Protokol k projektu - Signály a systémy

1.)

Názov súboru	Dĺžka nahrávky [vzorky]	Dĺžka nahrávky [sekundy]
maskoff_tone.wav	64645	4.04
maskon_tone.wav	61301	3.83

2.)

Názov súboru	Dĺžka nahrávky [vzorky]	Dĺžka nahrávky [sekundy]
maskoff_sentence.wav	40496	2.53
maskon_sentence.wav	35295	2.21

3.)

A.)

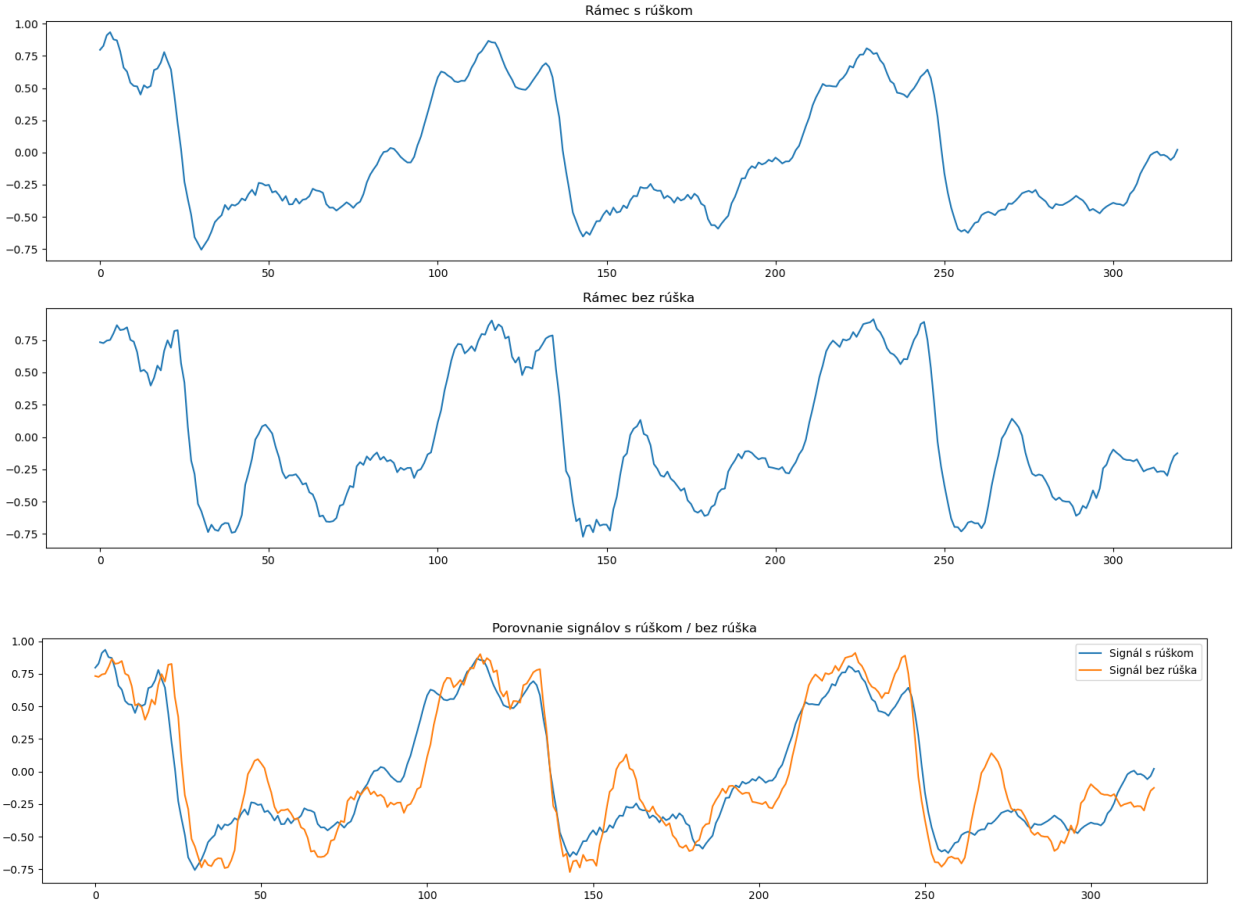
$$\text{Veľkosť rámca} = \text{vzorkovacia frekvencia} * \text{dĺžka rámca[s]}$$

V našom prípade mal rámec dĺžku 0.02 sekundy a vzorkovacia frekvencia bola 16 000 Hz

$$\text{Veľkosť rámca} = 16000 * 0.02$$

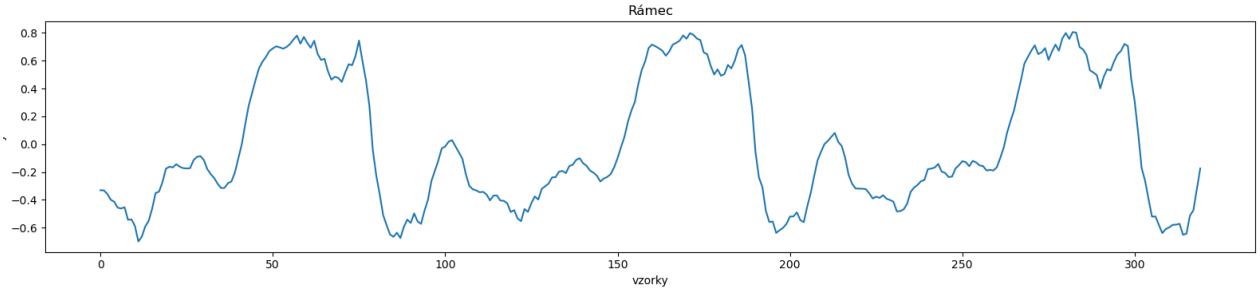
Výsledná veľkosť rámca pre tento prípad = 320 vzoriek.

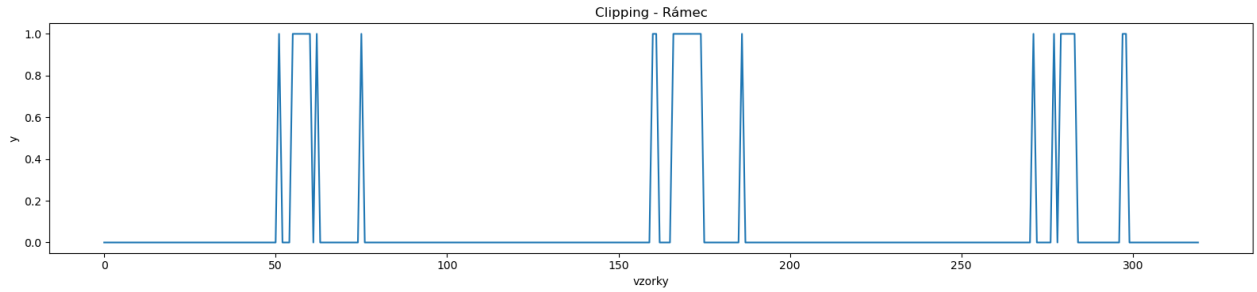
B.)

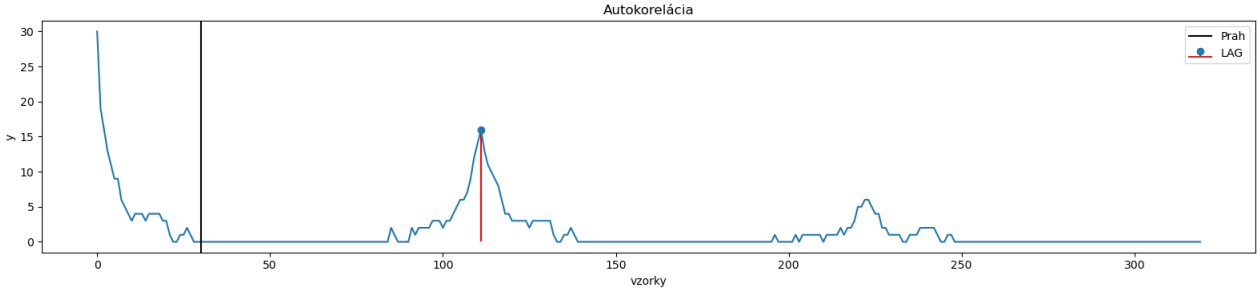


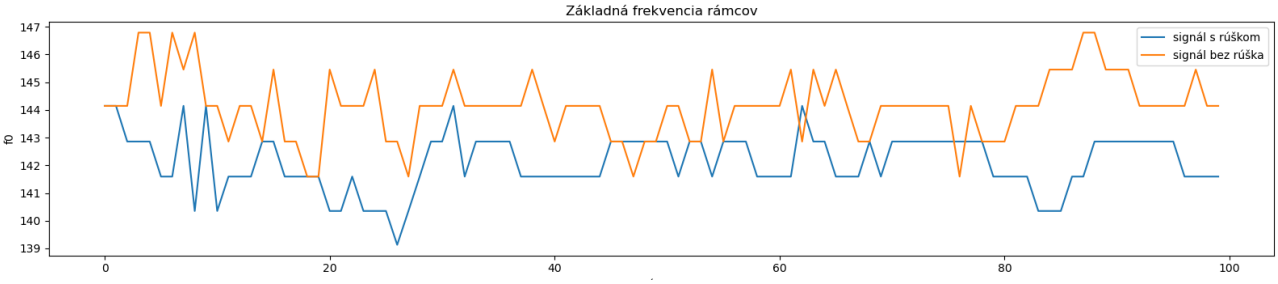
4.)

A.)









B.)

Rozptyl bez rúška: 1.3683279461180697

Rozptyl s rúškom: 1.018808349731852

Stredná hodnota bez rúška: 144.14414414414415

Stredná hodnota s rúškom: 141.5929203539823

C.)

Riešením je zmena vzorkovacej frekvencie. (napr. pri znížení vzorkovacej frekvencie z 16000 Hz na 8000 Hz dosiahneme zníženie rozdielu základnej frekvencie rámcov o polovicu.)

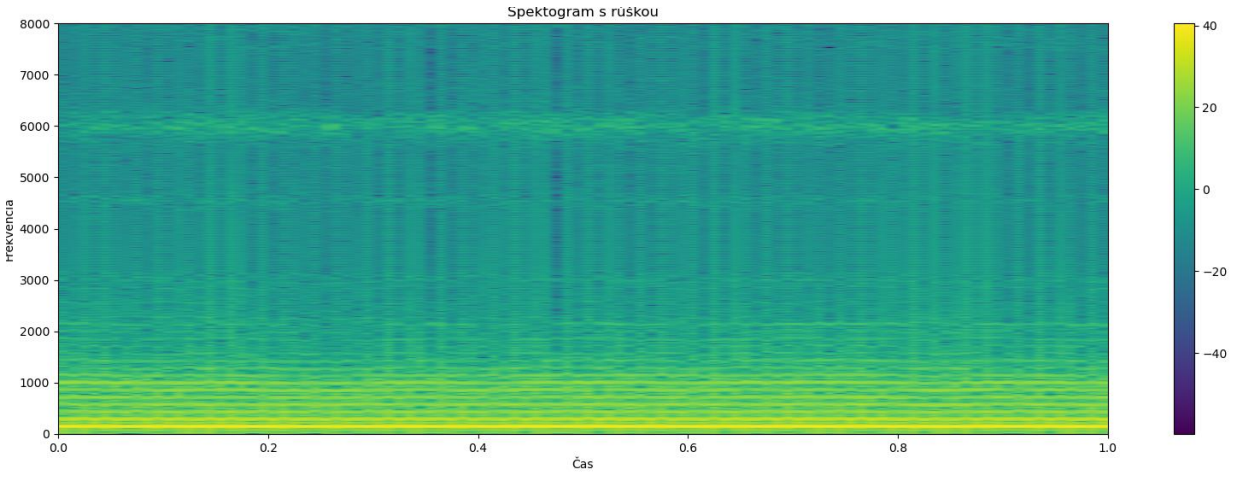
5.)

A.)

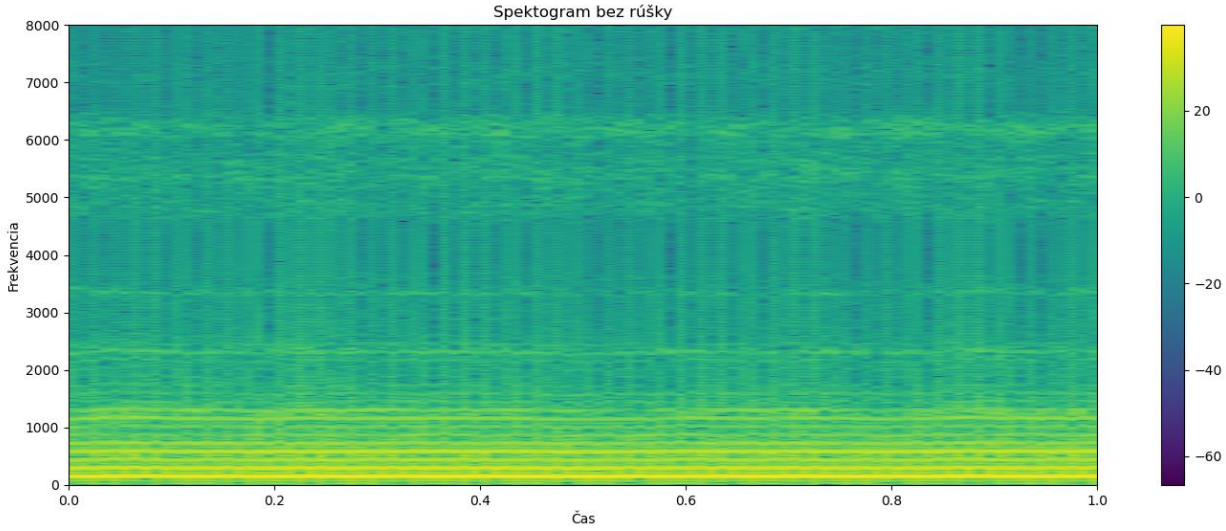
V zdrojovom súbore src/5.py je porovnanie s np.fft.fft na jednoduchom príklade, pri výpočtoch vzhľadom na vysokú časovú náročnosť tento algoritmus nepoužívam.

```
def my_dft(array, leng):
    arr_len = len(array)
    Xk = np.zeros(leng, dtype=complex)
    for k in range(leng):
        for n in range(arr_len):
            Xk[k] += array[n]*np.exp(-2j*np.pi*k*n/leng)
    return Xk
```

B.)



C.)



6.)

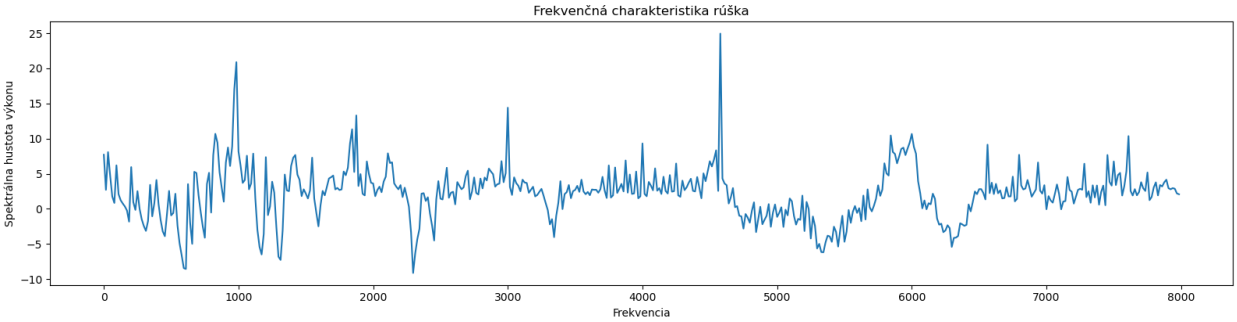
A.)

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

$$B(e^{jw}) = \text{výstup DFT} - \text{ signál s rúškom}$$

$$A(e^{jw}) = \text{výstup DFT} - \text{ signál bez rúška}$$

B.)



C.)

Frekvenčnú charakteristiku získame ako podiel výstupu Diskrétnej Fourierovej transformácie implementovanej v úlohe 5. Vzorky priemerujem cez všetkých 100 rámcov. Pri vykreslení na výkonové spektrum využívame rovnaký vzorec ako v 5. úlohe. Ako môžeme vidieť z grafu hodnoty sú pre vysoké frekvencie pomerne stabilné avšak občas sa v spektre vyskytne výraznejší skok.

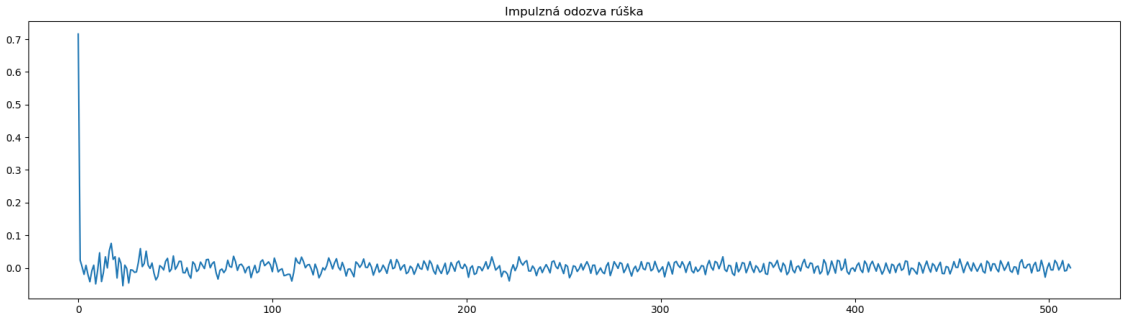
7.)

A.)

V zdrojovom súbore src/7.py je porovnanie s np.fft.ifft na jednoduchom príklade, pri výpočtoch vzhľadom na vysokú časovú náročnosť tento algoritmus nepoužívam.

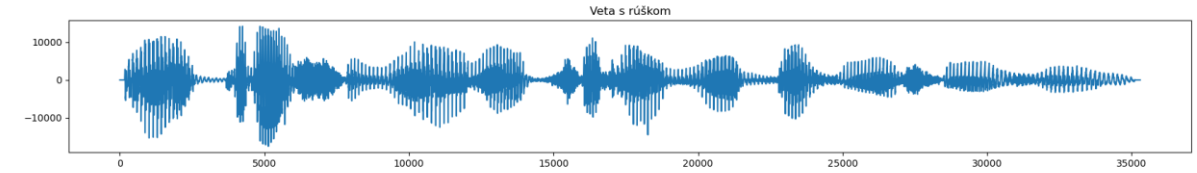
```python
def my_inverse_dft(array):
    arr_len = len(array)
    xn = np.zeros(arr_len, dtype=complex)
    for n in range(arr_len):
        for k in range(arr_len):
            xn[n] += array[k] * np.exp(2j*np.pi*(k/arr_len)*n) # arr_len = N
        if xn[n] != 0.0 and arr_len != 0:
            xn[n] = xn[n] / arr_len
    return xn
```

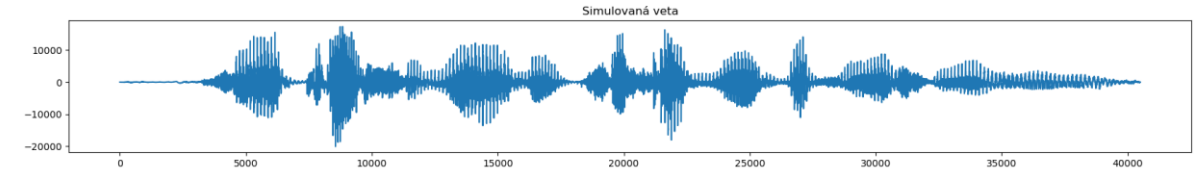
B.)

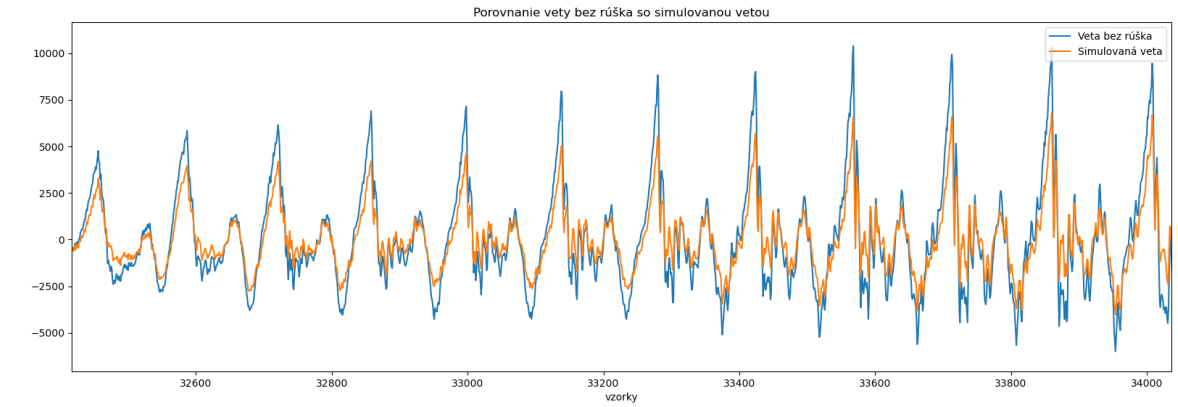


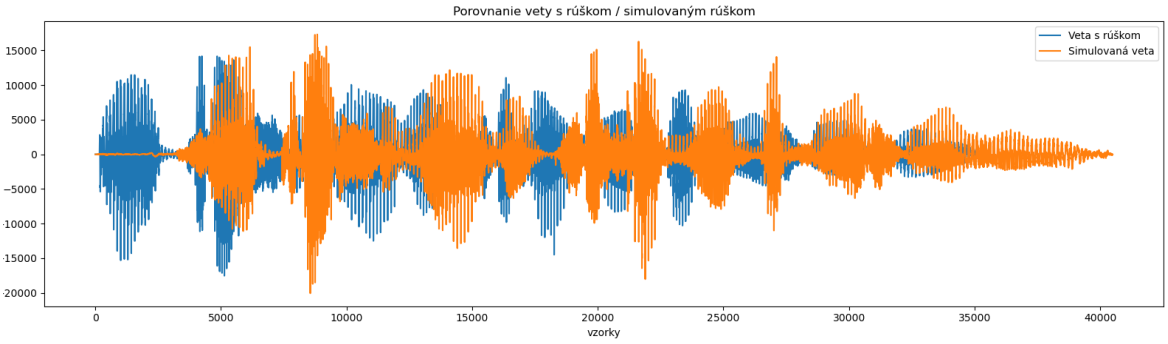
8.)

A.)









B.)

Pôvodne som využíval iba prvú polovicu výstupu DFT a tak bola veta so simulovaným rúškom značne zosilnená oproti vete s rúškom. Pri aktuálnom riešení som pracoval s celým výstupom DFT. Veta so simulovaným rúškom je oproti vete bez rúška pri vyšších hodnotách oslabená. Naopak veta so simulovaným rúškom a rúškom dosahuje pri vyšších hodnotách podobné výsledky ako veta nahrávaná s rúškom, avšak tempo reči sa mi nepodarilo pri nahravkách trafiť zhodne.

9.)

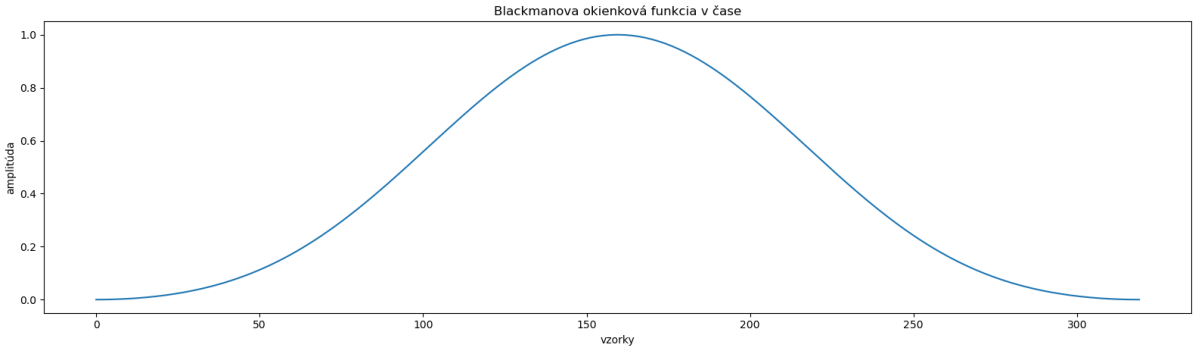
Najväčším problémom pri vypracovaní projektu bola absencia môjho hudobného sluchu a vytvorenie schopnej nahrávky. Porovnanie nahrávok pri základnom riešení neuspokojilo moje očakávania, avšak v doplňujúcich úlohách po určitých vylepšeniach sa simulované nahrávky oveľa viac zhodovali s nahrávkami s rúškom. Pri nahrávaní som veľakrát narazil na problémy, ktoré moju implementáciu projektu značne predĺžili.

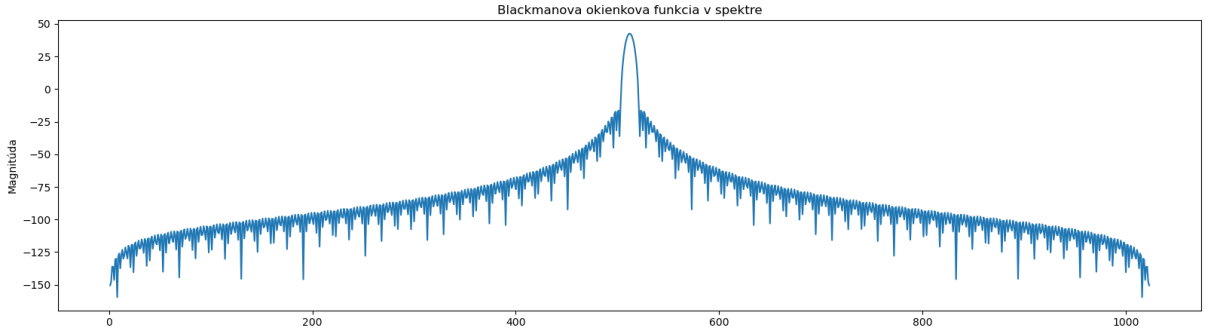
11.)

Pre túto úlohu som si zvolil Blackmanovú okienkovú funkciu.

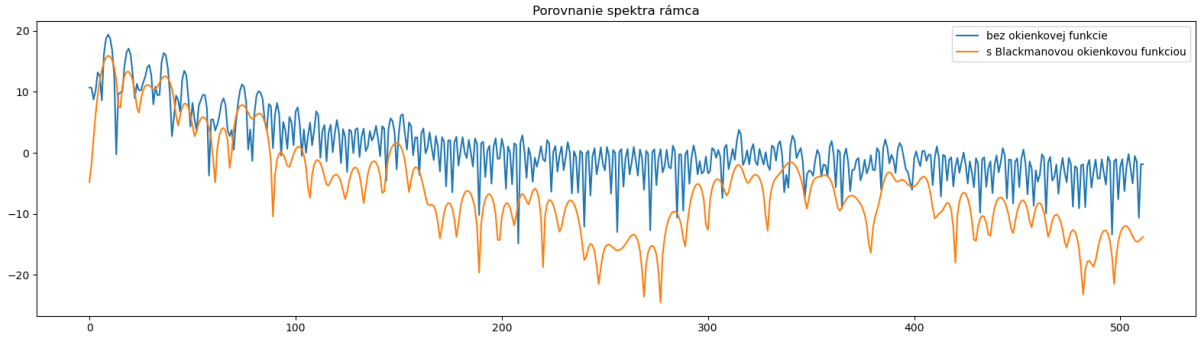
A.) $$w(n) = 0.42 - 0.5cos\left(\frac{2\pi n}{M}\right) + 0.08cos(\frac{4\pi n}{M})$$

B.)





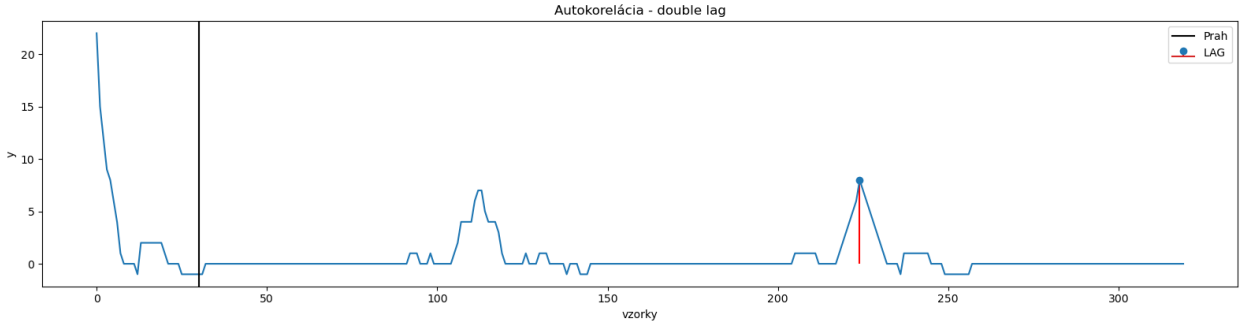
C.)

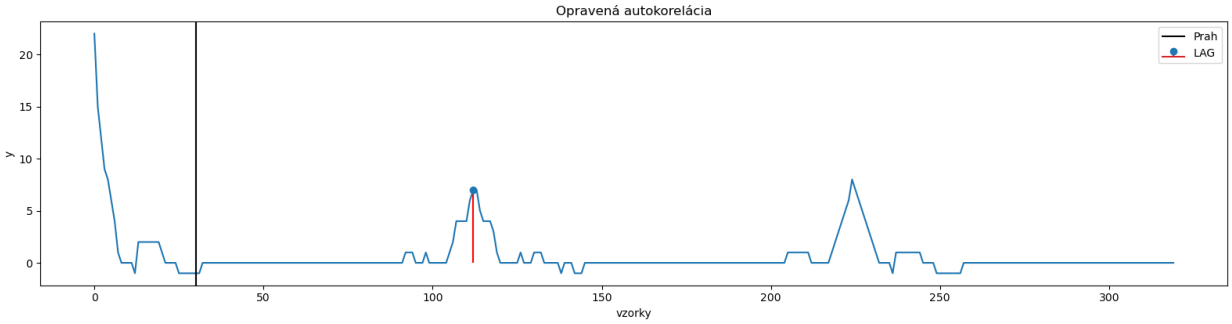


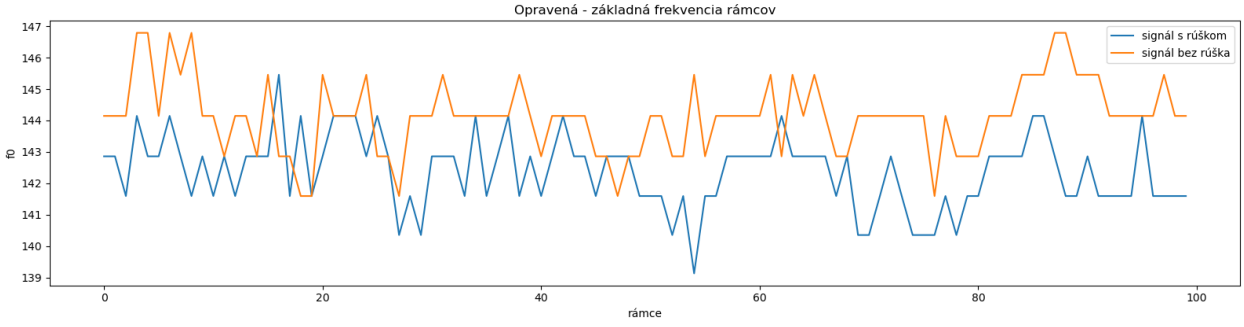
Okienková funkcia je užitočná, pretože tlmí začiatočné a koncové vzorky každého rámca a naviac vyhladzuje spektrum rámcov. Výsledné simulované nahrávky po použití okienkovej funkcie znejú oveľa lepšie ako bez použitia tejto metódy.

12.)

A.)



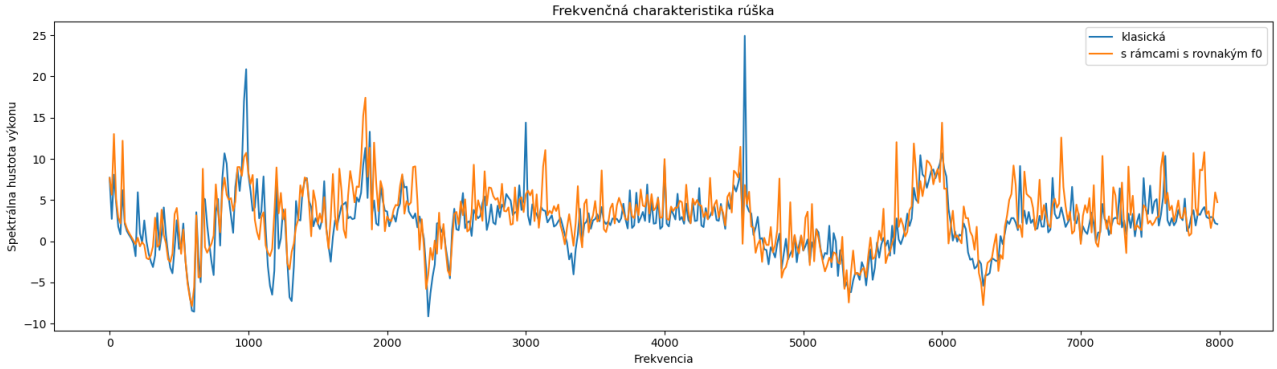




B.)

Pri odstraňovaní chyby som sa inšpiroval nelineárnou filtráciou mediánovým filtrom. Filter som zvolil tak aby som sa vyhol prílišnému vyhľadaniu f0. Hodnoty ktoré filter neopravuje sú v rozmedzí od medián lagu – prah po medián lagu + prah, pričom hodnota môjho prahu je 32, čo zodpovedá 10% dĺžky rámca.

13.)



Frekvenčná charakteristika spočítaná z rámcov s rovnakým základným tónom sa oproti klasickej frekvenčnej charakteristike spočítanej v predchádzajúcej úlohe (6.) líši tým, že tlmí veľké skoky frekvenčnej charakteristiky.

Zdroje:

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

https://numpy.org/doc/

https://www.mathworks.com/matlabcentral/answers/38316-obtain-the-impulse-response-from-a-frequency-response

https://tomroelandts.com/articles/how-to-plot-the-frequency-response-of-a-filter?fbclid=IwAR1NSCX-dKtnyWZAiM58vcana_BjqLmz-0wCN92jYAIAdrWbgIS4JvWDnl0

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

https://www.fit.vutbr.cz/study/courses/ISS/public/