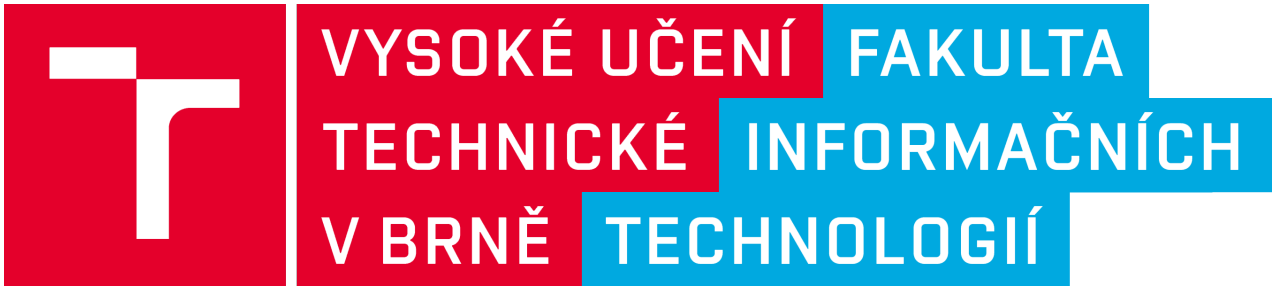
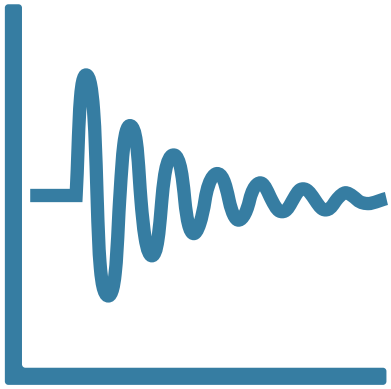


PROTOKOL

Predmet: **ISS** Signály a systémy



Autor: Samuel Líška

Login: xliska20

15. decembra 2020

Obsah

PRVOTNÉ ODHADY

1. ÚLOHA - TESTOVACIE NAHRÁVKY(TÓNY)

Súbory som nahral a nastavil podľa zadania (mono, 16kHz, 16 bit).

Súbor	Počet vzorkov	Dĺžka(s)
maskoff_tone.wav	43654	2.728375
maskon_tone.wav	55989	3.4993125

2. ÚLOHA - TESTOVACIE NAHRÁVKY(VETY)

Súbory som nahral a nastavil podľa zadania (mono, 16kHz, 16 bit).

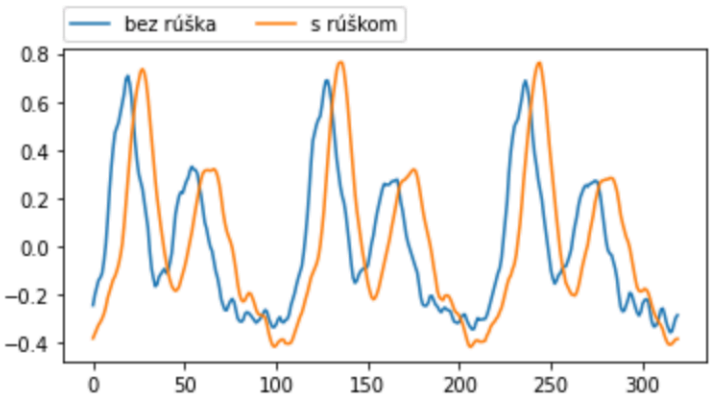
Súbor	Počet vzorkov	Dĺžka(s)
maskoff_sentence.wav	33437	2.0898125
maskon_sentence.wav	33808	2.1130

3. ÚLOHA - ÚPRAVA NAHRÁVKY

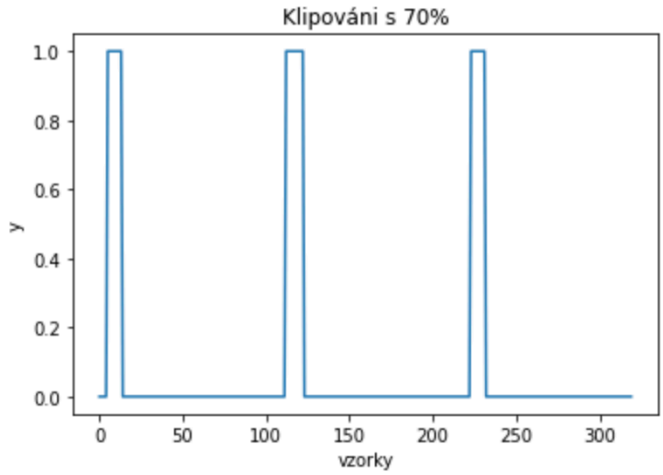
fs = vzorkovacia frekvencia, **fwidth** = dĺžka rámca, **F =** počet vzorkov v rámci. Následne som pomocou rovnice:

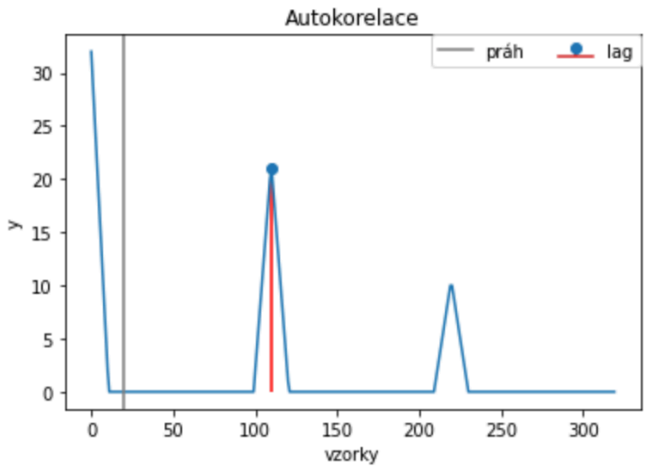
$$F = fs * fwidth$$

Vypočítal počet vzorkov v rámci a signál na ne rozdelil (99 rámcov)



4. ÚLOHA - KLIPOVANIE A AUTOKORELÁCIA





Maskoff stredná hodnota = 148.148148 Maskoff rozptyl = 1.4810046

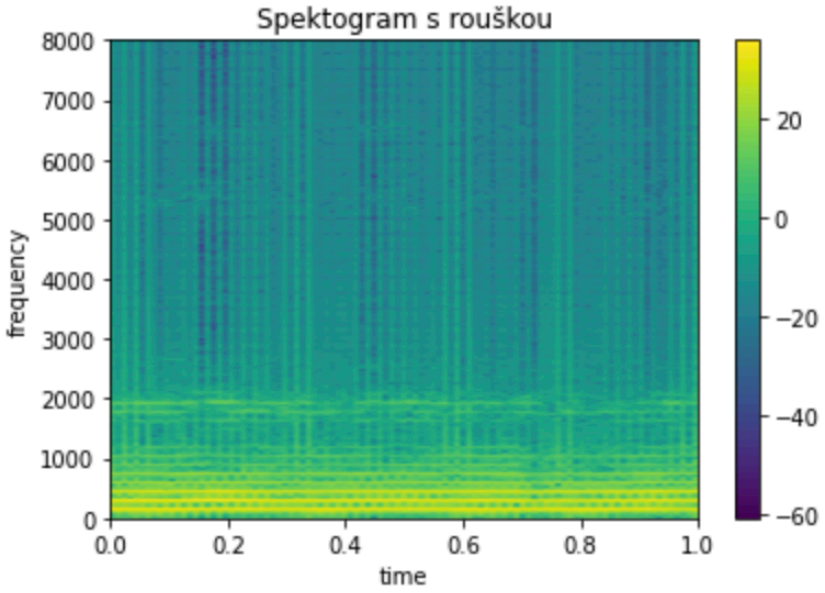
Maskon stredná hodnota = 148.148148 Maskon rozptyl = 1.0449961

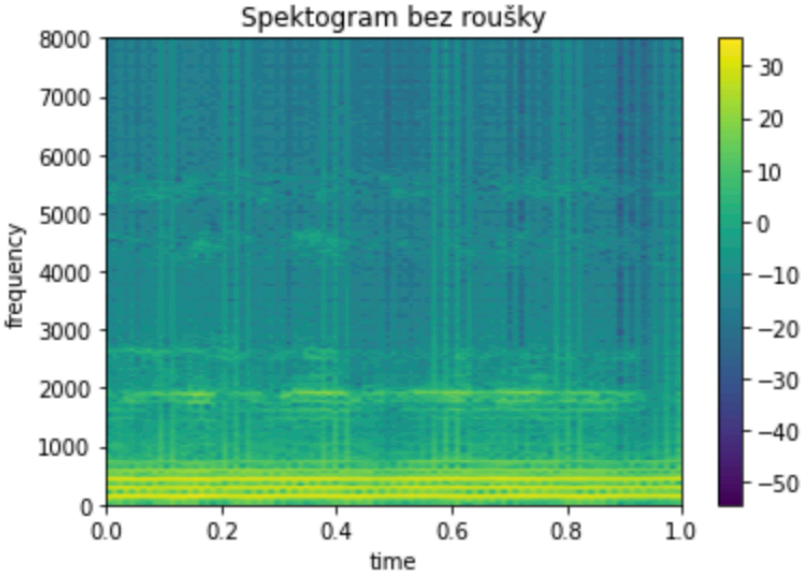
Pomocou jednoduchej (mnou vytvorenej) funkcie **centerClipping** fungujúcej presne podľa zadania, som implementoval klipovanie, a následne cez funkciu **autoCorrelation** autokoreláciu rámcov v ktorej som aj ukladal hodnoty a indexy lagov, z ktorých som následne vyrátal **základnú frekvenciu f0**.

5. ÚLOHA - DFT + SPEKTOGRAMY

Algoritmus pre výpočet DFT: Výsledok je totožný s knihovňovou funkciou np.fft.fft, avšak môj algoritmus je značne pomalší. **Hodnoty z DFT**(v spektograme) sú upravené podl'a vzorca v zadaní.

```python
def dft(signal):
    result = []
    for frame in signal:
        t = []
        N = len(frame)
        for i in range(N):
            a = 0
            for n in range(N):
                a += frame[n]*np.exp(-2j*np.pi*i*n*(1/N))
            t.append(a)
        result.append(t)
    return result
```



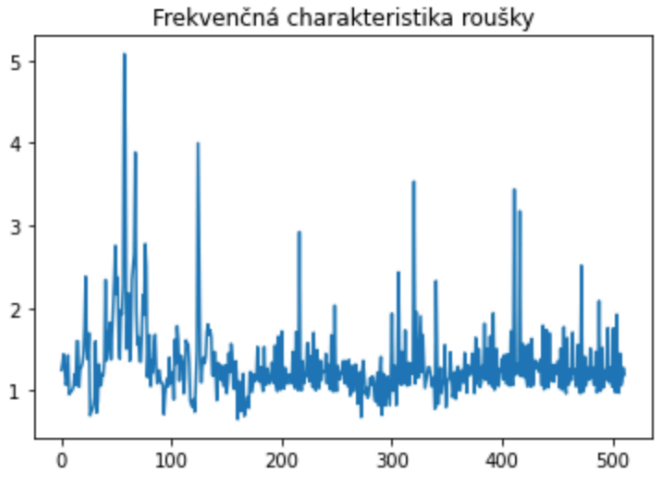


6. ÚLOHA - FREKVENČNÁ CHARAKTERISTIKA

Vzťah pre výpočet frekvenčnej charakteristiky filtra:

$$H(e^{j\omega}) = \frac{dft(maskon)}{dft(maskoff)}$$

Následne som rámce zpriemeroval aby vyšla jedna charakteristika.



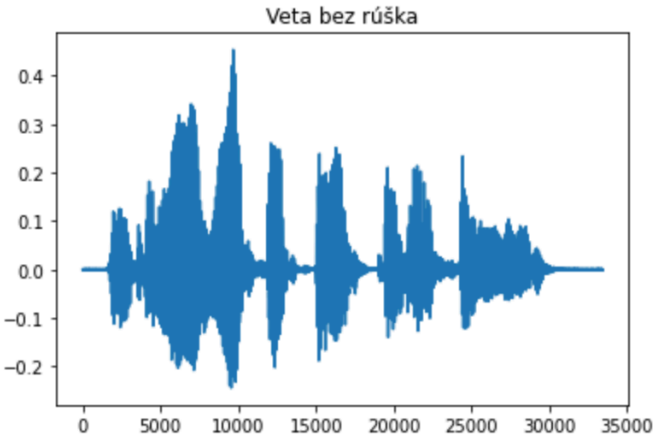
7. ÚLOHA - IDFT + IMPULZÍVNA ODOZVA

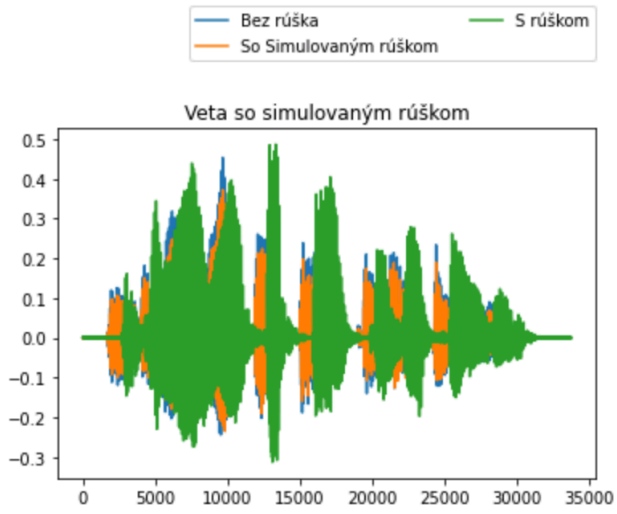
Algoritmus IDFT: Vlastná implementácia má rovnaký výsledok ako ifft z knihovne numpy, lenže jej výpočet je značne pomalší.

```
def idft(signal):
    result = []
    for frame in signal:
        x = []
        N = len(frame)
        for n in range(N):
            a = 0
            for k in range(N):
                a += frame[k]*np.exp(2j*np.pi*k*n*(1/N))
            a /= N
            x.append(a)
        result.append(x)
    return result
```

Impulzívna odozva:

8. ÚLOHA - APLIKOVANIE A POROVNANIE FILTRA





Veta bez rúška a s rúškom sa už na pohľad veľmi podobajú, avšak simulácia mierne znížila rozsahy. Tvar je približne rovnaký ale zmenšený, a niektoré hrany sú kostrbatejšie, iné hladšie. Zvukový rozdiel je minimálny a nepribližuje sa efektu mnou použitého reálného rúška.

9. ÚLOHA - ZÁVER A ZHODNOTENIE

Záver: Moja implementácia simulovaného rúška určite signál ovplyvňuje, bohužial' keď si simulovanú vetu porovnám s vetou kde som použil reálne rúško je to neporovnatel'ný rozdiel.

Zvuk vychádzajúci zo simulovaného rúška by sa dal možno porovnať s len vel'mi tenkým chirurgickým rúškom ktoré tol'ko neovplyvňuje hlas. Ja som ale použil vel'mi výrazné plátené ktoré navyše dosť pril'nulo, takže skreslenie bolo značné.

DOPLŇUJÚCE ÚLOHY

10. ÚLOHA - OVERLAP-ADD

Overlap-add metóda:

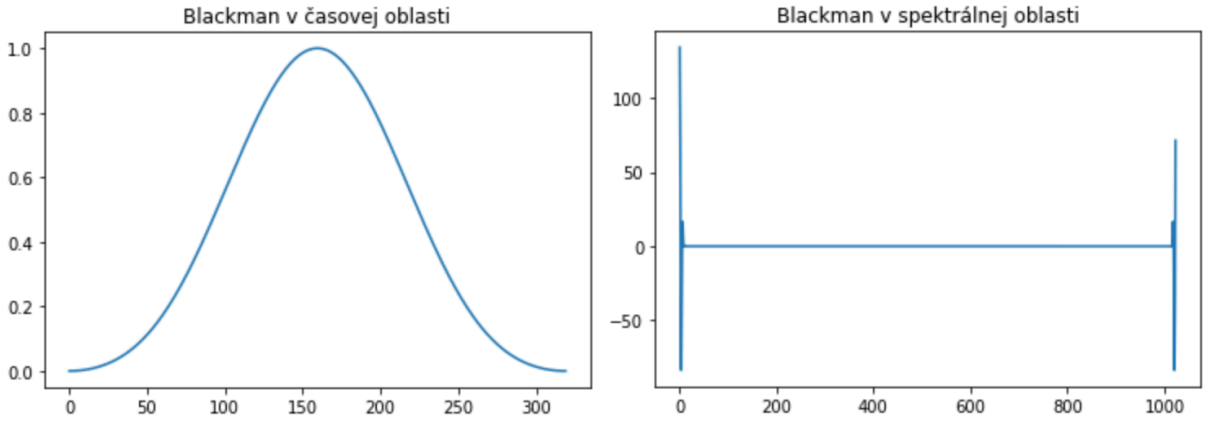
```
def overlap_add(flt, sig):
    L_I = flt.shape[0]
    #mocnina 2 väčšia ako. 2*L_I(zo stackoverflow)
    L_F = 2<<(L_I-1).bit_length()
    L_S = L_F - L_I + 1
    L_sig = sig.shape[0]
    offsets = range(0, L_sig, L_S)

    # komplexne a reálne čísla
    if np.iscomplexobj(flt) or np.iscomplexobj(sig):
        fft_func = np.fft.fft
        ifft_func = np.fft.ifft
        res = np.zeros(L_sig+L_F, dtype=np.complex128)
    else:
        fft_func = np.fft.rfft
        ifft_func = np.fft.irfft
        res = np.zeros(L_sig+L_F)

    FDir = fft_func(flt, n=L_F)

    # overlap and add
    for n in offsets:
        res[n:n+L_F] += ifft_func(fft_func(sig[n:n+L_S], n=L_F)*FDir)
    return res[:L_sig]
```

V adresári ./audio sú 2 súbory **sim_maskon_sentence_overlap_add.wav** a **sim_maskon_tone_overlap_add.wav.** V nahrávke okrem veľkého hluku väčší rozdiel nepočujem.

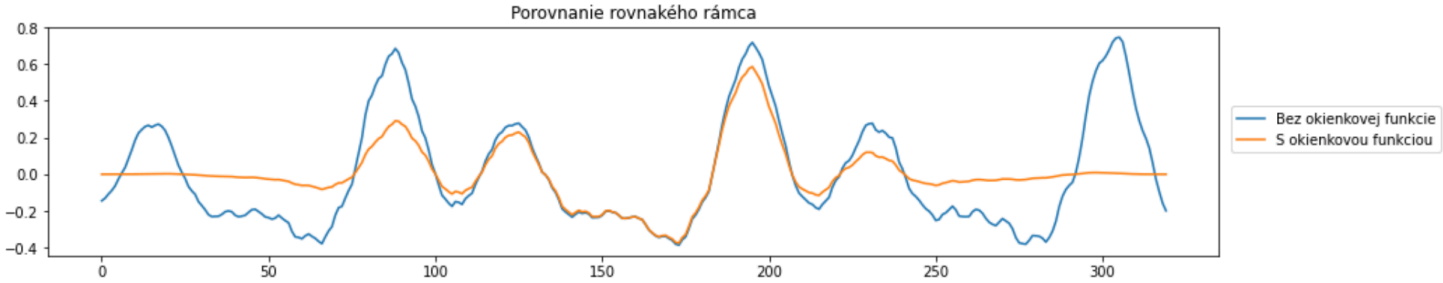
11. ÚLOHA - VYUŽITIE OKIENKOVEJ FUNKCIE

a) Väčšina odkazov na **Blackmanovo okno** pochádza z literatúry o spracovaní signálu, kde sa používa ako jedna z mnohých okienkových funkcií na vyhladenie hodnôt. Je tiež známa ako "apodization" (čo znamená „odstránenie chodidla“, t. j. Vyhladenie nesúvislostí na začiatku a na konci vzorkovaného signálu) alebo zužujúca sa funkcia.

b)



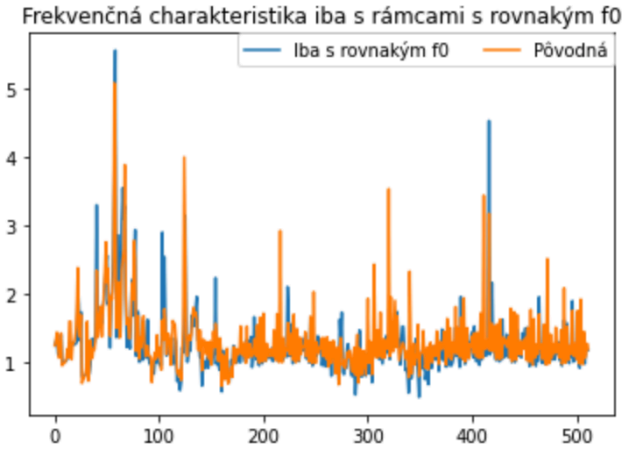
c) Porovnanie rámca



Použitie Blackmanovej funkcie je vhodé z toho dôvodu, že vylepší výkon(optimalizuje, urýchly) FFT.

13. ÚLOHA - FREK. CHAR. Z RÁMCOV S ROVNAKÝM F0

Frekvenčná charakteristika tvorená iba z rámcov ktoré majú skutočne rovnaký f0 má značne väčšie výkyvy. Tvar je rovnaký len niektoré “kopce” sú značne vyskočené(napr **frame 102**). Čiže filter by mal byť teoreticky mierne efektívnejší.



ZDROJE

https://numpy.org/doc/stable/

https://www.scipy.org/docs.html

http://ipython.org/ipython-doc/stable/index.html

https://matplotlib.org/contents.html

https://machinelearningmastery.com/gentle-introduction-autocorrelation-partial-autocorrelation/

https://en.wikipedia.org/wiki/Window_function

https://www.slideshare.net/GourabGhosh4/overlap-add-overlap-savedigital-signal-processing