ISS Projekt 2020 / 21

Protokol

Jakub Duda, xdudaj02

Použité knižnice

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.io import wavfile
from scipy import signal
import copy
import math
import cmath
```

Komplikácie

Z dôvodu hudobnej neschopnosti, som musel využiť nahrávku tónu v podaní mojej mamy.

Programová implementácia má z estetického hľadiska značné nedostatky a mohol som sa viac posnažiť pri názvoch premenných, no spätne sa neodvažujem do toho zasahovať.

Pri funkciách ako dft, idft či autokorelácia som z dôvodu časovej náročnosti nevyužil vlastné implementácie týchto funkcií ale radšej vstavané funkcie s identickou funkcionalitou.

Čas behu programu je napriek tomu mierne vyšší, čo je spôsobené cross-koreláciou nahrávok tónu, ktorých dĺžka sa pohybuje až okolo osem sekúnd.

Úloha č. 1

Tabuľka s informáciami o nahrávkach tónu:

maskoff_tone.wav	00:07.48	119 741
maskon_tone.wav	00:08.18	130 810

Úloha č. 2

Tabuľka s informáciami o nahrávkach vety:

maskoff_sentence.wav	00:02.88	46 080
maskon_sentence.wav	00:02.82	45 056

Úloha č. 3

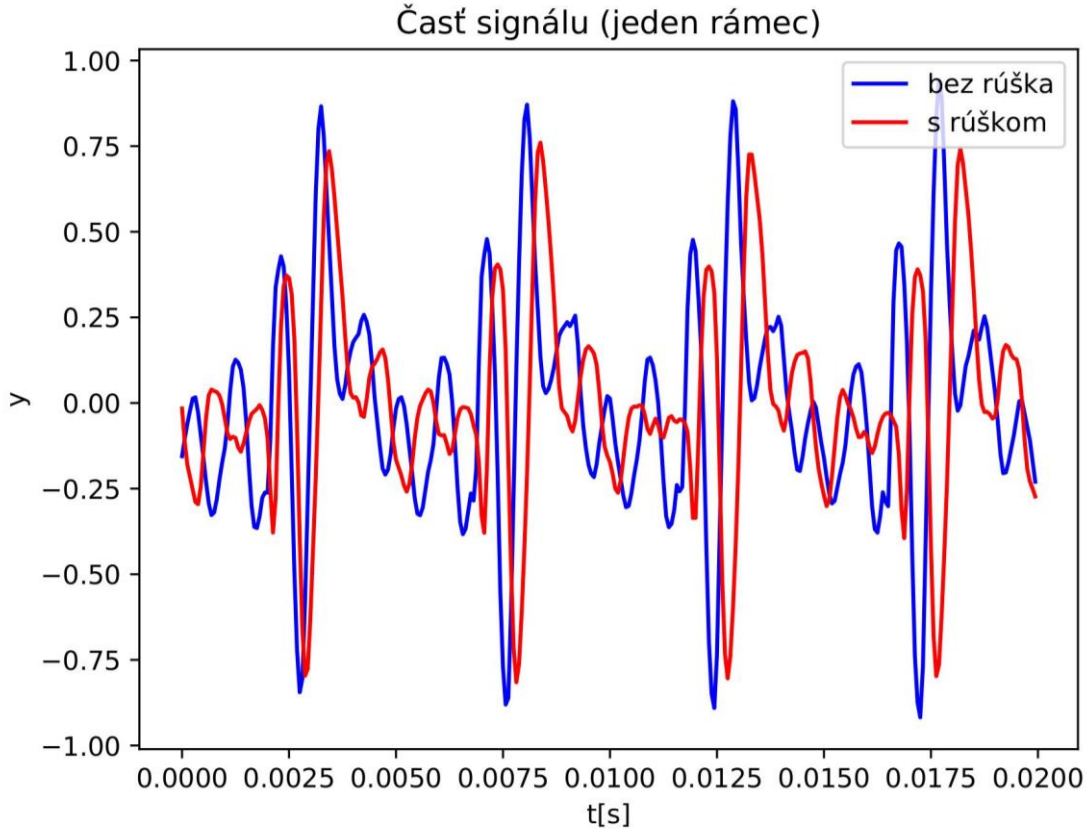
Vzorec pre výpočet veľkosti rámca vo vzorkách:

$$\text{frekvencia } [1/\text{s}] * \text{dĺžka rámca } [\text{s}] \quad => \quad 16\,000 * 0.02 = 320$$

Ručne som vybral časť dlhú jednu sekundu z prevej nahrávky a následne som pomocou cross-korelácie našiel najpodobnejšiu časť v druhej nahrávke.

Využil som 99 rámcov s dĺžkou 20 ms.

Graf jedného rámca:

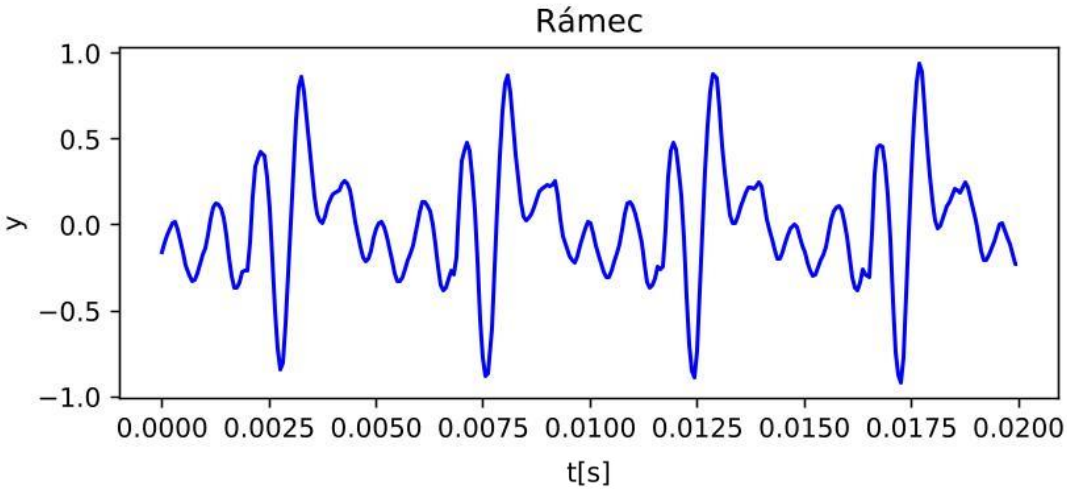


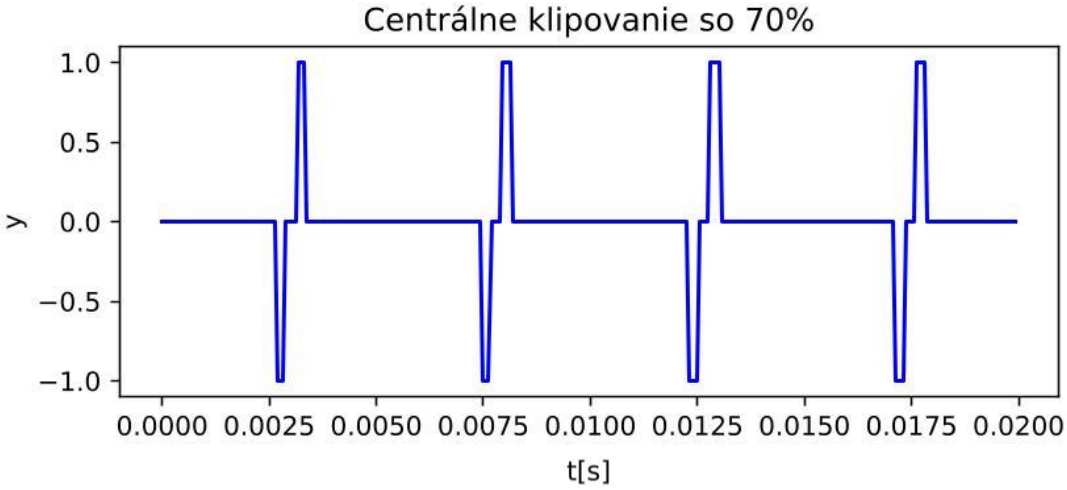
Úloha č. 4

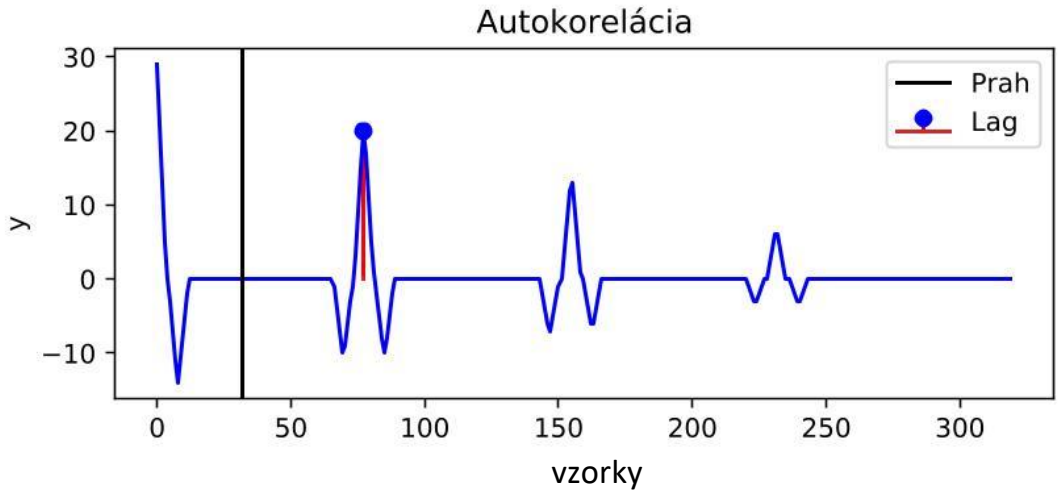
Vlastná implementácia auto-korelácie:

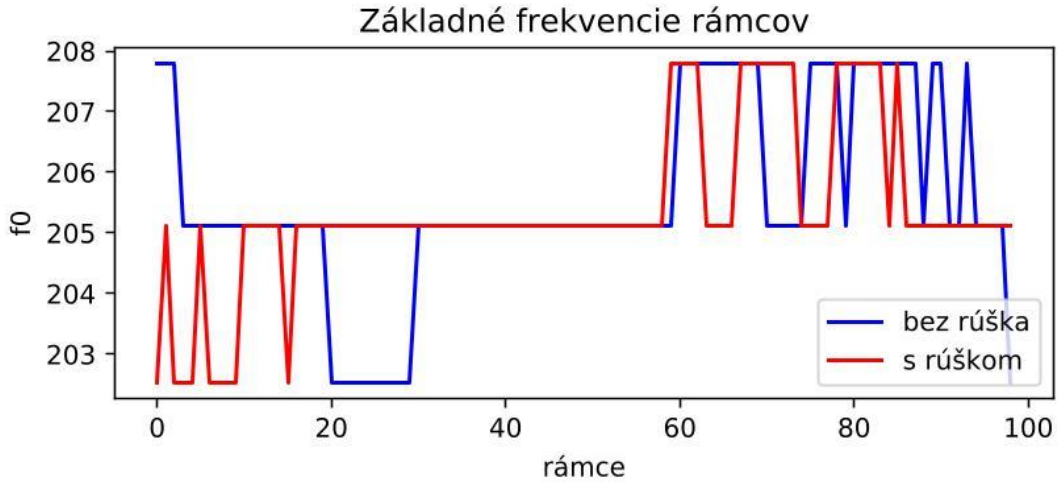
```
def autocorrelation(in_list):
    in_list = in_list.tolist()
    shift_list = copy.deepcopy(in_list)
    out_list = []
    for i in range(len(in_list)):
        if i != 0:
            in_list.append(0)
            shift_list.insert(0, 0)
        total = 0
        for j, k in zip(in_list, shift_list):
            total += j * k
        out_list.append(total)
    out_list = np.array(out_list)
    return out_list
```

Grafy:









Vypočítané hodnoty zaokrúhlené na tri desatinné miesta:

Stredná hodnota základnej frekvencie nahrávky bez rúška: 205.593 Hz

Stredná hodnota základnej frekvencie nahrávky s rúškom: 205.377 Hz

Rozptyl základnej frekvencie nahrávky bez rúška: 2.540 Hz

Rozptyl základnej frekvencie nahrávky s rúškom: 1.842 Hz

Odpoveď na otázku: *"Jistě jste si všimli, že pokud se hodnota "lag-u" liší o 1, poměrně dost to zamává s frekvencí. Jak by se dala zmenšit velikost změny f0 při chybě ± 1? "*:

Áno všimol som si toto správanie. Vďaka môjmu už spomínanému hudobnému "nadaniu" a teda tomu, že mi nahrávku tónu nahrávala moja mama, má táto nahrávka vyššiu frekvenciu a teda signál má menšiu periódu, čo spôsobuje, že lag sa vyskytne v rámci pomerne skoro a teda s nižším indexom. Keďže pre zistenie základnej frekvencie využívame vzťah $16\,000 \,/\, index\ lagu$, pri nižšom indexe lagu sa jeho rozdiely prejavia výraznejšie.

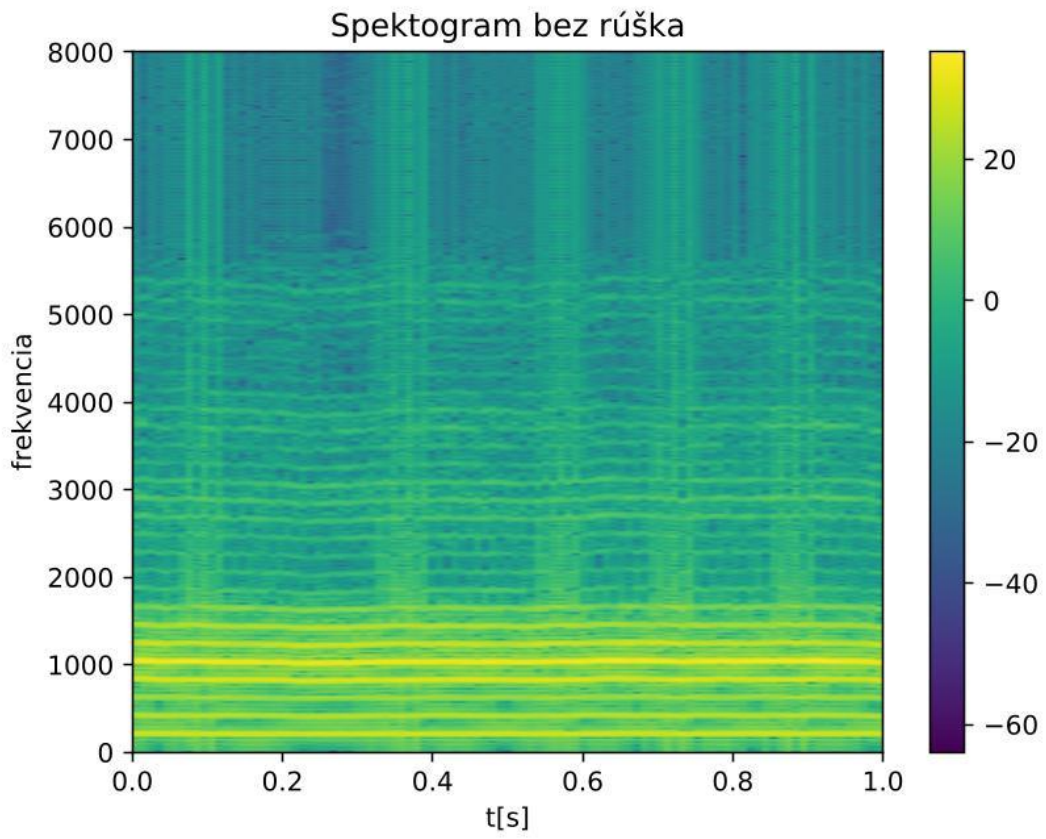
Vyriešiť by sa to dalo zvýšením hodnôt indexu lagu, čo sa dá dosiahnuť napríklad nahrávkou hlbšieho tónu.

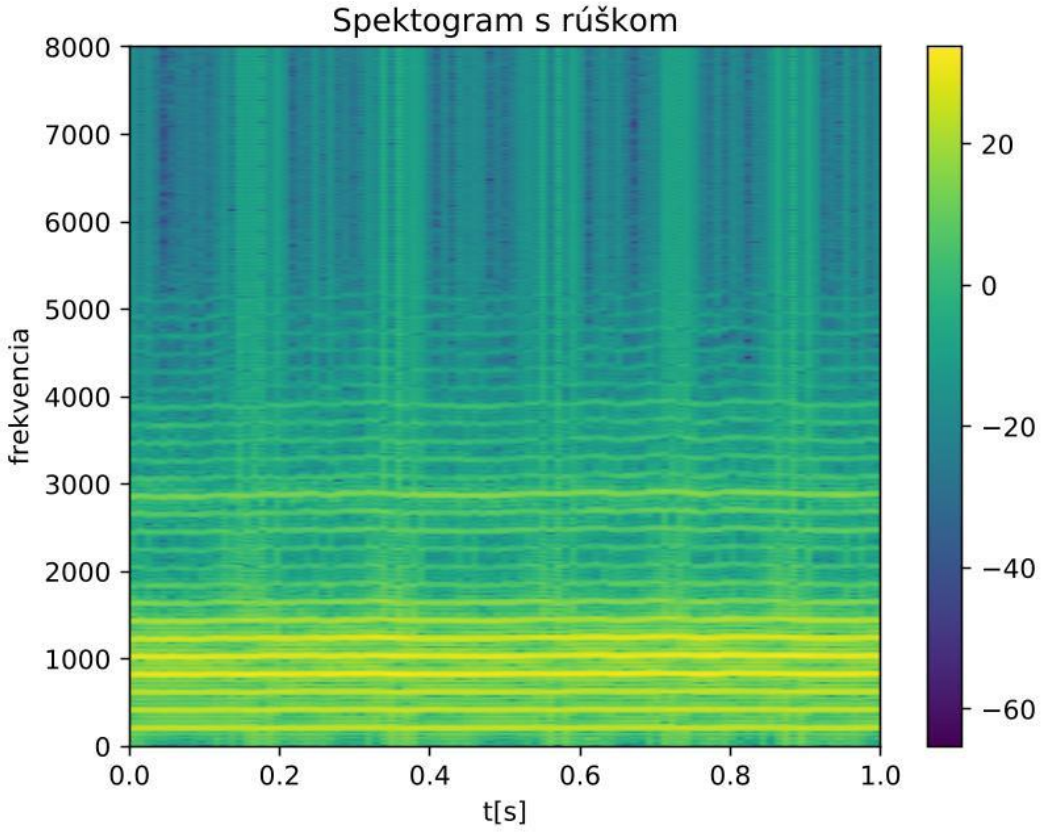
Úloha č. 5

Vlastná implementácia DFT:

```
def dft(in_list, size):
    in_list = in_list.tolist()
    out_list = []
    for i in range(size - len(in_list)):
        in_list.append(0)
    for k in range(size):
        total = 0
        for n in range(size):
            x = 2 * math.pi * n * k / size
            total += in_list[n] * (math.cos(x) - math.sin(x) * 1j)
        out_list.append(total)
    out_list = np.array(out_list)
    return out_list
```

Spektogramy:




Spektogram s rúškom
frekvencia
8000
7000
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
t[s]
20
0
−20
−40
−60

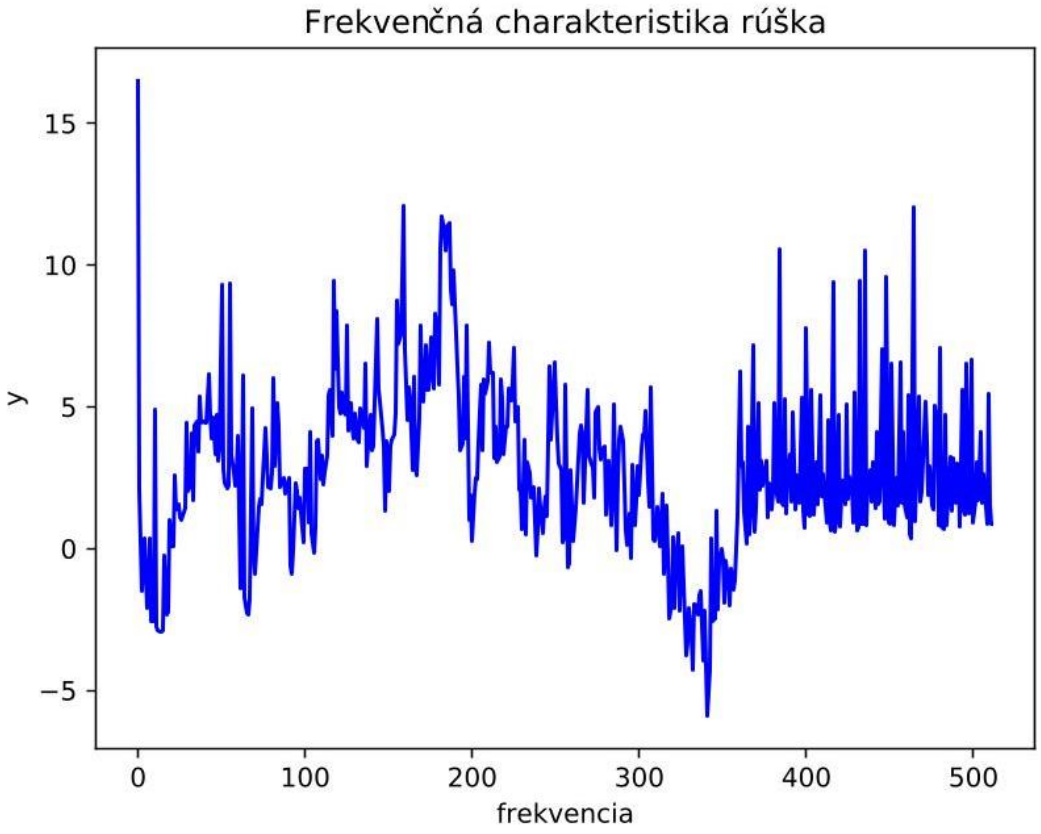
Úloha č. 6

Vzťah pre výpočet frekvenčnej charakteristiky rúška:

$$H(e^{j\omega}) = \frac{Y(e^{j\omega})}{X(e^{j\omega})} \ ,$$

kde $Y(e^{j\omega})$ je Fourierova transformácia výstupného signálu (signál s rúškom) a $X(e^{j\omega})$ je Fourierova transformácia vstupného signálu (signál bez rúška)

Frekvenčná charakteristika rúška:



Krátky komentár k filtru:

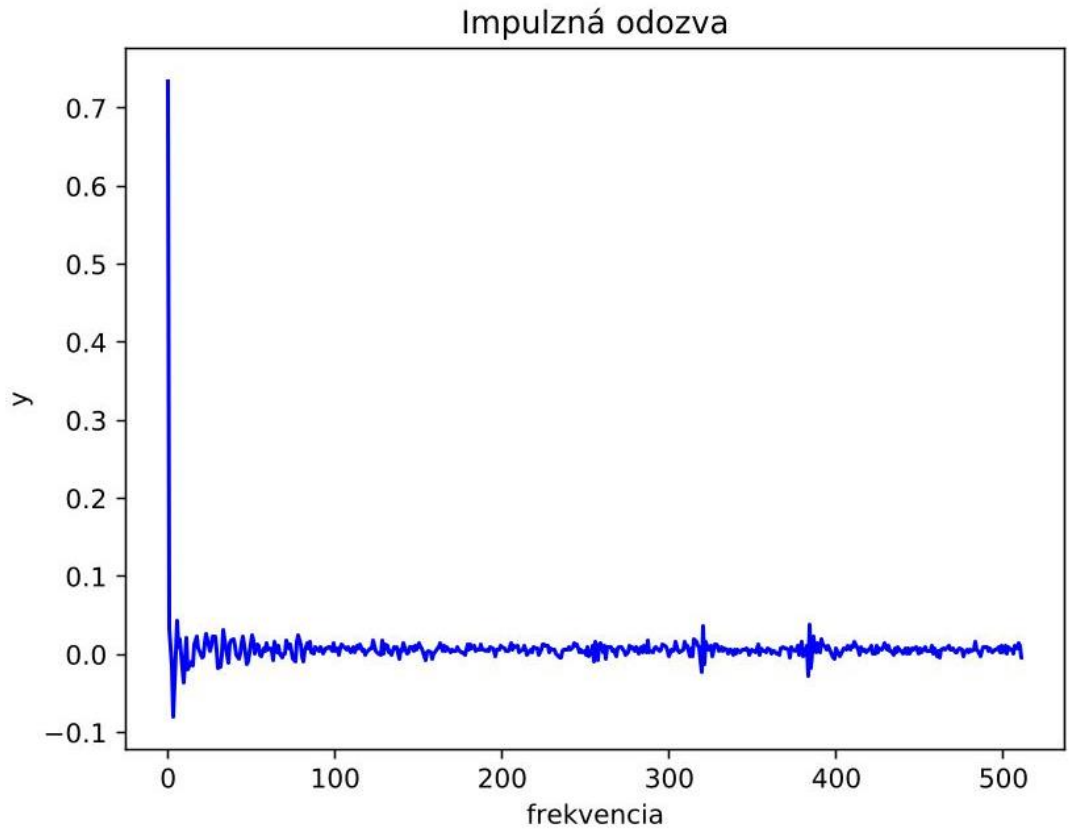
Filter je v pohode.

Úloha č. 7

Vlastná implementácia IDFT:

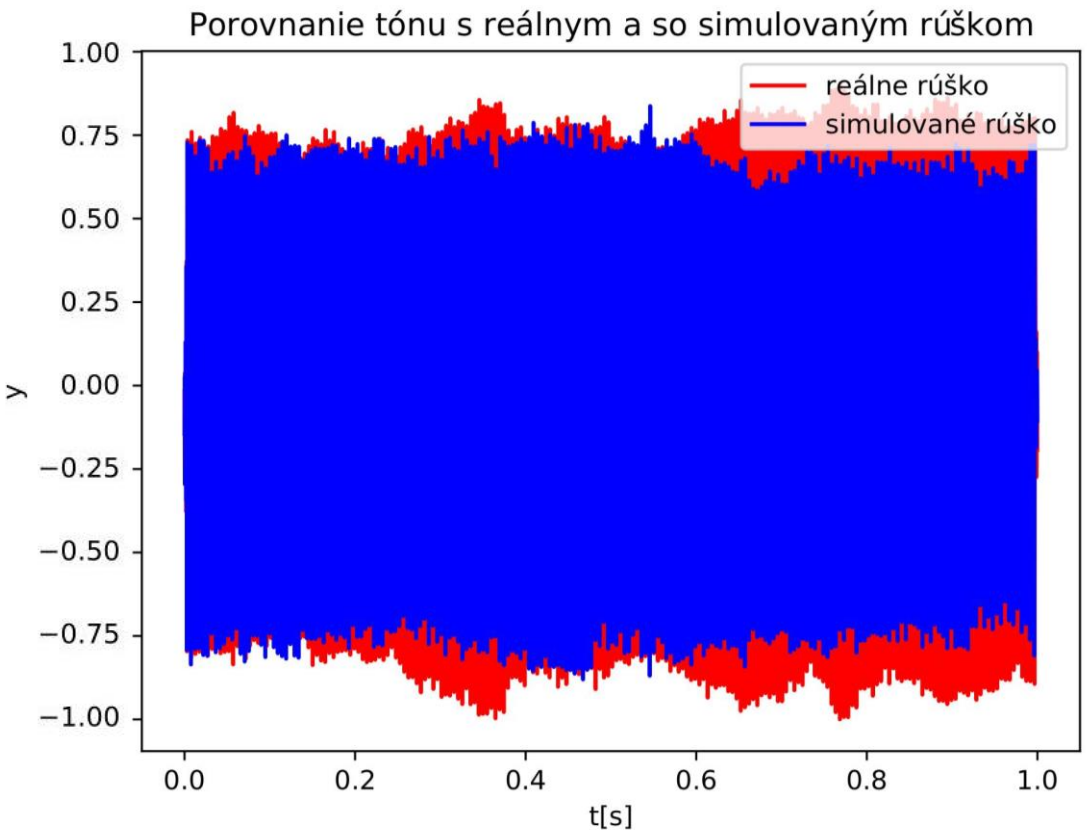
```
def idft(in_list):
    in_list = in_list.tolist()
    size = len(in_list)
    out_list = []
    a = 2j * math.pi / size
    for n in range(size):
        total = 0
        for k in range(size):
            total += in_list[k] * cmath.exp(a * k * n)
        out_list.append(total / size)
    out_list = np.array(out_list)
    return out_list
```

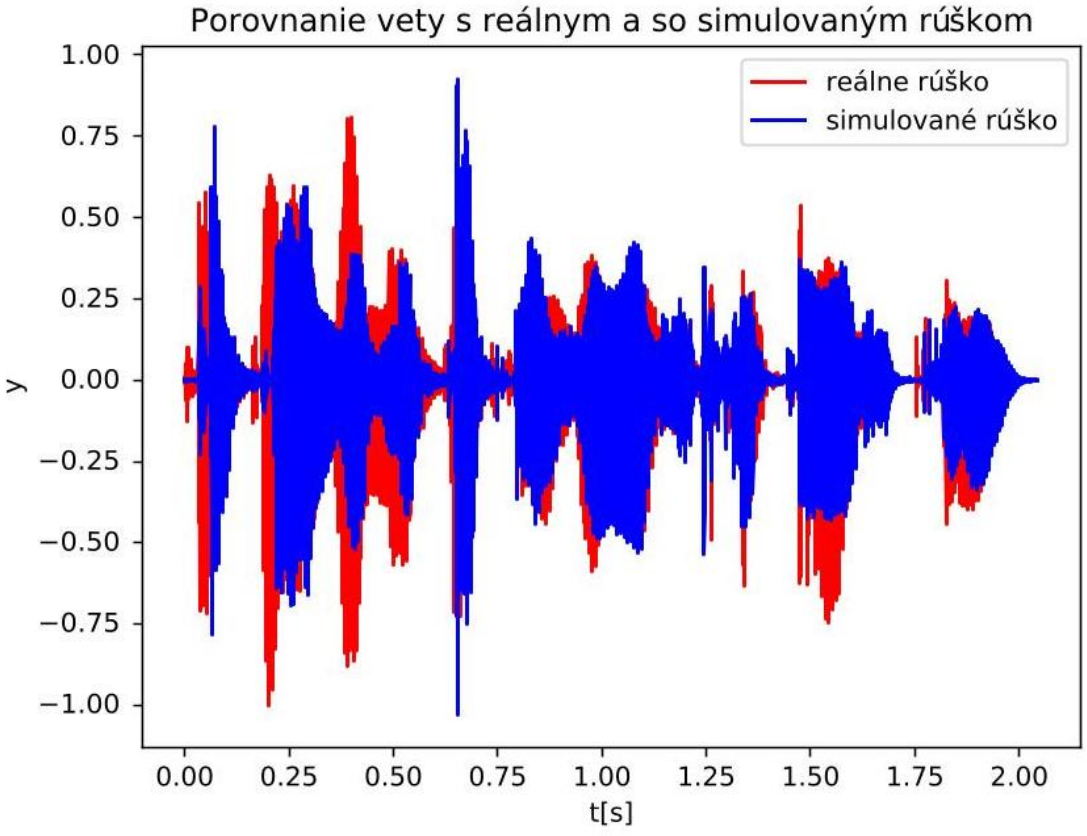
Graf impulznej odozvy rúška:



Úloha č. 8

Grafy:





Odpovede na otázky:

Myslím si, že signály sú celkom podobné.

Úloha č. 9 (Záver)

Moje riešenie si myslím, že celkom slušne funguje. Pri porovnaní pôvodného signálu pred prevedením cez filter (simulované rúško) a signálu po prevedení, je značne vidieť rozdiel, teda filter očividne filtruje. Taktiež vracia celkom podobné hodnoty ako sú v nahrávke s použitím skutočného rúška. Pri akustickej kontrole už je síce rozdiel citeľný, keď pri nahrávke so skutočným rúškom je zvuk viac tlmenejší, pričom nahrávka so simulovaným rúškom nemá až tak tlmený zvuk a menej sa líši od nahrávky bez rúška.

Som si istý, že výsledky nie sú nijak svetoborné a mohli by byť lepšie, no som s nimi celkom spokojný (myslím si, že implementácia je celkom fajn).

Bonusová úloha č. 12

Chybu detekcie n-násobného lagu som vďaka kvalitnému speváckemu výkonu mojej mamy nezaznamenal. Jej výskyt by som vedel docieliť napríklad výrazným posunutím prahu detekcie, až za prvý lag (190 Hz namiesto 500 Hz). Toto, by ale spôsobilo preskočenie prvého lagu v každom rámci. Kvôli veľkej pravidelnosti výskytu lagu, sa mi nedá zvoliť vhodný prah detekcie, kde by bolo možné aplikovať opravnú funkciu.

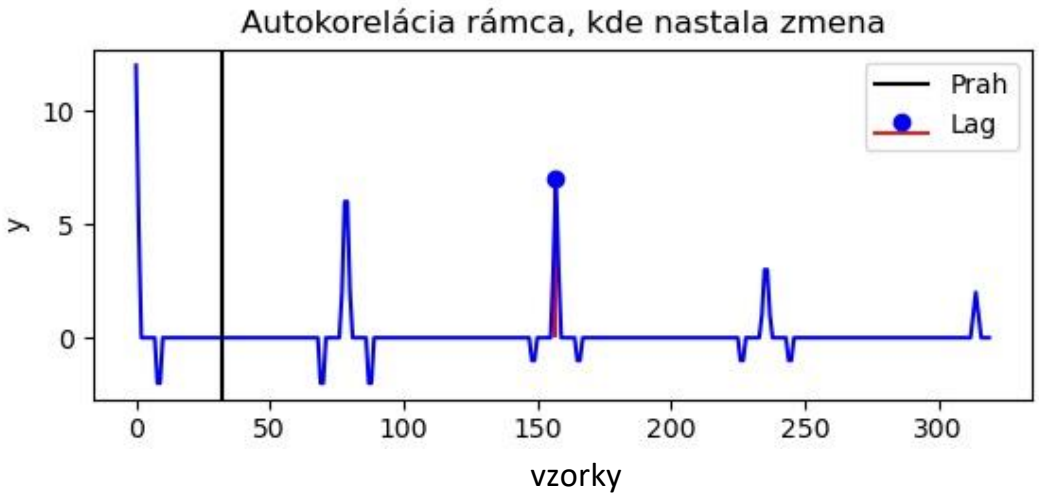
Avšak, zmenou úrovne klipovania sa mi to podarilo bez problémov.

Vlastná implementácia mediánovej filtrácie:

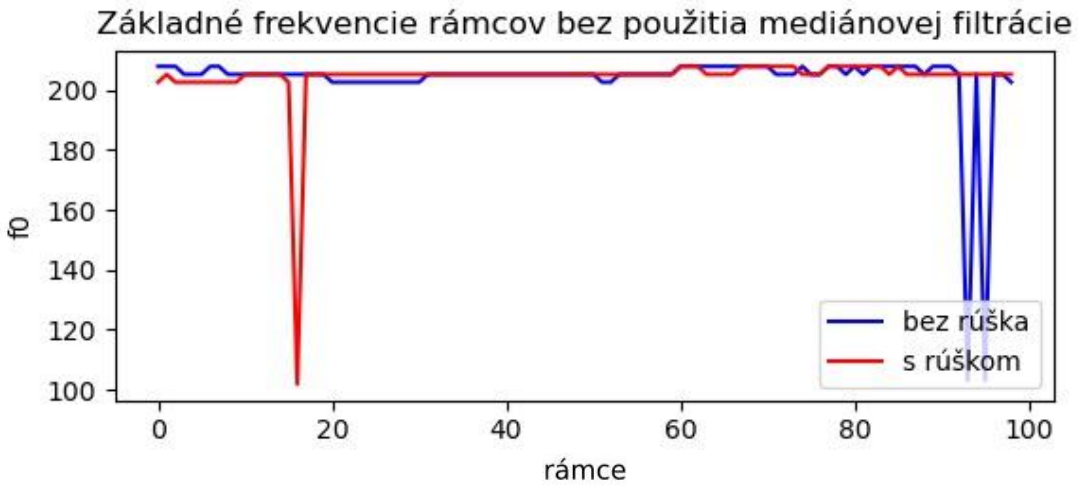
```
def median_filtr(data, median_from):
    for i in range(len(data)):
        k = int((median_from - 1) / 2)
        if i < k:
            k = i
        if i >= (len(data) - k):
            k = len(data) - i - 1
        data[i] = np.median(data[i - k:i + k + 1])
    return data
```

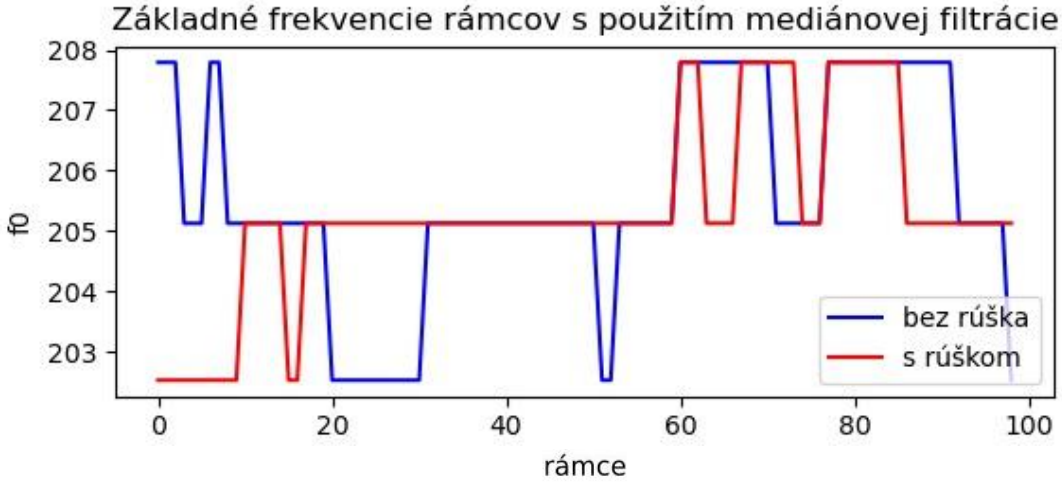
Pri zmene úrovne klipovania na 0.9 nastala v niekoľkých rámcoch zmena. Napríklad v rámci číslo 16 nastal double lag a následne v ňom preto vyšla polovičná frekvencia.

Graf rámca číslo 16 s úrovňou klipovania 0.9:



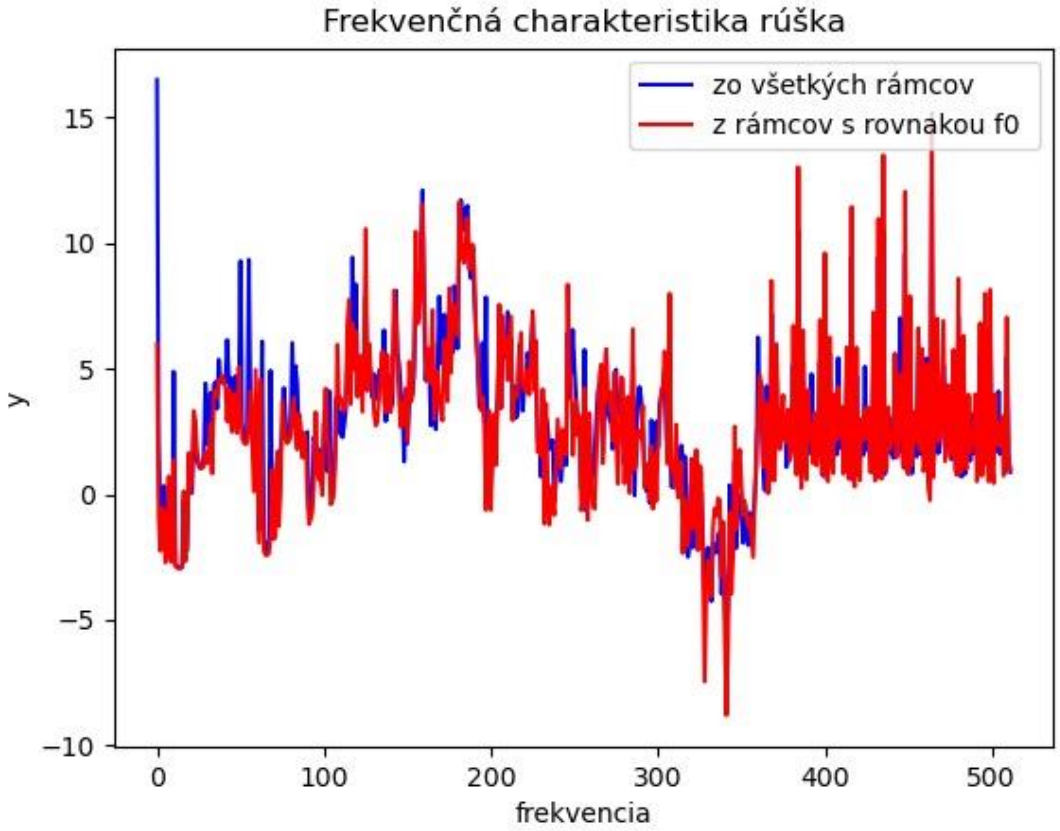
Grafy pri úrovni klipovania 0.9:





Bonusová úloha č. 13

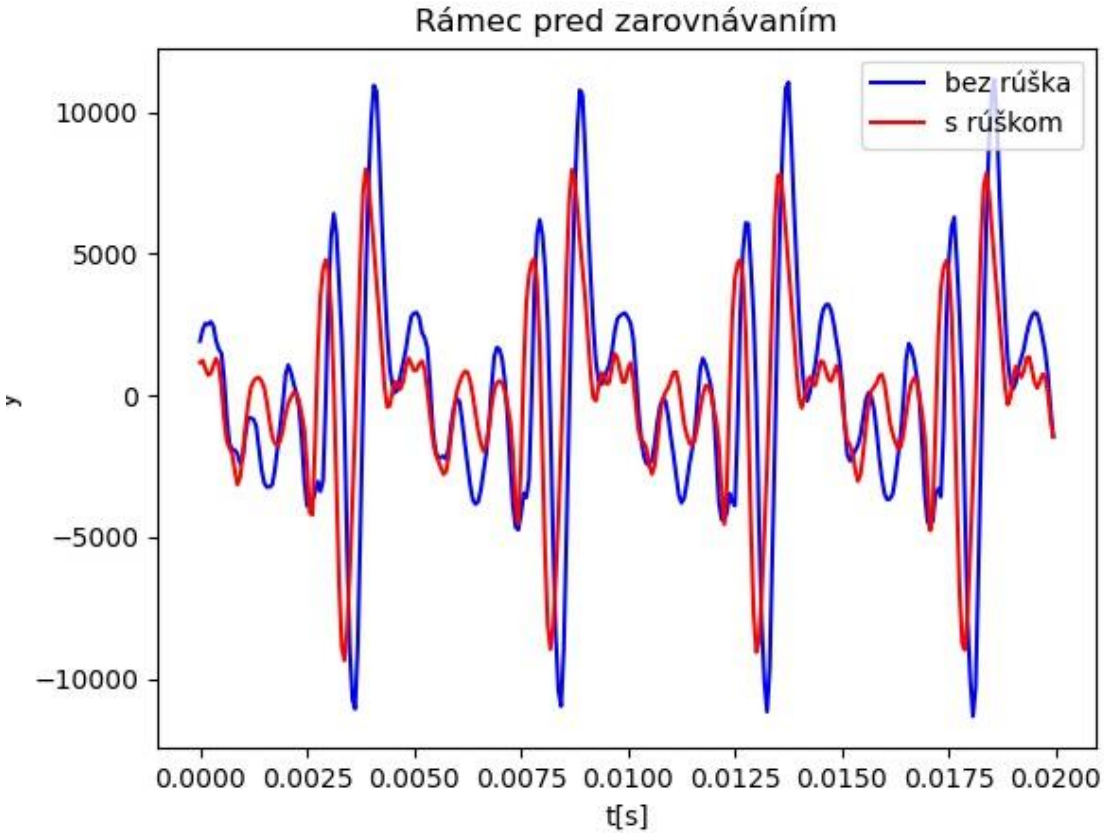
Graf:

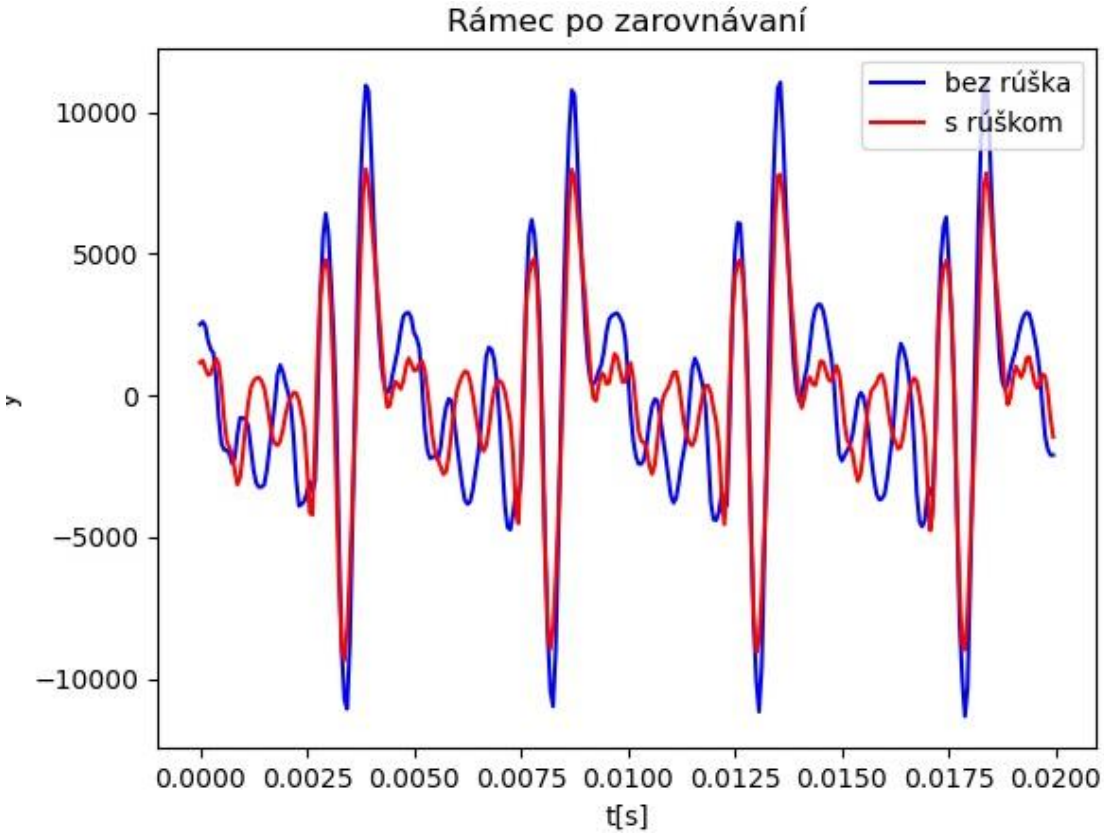


Frekvenčná charakteristika rúška vypočítaná iba z rámcov s rovnakou základnou frekvenciou vyšla vo väčšej miere podobná ako frekvenčná charakteristika rúška vypočítaná zo všetkých rámcov.

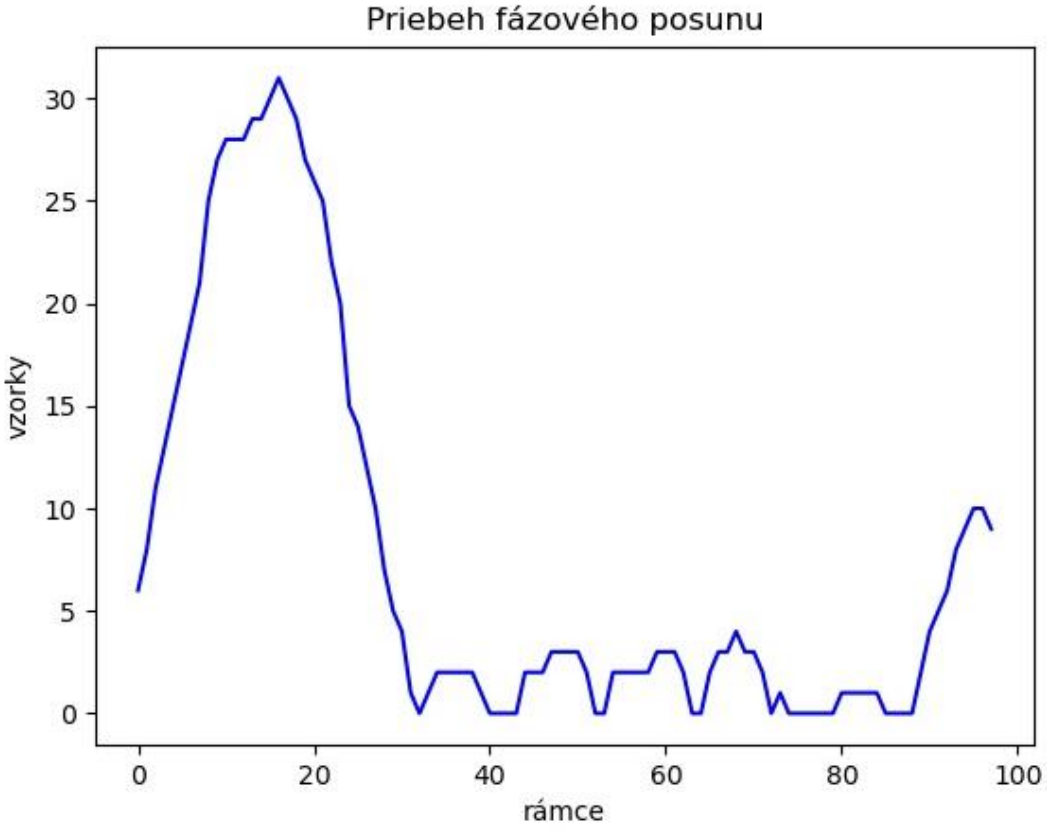
Bonusová úloha č. 15

Graf rámcu číslo 61:





Graf:



Teoretická otázka:

> Je to spôsobené tým, že fázový posun doľava udáva o koľko sa musí posunúť jeden signál voči druhému, aby boli zarovnané a fázový posun doprava udáva o koľko sa musí tento istý signál posunúť doprava aby boli zarovnané. Posunom doľava by sa tento signál zarovnal napríklad s *k*-tou periódou druhého signálu a posunom doprava by sa teda zarovnal s *k+1*-ou periódou. Z toho logicky vyplýva, že súčet posunu doľava a posunu doprava sa rovná dĺžke jednej periódy.
>
> A kedže vieme, že lag určuje po akom čase sa signál na seba, čo najviac podobá, čo je v podstate dĺžka periódy, je logické, že sčítaný fázový posun a lag nadobúdajú veľmi podobné hodnoty.
>
> Výnimkou je sitácia, kedy sú signály presne zarovnané a fázové posuny na obidve strany vychádzajú nulové.

Záver (doplnenie)

Po vyriešení niekoľkých bonusových úloh som zo svojim riešením viac spokojný. Úlohy 12 a 13 vyprodukovali podobné výsledky ako základná časť projektu, no po prihliadnutí na fázový posun v úlohe 15 sa simulovaný signál viac podobá na signál s normálnym rúškom. Tak či tak je rozdiel medzi nimi dosť veľký a úplne sa mi nepodarilo dosiahnuť také stlmenie signálu aké spôsobuje ozajstné rúško.