VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Signály a systémy – Projekt

Protokol

4. ledna 2021 Jozef Makiš (xmakis00)

Obsah

1 Úvod

V nasledujúcich sekciách sa nachádza vypracovaný protokol k projektu do predmetu Signály a systémy. Prvé štyri úlohy sa nachádzajú v súbore `moj_projekt`, 5-8 v súbore `moj_projekt5` a doplňujúce úlohy sú v súboroch samostatne. Každá implementácia DFT pre rámce s rúškou aj bez môže trvať až 10-15 minút. Obe tieto implementácie sú vo funkciách v súbore `moj_projekt5`. Dajú sa zakomentovať v prípade opravovania. V projekte využívam FFT, kvôli rýchlosti výpočtu. Na merlinovi som nenašiel knižnice, ktoré môj projekt využíva. Dajú sa nainštalovať vďaka súboru `requirements.txt`, a prákazu `pip install -r requirements.txt`. Ide hlavne o knižnice `soundfile` a `copy`.

2 Úloha 1

Nahral som dve testovacie nahrávky `maskoff_tone.wav` a `maskon_tone.wav` s rovnakým tónom a samohláskou „a“ bez rúšky a s rúškou. Nahral som ich pomocou mobilu s androidom tak som potreboval upraviť nahrávky pomocou `ffmpeg`. Po upravení nahrávky podľa požiadavok som použil utilitu soxi pre overenie nahrávok a vytvorenie príslušnej tabuľky.

2.1 Použité príkazy:

```
ffmpeg -i  maskoff_tone.mp3 -ar 16000 -ac 1 -acodec pcm_s16le maskoff_tone.wav
soxi maskoff_tone.wav
```

Input File:	'maskoff_tone.wav'
Channels:	1
Sample Rate:	16000
Precision:	16-bit
Duration:	00:00:04.43 = 70848 samples ~ 332.1 CDDA sectors
File Size:	142k
Bit Rate:	256k
Sample Encoding:	16-bit Signed Integer PCM

```
ffmpeg -i  maskon_tone.mp3 -ar 16000 -ac 1 -acodec pcm_s16le maskon_tone.wav
soxi maskon_tone.wav
```

Input File:	'maskon_tone.wav'
Channels:	1
Sample Rate:	16000
Precision:	16-bit
Duration:	00:00:07.49 = 119808 samples ~ 561.6 CDDA sectors
File Size:	240k
Bit Rate:	256k
Sample Encoding:	16-bit Signed Integer PCM

3 Úloha 2

Podobne už ako v predchádzajúcej úlohe som nahral dve nahrávky viet a následne som ich upravil pre možné použitie v projekte. Nahrávky `maskoff_sentence.wav` a `maskon_sentence.wav` som následne preveril aby vyhovovali požiadavkom zadania.

3.1 Použité príkazy:

```
ffmpeg -i  maskon_sentence.mp3 -ar 16000 -ac 1 -acodec pcm_s16le \
maskon_sentence.wav
soxi maskon_sentence.wav
```

Input File:	'maskon_sentence.wav'
Channels:	1
Sample Rate:	16000
Precision:	16-bit
Duration:	00:00:02.60 = 41610 samples $\sim$ 195.047 CDDA sectors
File Size:	83.3k
Bit Rate:	256k
Sample Encoding:	16-bit Signed Integer PCM

```
ffmpeg -i maskoff_sentence.mp3 -ar 16000 -ac 1 -acodec pcm_s16le \
maskoff_sentence.wav
soxi maskon_tone.wav
```

Input File:	'maskoff_sentence.wav'
Channels:	1
Sample Rate:	16000
Precision:	16-bit
Duration:	00:00:02.60 = 41610 samples $\sim$ 195.047 CDDA sectors
File Size:	83.3k
Bit Rate:	256k
Sample Encoding:	16-bit Signed Integer PCM

4 Úloha 3

Vybral som druhú sekundu z nahrávky `maskon`, pretože mi najviac sedela. Potom som si vybral druhú sekundu z nahrávky `maskoff`, pretože mi podla oka prišla najviac podobná. No to sa na základe 4. úlohy samozrejme zmenilo. Signály som podla vzorcov v zadaní ustrednil a znormalizoval do dynamického rozsahu. Vypočítal som si potrebné údaje, ktoré som potreboval pre implementáciu rámcov aké sú dĺžka prekrytia rámcov, počet vzorkov v jednom rámci a počet rámcov v jednej sekunde nahrávky. Po implementácii rámcov viz. súbor `moj_projekt.py`, som sa mal rozhodnúť čo spravím s posledným rámcom, ktorý je polovičnej veľkosti. Rozhodol som sa že budem pracovať s 99 rámcami miesto 100, takže som posledný rámec zahodil. Na prvom obrázku môžme vidieť prvý rámec nahrávky `maskon`. Následne som vybral ručne jemu podobný tretí rámec z nahrávky `maskoff`.

4.1 Použité vzorce:

Vzorec pre výpočet vzorkov v jednom rámci. Kde `n_v` je počet vzorkov v jednom rámci, `t` je dĺžka jedného rámca v sekundách a `Fs` je vzorkovacia frekvencia.

$$n_v = t * Fs$$

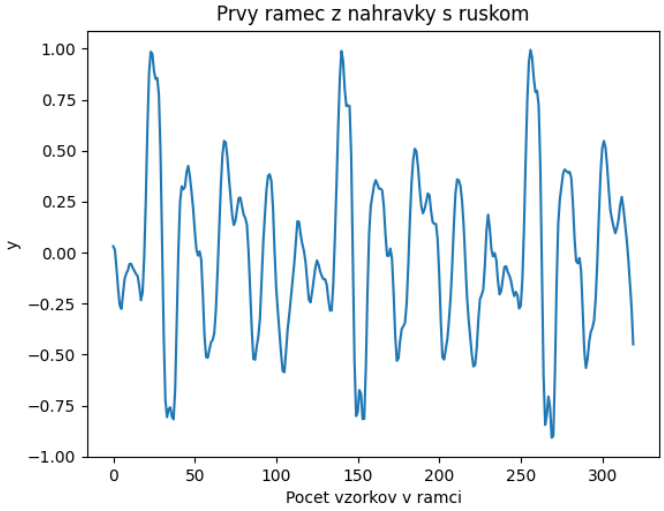
Vzorec pre výpočet dĺžky prekrytia rámcov vo vzorkoch. Kde `n_pv` je počet prekrytých vzorkov v jednom rámci, `t` je dĺžka prekrytia jedného rámca v sekundách a `Fs` je vzorkovacia frekvencia.

$$n_{pv} = t * Fs$$

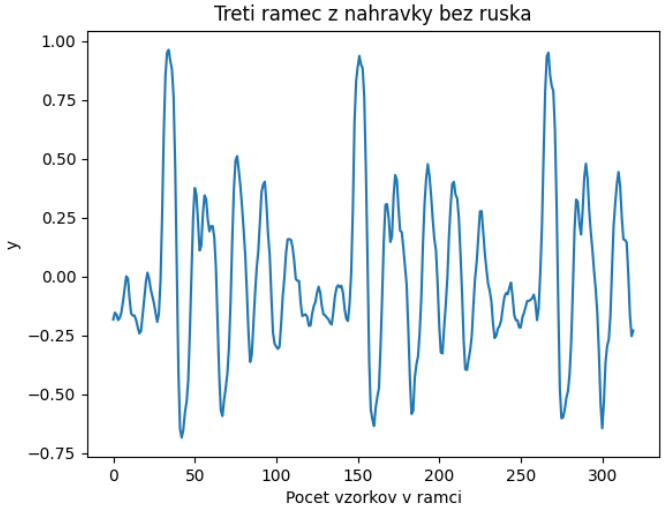
Vzorec pre výpočet počtu rámcov. Kde `Fs` je vzorkovacia frekvencia a `n_pv` počet prekrytých vzorkov.

$$n = \frac{Fs}{n_{pv}}$$

4.2 Obrázky rámcov:



Obrázek 1: Obrázok s maskou

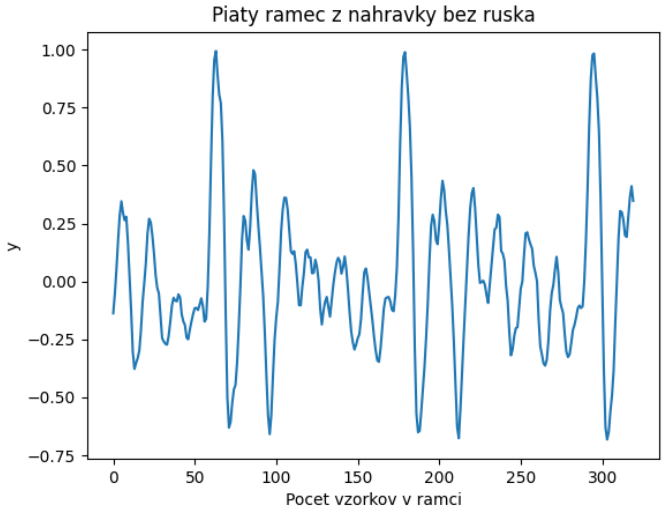


5 Uloha 4

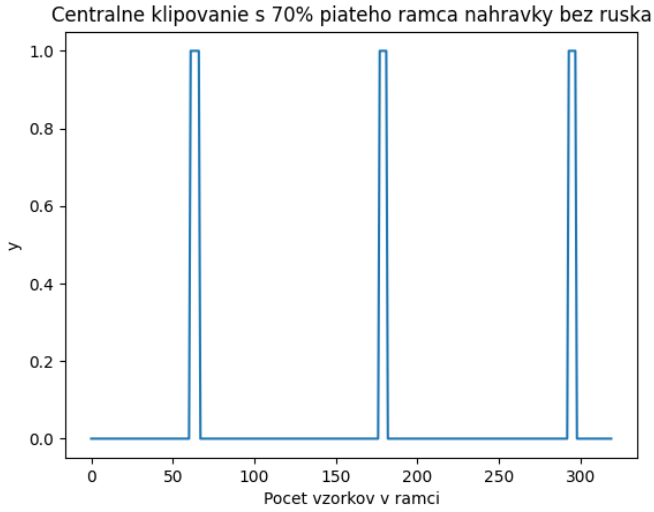
Riešenie štvrtej úlohy som začal tým že som na získané rámce aplikoval metódu centrálneho klipovania. To znamená že som pre individuálne rámce stanovil kladné a záporné absolútne maxima, a tak som hodnoty previedol na -1, 0 a 1. Následne som implementoval vlastnú autokorelaciu, ktorá násobí rámec sám so sebou a zároveň jeho násobiteľ posúva v jednotlivých krokoch. Vynásobené hodnoty v jednotlivých krokoch som sčítal a získal hodnoty autokorelacie funkcie. Pretože boli prvé hodnoty vyššie tak som si zvolil prah 500Hz, ktorý je uvedený v zadaní. Indexy maxím v jednotlivých rámcoch autokorelacie som previedol na frekvenciu a tak zostrojil základne frekvenciu rámcov. Frekvencie som porovnal pozorovaním a zistil som že mám obrovské rozdiely vo frekvenciách. Preto som opakoval nahrávanie až pokiaľ som nedošiel k slušnej podobnosti frekvencii. Takisto som tieto frekvencie overil pomocou strednej hodnoty a rozptylu, kedy hodnota strednej hodnoty je **138.1494** a rozptylu **1.0835**. Stredna hodnota nahravky maskon je 138.1184 s rozptylom 0.959. Stredná hodnota nahrávky maskoff je 138.1804 s rozptylom 1.206.Mal som možnosť sledovať správanie sa lagu, zistil som že lag je nepriamo úmerný vzorkovacej frekvencii.

V mojom prípade ak sa lag zvýši o 1 tak zmenší moju základnú frekvenciu rámca približne o 1,2Hz a naopak ak sa zníži tak zvýši frekvenciu približne o 1,2Hz. V prípade že by som chcel znížiť veľkosť zmeny `f0` je možné posunúť hranicu prahu tak že eliminujeme omnoho viac rušivých zvukov z okolia. Takže by sme dostali čistí hlas, ktorý by nemal ideálne žiadnu zmenu, v prípade že zoberieme nahrávku s rúškom ktorú sme nahrali s rovnakým tónom. V mojom prípade by ju šlo zvýšiť napríklad na 600Hz, alebo upraviť hranicu klipovania. Taktiež by sa dal nahrať kvalitnejší ton, na ktorý mohlo v momente nahrávanie vplývať rôzne veci ako únava, melódia a stres. Všetky tieto veci mohli ovplivniť zakladnu frekvenciu. Riešením by teda mohlo byť odstránením čo najviac týchto faktorov a získať čo najlepšiu základnú frekvenciu.

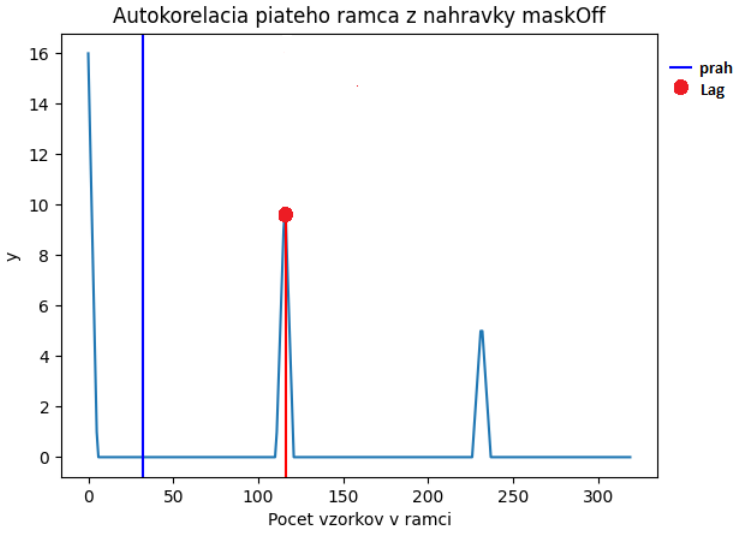
5.1 Grafy :



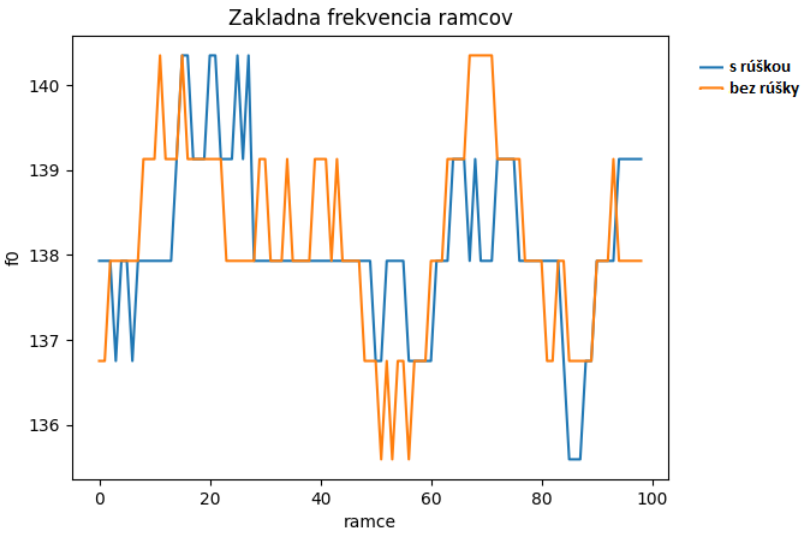
Obrázek 2: Obrazok ramca bez masky



Obrázek 3: Sklipovaný rámec bez rusky



Obrázek 4: Autokorelacia rámca bez rusky



Obrázek 5: Základná frekvencia rámcov

6 Úloha 5

Implementácia nasledujúcich úloh sa nachádza v druhom súbore `moj_projekt5.py`. Vypracovanie tejto úlohy som začal tým že som použil knižnicovu funkciu fft, ktorá mi načrtla ako má vyzerať môj spektrogram. Následne som sa vrhol na implementáciu mojej vlastnej funkcie DFT. Po jej implementácii som ju porovnal s knižnicovou implementáciou fft. Zistil som že najväčší rozdiel je v čase výpočtu. Moja DFT môže trvať až 15 minút narozdiel od fft, ktorá to ma spočítané do niekoľkých sekúnd. Po vizuálnej kontrole na oboch grafoch som nenašiel rozdiel medzi FFT a DFT.

Funkcia prijíma dvojrozmerné pole hodnôt, ktoré je rozšírené o nuly na požadovanú veľkosť 1024. Následne sa inicializuje lokálna premenná pre ukladanie súčtov a pole pre ukladanie logaritmického výkonu. Po ukončené najvnútornejšieho cyklu sa súčet uloží do pola hodnôt mojej DFT, ktoré sa následne vracia a ďalej vykresluje ako spektrogram s koeficientami 0..512, pretože druhá časť je k nej symetrická.

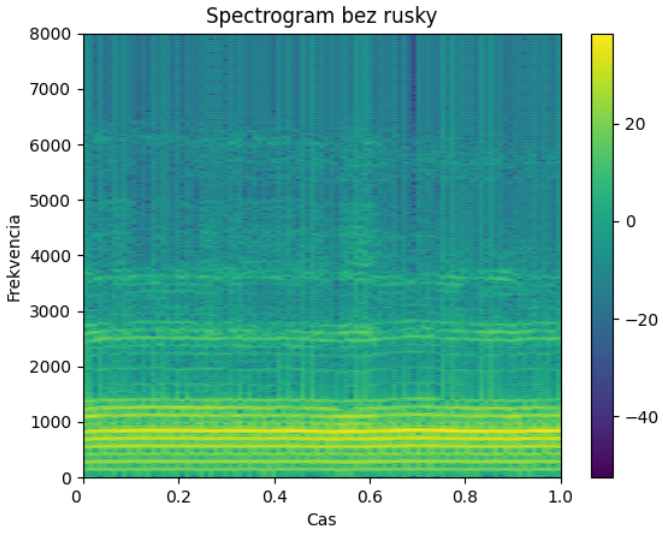
6.1 DFT funkcia:

Algoritmus 1: DFT

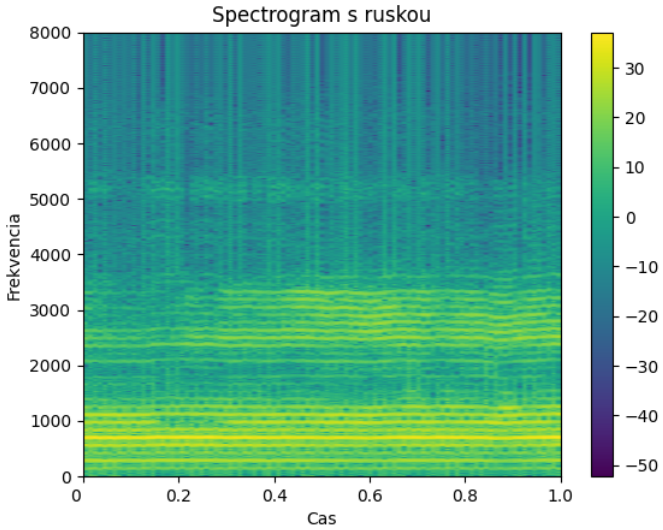
Input: X
Output: X_{PK}

1: $dftsum = 0j$
2: $resultOnFunc = np.ndarray((99, 1024), dtype = np.complex\_)$
3: **for** $i = 0$ to 99 **do**
4: **for** $f = 0$ to 1024 **do**
5: **for** $k = 0$ to 1024 **do**
6: $dftsum = dftsum + (x[i][k] * (np.cos((2 * np.pi * k * f)/1024) - j * np.sin((2 * np.pi * k * f)/1024)))$
7: **end for**
8: $resultOnFunc[i][f] = dftsum$
9: $dftsum = 0j$
10: **end for**
11: **end for**
12: **return** X_{PK}

6.2 Spektrogramy nahrávok:



Obrázek 6: Spektrogram s ruškou



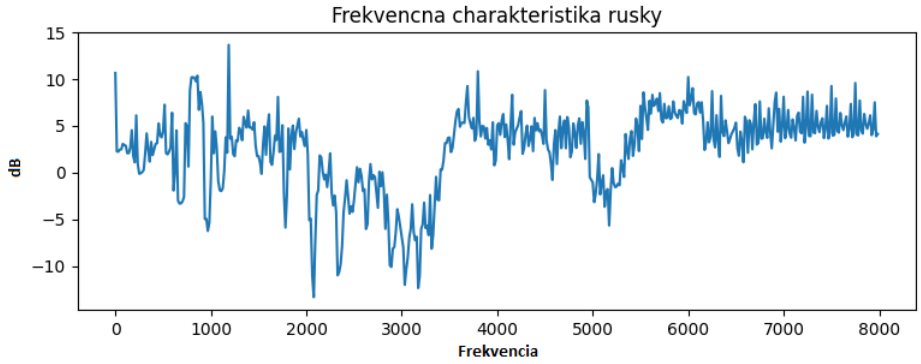
Obrázek 7: Spektrogram bez rušky

7 Úloha 6

Úlohu som začal riešiť tým že som si najprv našiel vzorec pre frekvenčnú odozvu. Frekvenčná odozva je daná podielom výstupného signálu a vstupného signálu. Keďže pracujeme s „rúškovým filtrom“ tak náš výstupný signál je furierova transformácia nahrávky s rúškom, a vstupný signál je furierova transformácia nahrávky bez rúška. Výsledok som spriemeroval a previedol na výkonové spektrum, ktoré som vykreslil.

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

7.1 Frekvenčná charakteristika:



Obrázek 8: Frekvenčná charakteristika

Môžeme spozorovať že náš „rúškový filter“ je značné zošumený. Preto je potrebné s ním ďalej pracovať a skvalitniť ho, pokiaľ chceme získať slušný výsledok. Môj filter sa najviac podobná na filter s dolno pásmovou priepustnosťou. Môžme vidieť že okolí frekvencie 3000 sa aj tak správa. Filter v tomto okolí zvýrazňuje určité frekvencie. Mimo tohoto okolia filter frekvencie potláča a zavádza určitý šum, ktorý sa prejaví vo výslednej nahrávke. Taktiež sa snaží mierne prepúšťať frekvencie v okolí 5000Hz bohužiaľ už nie moc efektívne. Filter je diskrétny a pracuje s konečnou impulznou odozvou.

8 Úloha 7

Z predchádzajúcej úlohy som získal frekvenčnú charakteristiku, ktorú som použil pri tejto úlohe. Previedol som ju pomocou IDFT na impulznú odozvu mojej rúšky.

IDFT funkcia prijíma frekvenčnú charakteristiku, ktorú prevádza na impulznú odozvu. Moja implementácia mi vyšla rovnako s knižnicovou implementáciou. Jediny rozdiel je čas výpočtu. Implementoval som podla vzorca kde sa výsledna suma násobit 1/N.

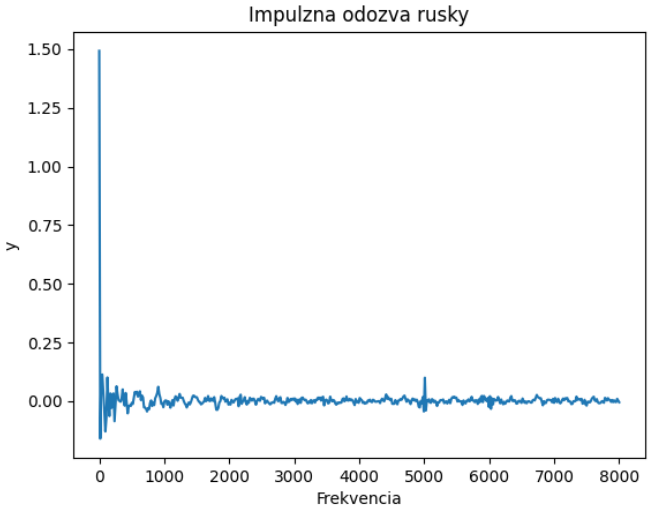
8.1 IDFT funkcia:

Algoritmus 2: IDFT

Input: X
Output: X_{IDFT}

1: $idftsum = 0j$
2: $resultOnFunc = np.ndarray((1024), dtype = np.complex\_)$
3: **for** $k = 0$ to 1024 **do**
4: **for** $f = 0$ to 1024 **do**
5: $idftsum = idftsum + (x[f] * (np.cos((2 * np.pi * k * f)/1024) + j * np.sin((2 * np.pi * k * f)/1024)))$
6: **end for**
7: $resultOnFunc[k] = 1/1048 * idftsum$
8: $dftsum = 0j$
9: **end for**
10: **return** X_{IDFT}

8.2 Impulzna odozva:



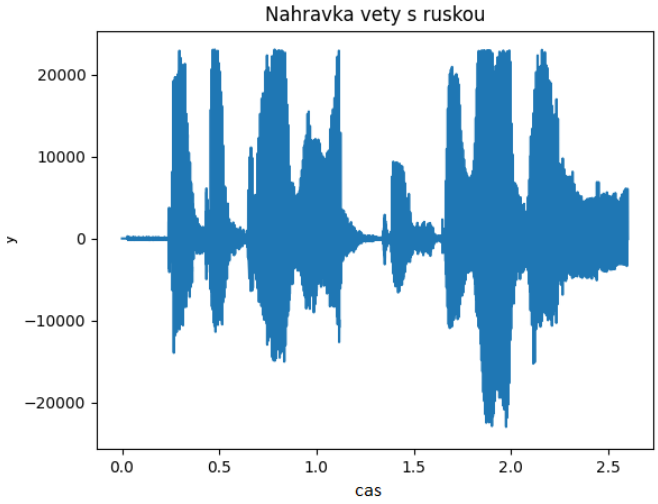
Obrázek 9: Impulzna odozva rusky

9 Úloha 8

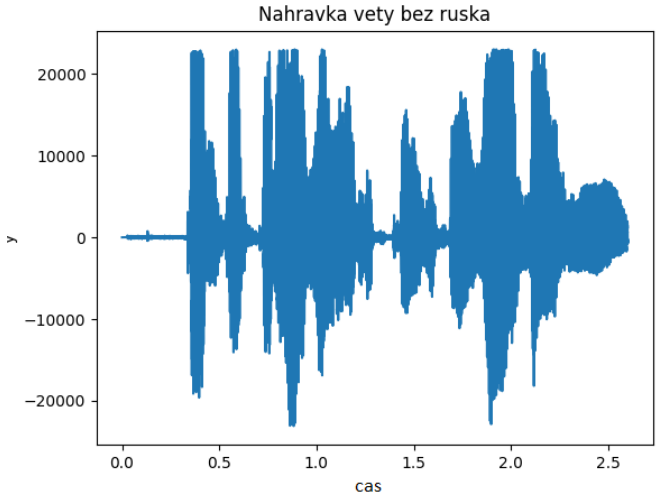
Koeficienty z masky, ktoré som získal vďaka predchádzajúcej úlohe som vložil do knižnicovej funkcie `scipy.signal.lfilter` spolu s koeficientom a, ktorý je 1 a nahrávkou. Mal som možnosť pozorovať nahrávku s rúškou a so simulovanou rúškou. Najväčší rozdiel, ktorý sa dá pozorovať je v hlasitosti nahrávky a jej zrozumitelnosti, ktorá klesá. V grafe sa dá vidieť že najväčšie rozdiely nastávajú vtedy keď dostatočne dlho držime nejaký ton alebo zvýšime hlas. Preto môžme vidieť že najväčšía podoba simulovanej rúšky a realnej rúšky je tam kde je hlas najslabší, alebo kde je hlasový výkyv najmenší. Pokiaľ by sme teda chceli aby boli

rozdiely medzi reálnou a simulovanou rúškou najmenšie potrebovali by sme mať nahrávku v rozumnej hlasitosti a čo najviac zrozumiteľnú. Samozrejme by aj pomohlo to že by sme odstránili čo najviac okolitého šumu, ktorý by spresnil náš odhad simulovanej rúšky. Poprípade by sme mohli použiť lepšie techniky a algoritmi pri odhade rúškového filtru.

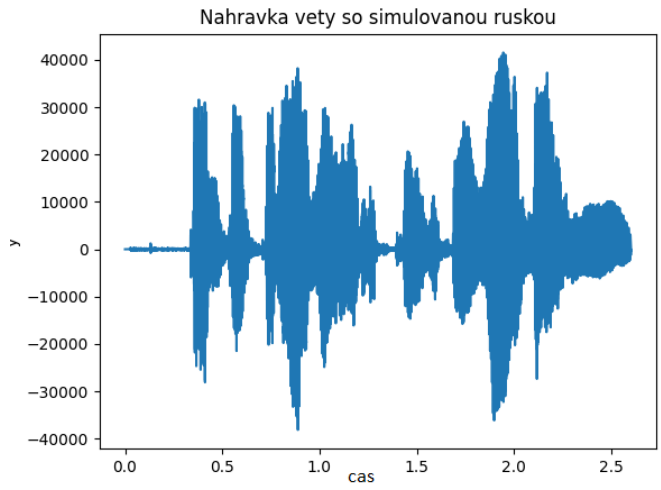
9.1 Grafy:



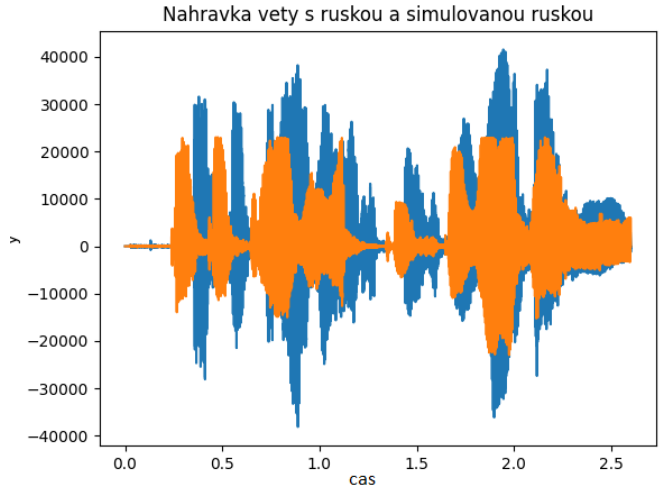
Obrázek 10: Nahravka s ruskou



Obrázek 11: Nahravka bez rusky



Obrázek 12: Nahravka s simulovanou ruskou



Obrázek 13: Porovnanie nahravky s ruskou a simolovanym ruskom

10 Úloha 9

Spracovávanie signálu určite nie je najľachšie, je potrebné vedieť množstvo algoritmov. Tieto algoritmy sa aplikujú na našu nahrávku, ktorá nie je ideálna a aj keď som sa snažil byť v čo najtichšom prostredí a držať čo najpodobnejší ton v nahrávkach. Bohužiaľ nie som trénovaný človek a ani stroj, takže tón v nahrávkach nie je ideálny, ale postačil pre slušný odhad simulovanej rúšky. Samozrejme že ďalšie chyby mohli nastať v rôznych algoritmoch, ktoré nie sú tie najrýchlejšie a najlepšie implementované. Nepracujem s najvyššou precíznosťou, a popritom ako sa v signáloch pracuje s dosť malými číslami, tak mohlo dojsť k chybám. Celkovo si myslím že na to že som sa dopracoval k slušnému riešeniu za ktoré sa nemusím hanbiť. To to bolo moje prvé stretnutie so signálmi na programovej úrovni, kedy som ich mohol sám spracovávať alebo tvoriť. Určite v budúcnosti keď sa stretnem so signálmi chcel by som použiť kvalitnejšie prostriedky pri nahrávaní a v lepšom prostredí ako doma. Mohol by som sa teda „odpichnúť“ od čo najkvalitnejšej nahrávky a zvoliť k nej presnejšie algoritmy a postupy.

Po spracovaní rozšírení som sa oboznámil s algoritmami, ktoré dokážu zdokonaliť odhad rúškového filtra. Taktiež som si vyskúšal detekovanie rôznych chýb nahrávok a ich riešenie. Mal som si možnosť overiť že základné riešenie ani zďaleka nedokáže presne odhadnúť správny rúškový filter. Aplikovanie týchto rozšírení mi pomohlo zdokonaliť môj filter a spraviť moje riešenie omnoho kvalitnejšie a menej nachylné na rečové chyby. Zistil som že najviac sa dokážem priblížiť k presnému rúškovému filtru pomocou aplikácie rôznych okienkových funkcii.

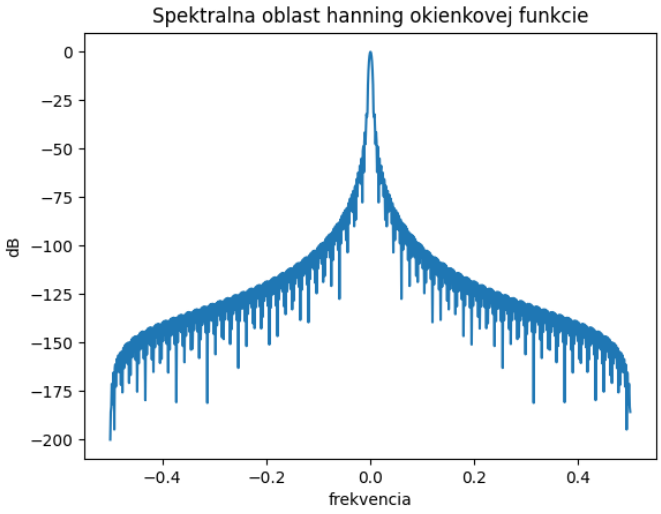
Počas riešenia projektu som sa oboznámil s množstvom techník a algoritmov, ktoré sa aplikujú pri spracovaní rečových signálov. Odskúšal som si v praxi ako sa vytvárajú a funguju filtre. Mal som možnosť spoznáť viac svôj hlas a ako sa správa počas toho ako nie je tlmený rúškou a naopak. Zistil som že moja rúška napríklad nema skoro žiaden vplyv na môj tón, a ak nejaký tak veľmi minimálny. Naopak som zistil že najväršší vplyv na môj tón má spôsob akým sa vyjadruhjem a nálada v akej sa nachádzam. Vyskúšal som si ako sa aplikujú rôzne filtre na reč a v akej rôznej kvalite sa môžu nachádzať. Oboznámil s programovacím jazykom python, ktorý bude určite užitočný v mojej profesionálnej kariere.

11 Úloha 11

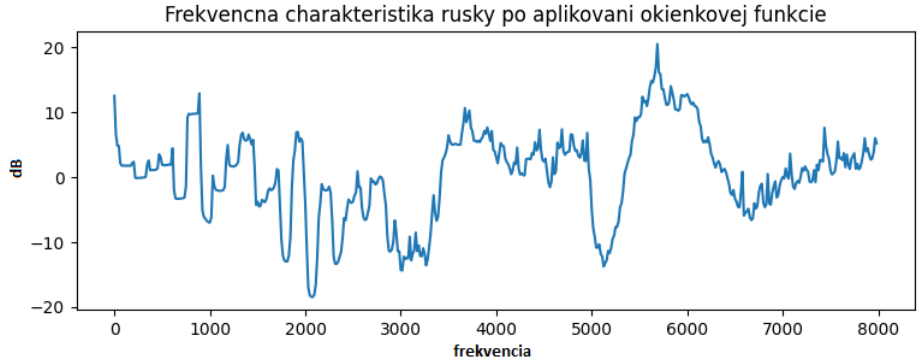
Implementácia tejto doplňujúcej úlohy sa nachádza v súbore `moj_projekt11.py`. Vo svojom projekte som použil hanning okienkovú funkciu. Skúsil som rôzne okienkové funkcie, ale táto mi prišla že dokázala mi najviac pomôcť sa priblížiť k reálnemu rúškovému filtru. Rozhodoval som sa najviac medzi Hamming a Hann okienkovými funkciami, pretože sú najuniverzálnejšie. Vybral som si ale Hann okienkovú funkciu vďaka ktorej sa mi darí úspešne potláčať vzdialenejšie produkty a je najvhodnejšia v množstve prípadov. Hlavná výhoda je že má dobrú frekvenčnú presnosť.

11.1 Grafy:

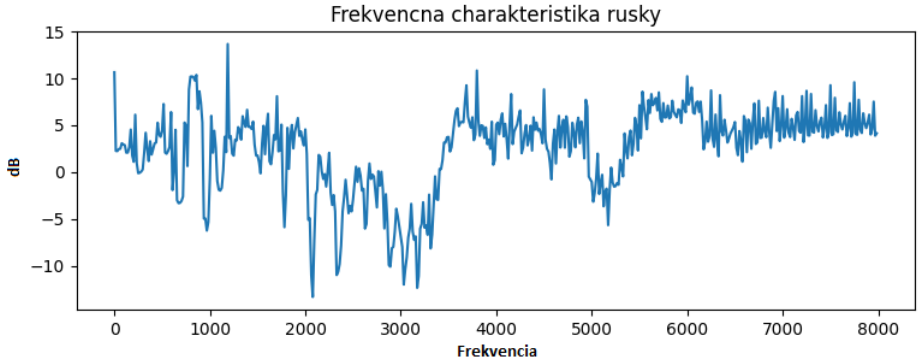
Obrázek 14: Hanning funkcia v case jedneho rámca



Obrázek 15: Hanning v spektralnej oblasti



Obrázek 16: Aplikovana hanning



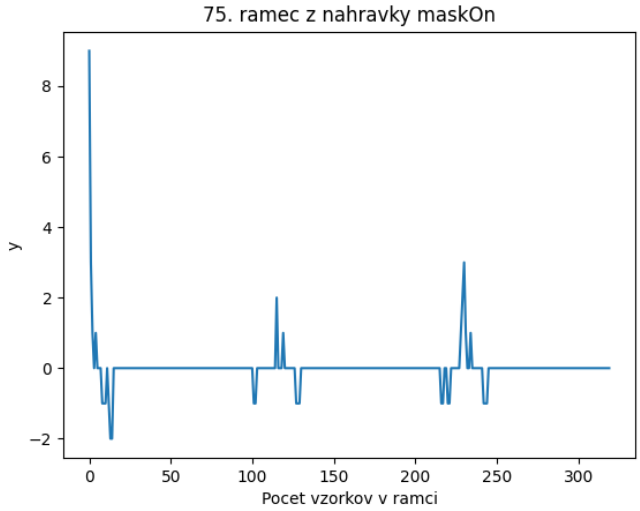
Obrázek 17: Pred aplikaciou hanning funkcie

Hanning funkcia mi pomohla presnejšie odhadnúť frekvencie, potlačiť vzdialené produkty a zbaviť ma falošných vysokofrekvenčných produktov. Taktiež mi pomohla zmenšiť amptitúdy v koncových miestach signálu kde vznikali nepresnosti. Spozoroval som že sa odstránila veľká časť zošumenia filtru.

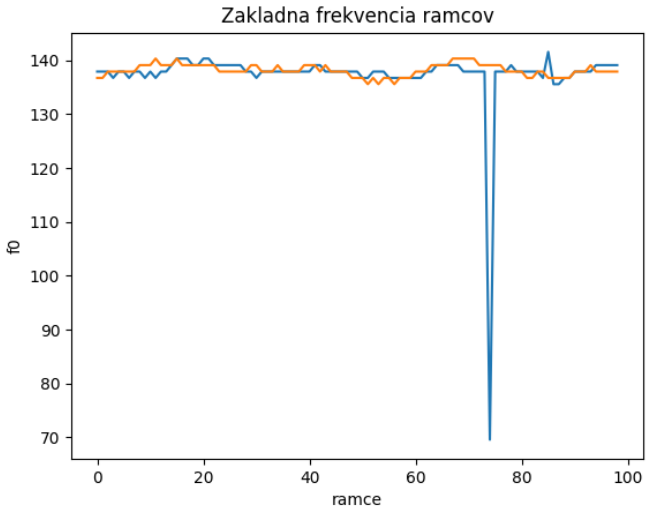
12 Úloha 12

Implementácia sa nachádza v súbore `moj_projekt12.py`. Popri práci so základním riešenim som našťastie neobjavil dvojnásobný lag, preto som sa potreboval pohrať s prahom a úrovňou klipovania. Potom ako som som sa dostal na úroveň klipu s 86,5% sa mi podarilo naraziť na tento problém. Nachádza sa na 75. rámci nahrávky s rúškom a znižuje základnú frekvenciu približne o polovicu.

12.1 Grafy:



Obrázek 18: Dvojnásobný lag na 75 rámci nahravky s ruskou



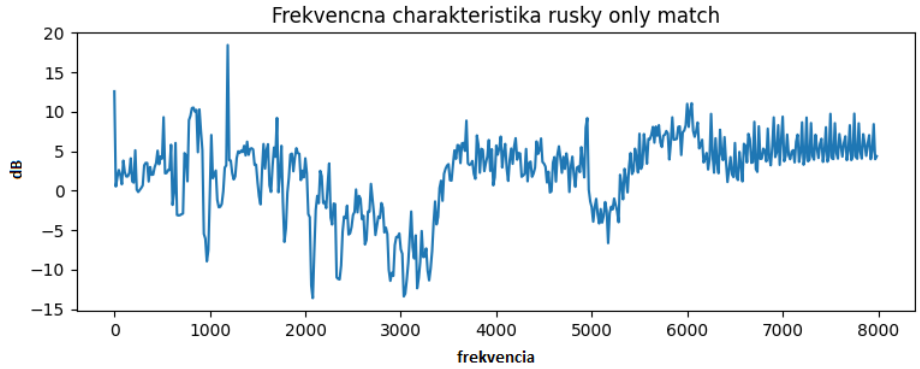
Obrázek 19: Základná frekvencia pri probléme dojnásobného lagu

Problém n-násobného lagu som vyriešil tak že som na hodnoty základnej frekvencie aplikoval nelineárny medianovy filter. N-násobný lag môže znamenať niekoľko násobné zníženie základnej frekvencie. Rozhodol som sa ho detekovať pri tretinovom výkyve od strednej hodnoty základnej frekvencie. Túto hodnotu n-násobného lagu nahradím v základnej frekvencii medianom zo základnej frekvencie.

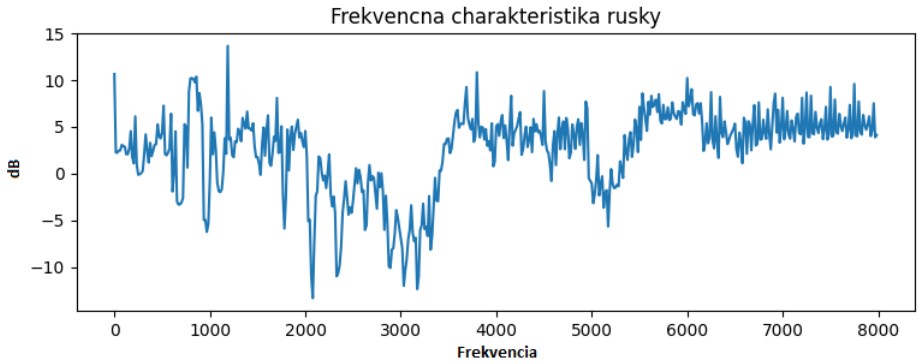
13 Úloha 13

Implementácia sa nachádza v súbore `moj_projekt13.py`. Môžme sledovať určité zlepšenie charakteristiky. Bohužial táto zmena nie je určite obrovská, v porovnaní s okienkovými funkciami takmer žiadna. Urcie nam pomohla vyhladit charakteristiku a aj pomohla spresnit odhad ruskoveho filtra. Táto metóda sa preukázala ako účinná, ale určitom malom rozmedzí.

13.1 Grafy:



Obrázek 20: Po aplikacii only match funkcie

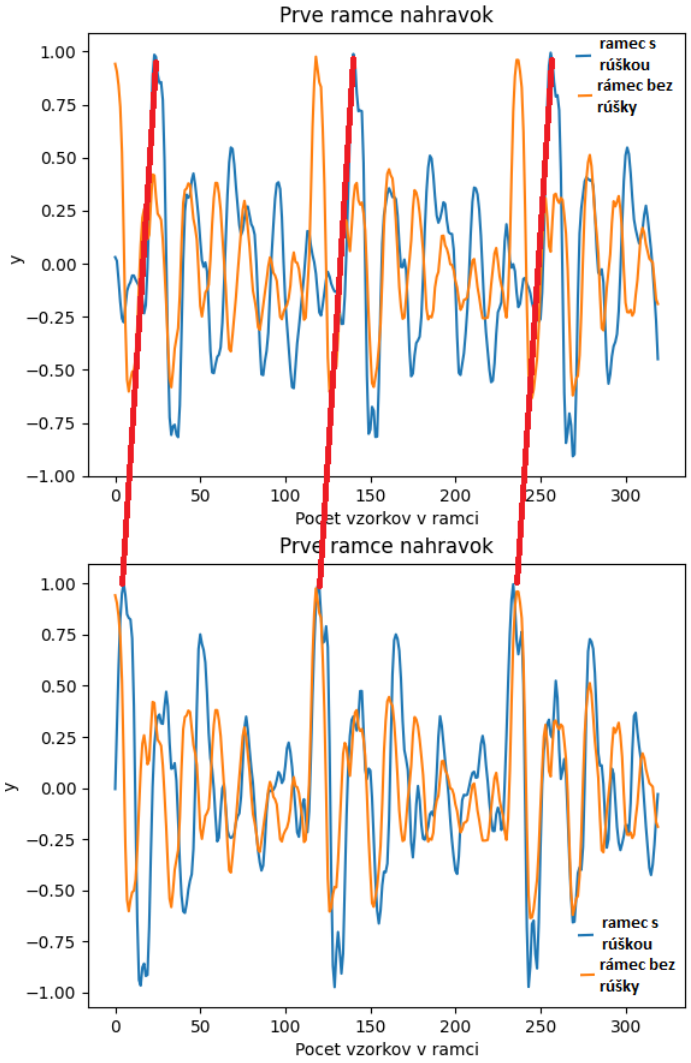


Obrázek 21: Pred aplikaciou only match funkcie

14 Úloha 15

Implementácia sa nachádza v súbore `moj_projekt15.py`. Vybral som si sekundu na ktorej som pracoval pri základnom riešení. Vytvoril som rámce pre obe nahrávky s dĺžkou 25ms. Následne som rámce sklipoval a podla zadania previedol korelaciu medzi rámcami s rúškou, bez nej a naopak. Jednotlivé fázové posuvy som si uložil do pola hodnôt. Rámce som prešiel cyklom a podla správnej hodnoty fázového posuvu som sa rozhodol, ktorý rámec posuniem. Rámec posuniem o fázový posuv vpred tým že jeho orežem jeho začiatok o hodnotu fázového posuvu. Tento rámec zároveň skrátim tak aby sa mi zaroval k odpovedajucej dĺžke 20ms rámca. Ak by som mal orezať z druhého rámca koniec, tak ten taktiež rovno urovnám na 20ms dĺžku rámca. V prípade že mám hodnotu fázového posunu korelacie nerušky a rúšky 22 vo vzorkoch a dĺžka rámca je 400 vzorkov. Rozhodnem sa že z nerúškového rámca orežem 22 vzorkov a rovno ho zarovnám na 20ms rámec tým že vyberiem hodnoty z indexu 22 až 342. Z rúškového rámca by som mal orezať koniec na rámec s indexmi 0 až 378, keď že potrebujem rovnakú dlžku rámca tak ho rovno zarovnám na hodnoty indexu 0 až 320.

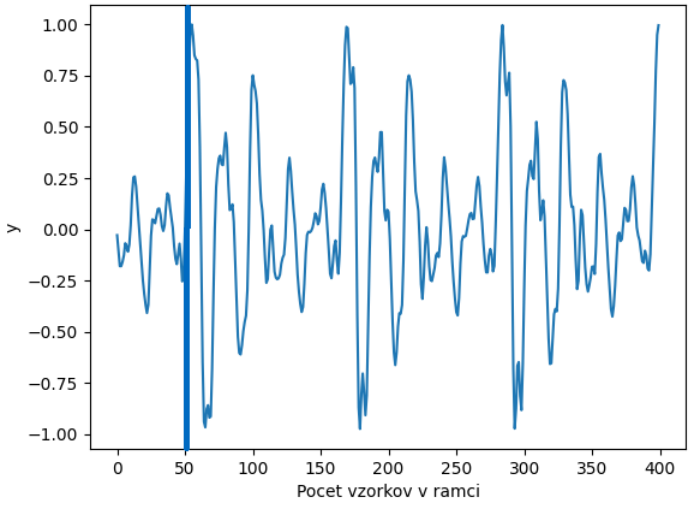
14.1 Grafy:



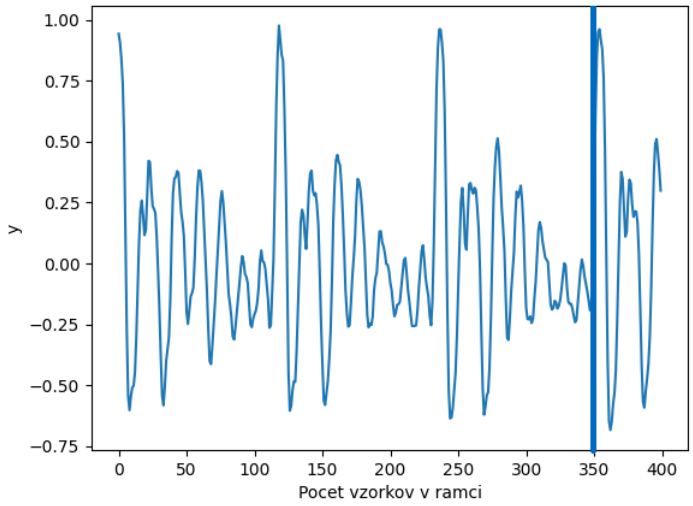
Obrázek 22: Graf zarovnania dvoch rámcov

Vybral som si priebeh fázoveho posunu medzi prvy ramcom s rúškou a bez. Bol vybraty správny fázovy posun, pretože väčší mal hodnotu 60 vzoriek a menší 50. Môžeme vidieť že ramec s rúškou sa posúva o 50

vzoriek z predu a ramec bez rúšky sa posuva o 50 z opacnej strany.

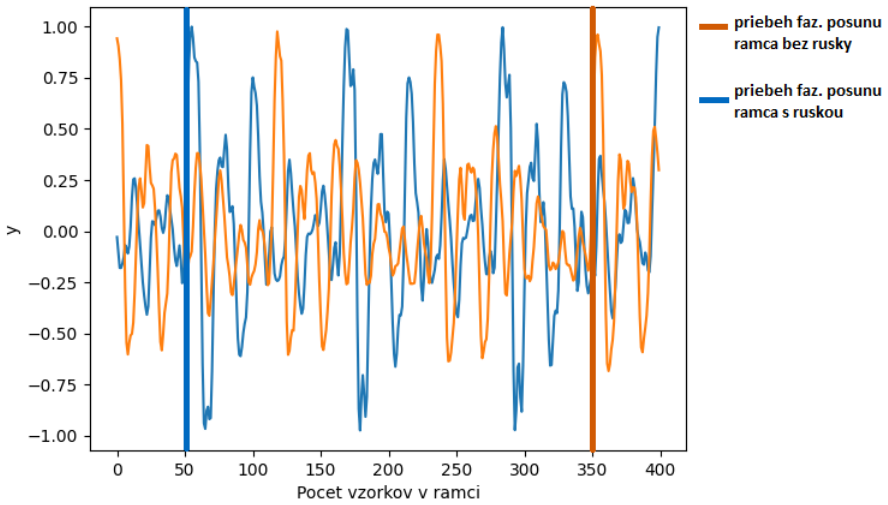


Obrázek 23: Fazovy priebeh posunu ramca s ruskou

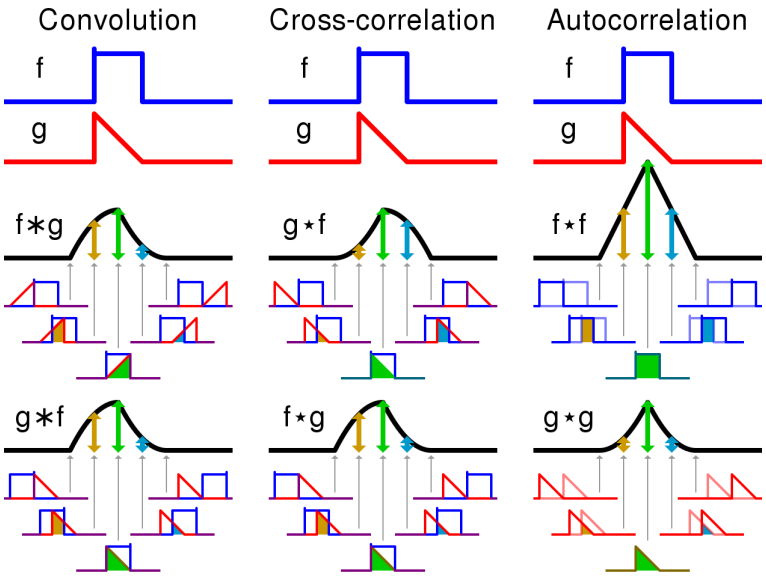


Obrázek 24: Fazovy priebeh posunu ramca bez rusky

Počas môjho riešenia som si všimol že ak sčítam fázové posuny tak získam hodnotu lagu daného rámca. Je to spôsobené tým že tieto vzájomné korelácie medzi rámcami rúšky a bez úsko súvisia. Korelácia je symetrická operácia a v prípade že by sme chceli získať hodnoty autokorelac0ie potrebovali by sme tieto hodnoty sčítať s určitým posuvom. Najlepšie je táto problematika znázornená na obrázku 27, ktorý som prevzal z wikipedie.



Obrázek 25: Priebeh fazoveho posunu medzi ramcami s ruskou a bez



Obrázek 26: Zmázornené korelácie

15 Zdroje

Wikipedia: Finite impulse response
https://en.wikipedia.org/wiki/Finite_impulse_response
Wikipedia: Discrete Fourier transform
https://en.wikipedia.org/wiki/Discrete_Fourier_transform
Wikipedia: Autocorrelation
https://en.wikipedia.org/wiki/Autocorrelation
Dokumentácia numpy, scipy, pyplot, copy
Tomáš Thúróczy: Měření rádiových parametrů a aplikace softwarově definovaného přijímače. Praha, 2016. Bakalarska práca. Fakulta elektrotechnická, ČVUT v Prahe. Vedúci práce Ing. Karel Ulovec, Ph.D.