří skok. Okénková funkce tento skok eliminuje, a proto je výsledné spektrum vyhlazenější.

12. Detekce n-násobného lagu



Chyba byla odstraněna použitím nelineárního mediánového "filtru". (použil jsem k=2)

$L(i) = \text{med}\,[L(i-k), L(i-k+1), \ldots, L(i), \ldots, L(i+k)]$

9. Závěr

Mám dojem, že moje řešení je naprogramováno správně. Filtr každopádně působí naprosto minimální změnu, za což dávám vinu svým příliš podobným nahrávkám s rouškou a bez roušky.