Protokol k projektu v předmětu Signály a systémy

- Autor: David Hudák
- Předmět: ISS
- Login: xhudak03

Obsah

Úvod

Tato práce se zabývá protokolem k projektu v předmětu ISS. Jako rouška byl zvolen respirátor FFP2, který byl zrovna dostupný a použitý, tudíž nedošlo k vyplýtvání žádných ochranných pomůcek bez hygienického využití. Rád bych také zdůraznil, že při pořizování nahrávek nedošlo k ublížení na zdraví či psychické újmě žádného člověka či zvířete, jelikož nahrávky byly pořizovány v době, kdy nikdo nebyl v blízkém okolí, a v místnosti, kde byla pevně zavřená okna.

Použité nástroje

K vývoji byl použit programovací jazyk Python v prostředí Windows. Pro vývoj a spouštění jsem zvolil program PyCharm od společnosti JetBrains. Pro nahrávky byl použit program Audacity s použitím mikrofonu na standardní náhlavní soustavě. Pro správné a funkční spuštění programu je nutné natáhnout všechny audio soubory bez předpony sim do složky, ve které se nachází interpretovatelný soubor main.py, ten totiž předpokládá jejich přítomnost ve své složce.

Zdrojové kódy

Všechny zdrojové kódy, které sloužily ke generování grafů atp. k tomuto protokolu se nachází v souboru main.py. Pokud by někdo chtěl generovat nové grafy atp. je nutné některé části odkomentovat či zakomentovat dle aktuální potřeby.

Tabulka audio souborů

Soubor	Krátký popis
maskoffsentence.wav	Věta nahraná bez roušky
maskonsentence.wav	Věta nahraná s rouškou
maskofftone.wav	Tón nahraný bez roušky (á)
maskontone.wav	Tón nahraný s rouškou (á)
sim_maskon_sentence.wav	Simulovaná rouška na větě
sim_maskon_tone.wav	Simulovaná rouška na tónu
sim_maskon_sentence_only_match.wav	Simulovaná rouška na větě s odstraněním rámců
sim_maskon_tone_only_match.wav	Simulovaná rouška na tónu s odstraněním rámců
sim_maskon_sentence_window.wav	Simulovaná rouška na větě s okénkovou funkcí
sim_maskon_tone_window.wav	Simulovaná rouška na tónu s okénkovou funkcí

Úkol 1

Jak bylo zmíněno výše, pro nahrávky byl použit program Audacity a zapůjčená náhlavní soustava s mikrofonem. Nahrávky byly úmyslně udělány delší, aby se snadněji hledala sekvence stejných rámců v následujících úkolech. Obsahem nahrávek je samohláska á. Ze začátku jsou bohužel slyšet nádechy, ale je to především z toho důvodu, že první nahrávka byla nejlepší a u ostatních nahrávek se mi nepodařilo najít tak dobrou shodu v rámcích a tak dobře držet tón. Délka je odvozena dle .size vlastnosti pole získané nahráním souboru do pole s pomocí metody z knihovny soundfile read(). Délka v sekundách je pak zjišťována pomocí vztahu počet vzorků/vzorkovací frekvence (16000)

Jméno souboru	Popis	Délka [s]	Délka [1] (vzorky)
maskofftone.wav	Bez roušky	12.92	206720
maskontone.wav	S rouškou	22.885	366160

Úkol 2

Tato úloha byla řešena velmi podobně jako úloha výše. Výsledkem jsou zase dva soubory (viz tabulka). Oproti předchozí úloze, tyto nahrávky byly pořízeny zhruba o dva týdny později v jiném prostředí.

Jméno souboru	Popis	Délka [s]	Délka [1] (vzorky)
maskoffsentence.wav	Bez roušky	4.235	67760
maskonsentence.wav	S rouškou	4.655	74480

Úkol 3

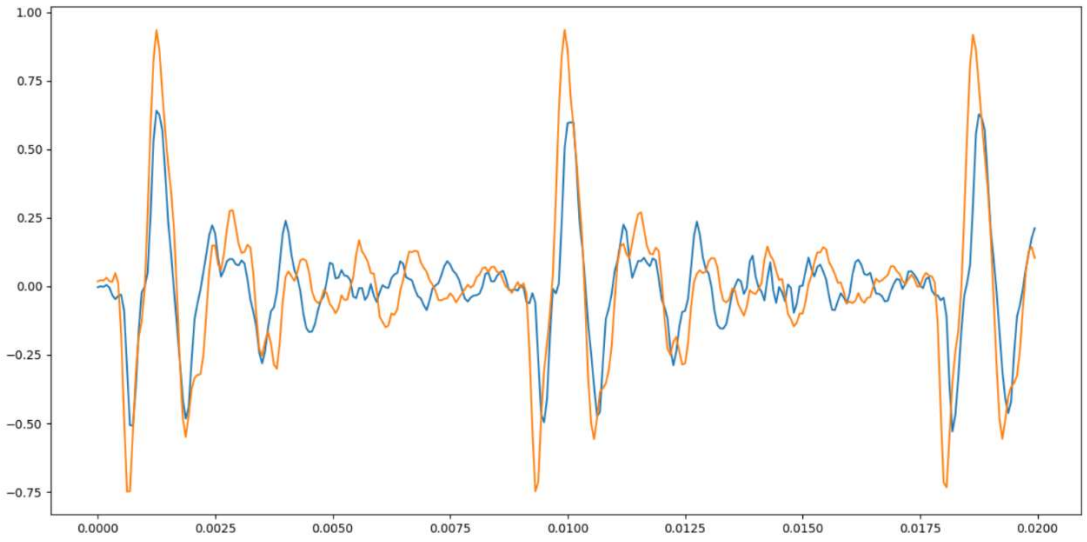
První částí této úlohy bylo hledání vhodného rámce. K tomu bylo použito poněkud nepraktická, přesto účinná metoda – ruční. Nejdříve jsem zvolil naprosto vizuálně vteřinu z každé nahrávky, která byla nahrána co nejvíce čistě (při nahrávce jsem měl problém s držením stability tónu, tudíž jsou nahrávky dlouhé). Tyto vteřiny pak byly dle vzorců ze zadání ustředněny. Následně jsem aplikoval výpočet rámců, který byl aplikován ve for cyklu s iterovanou proměnnou i. Vzorec pro výpočet velikosti dílčího rámce (nebo spíše jeho ohraničení v rámci pole) je v pythonovském kódu následující:

- int(i*fs*kolik2/2):int(fs*i*kolik2/2+kolik2*fs)
- Použití funkce přetypování v této úloze zařizuje, aby se správně indexovalo pole.
- Iterovaná proměnná i slouží k odlišení vzorku v rámci vteřiny.
- Fs (fs) je vzorkovací frekvence nahrávky, v tomto případě 16000.
- kolik2 je proměnná obsahující délku žádaného rámce (v základní úloze 0.02)
- Dvojtečka značí, odkud pokud se bere rámec (od levé části do pravé)

Co se týká přebývajícího rámce (posun byl 0.01 ms, při 100 rámcích poslední přesahuje oblast jedné vteřiny), ten byl prostě nevypočítáván, zahozen.

Po provedení výpočtu rámců jsem vždy tisknul první rámce a následně sledoval jejich vykreslené grafy. Grafy obou funkcí vypadaly vždy velmi podobně, jediným problémem byl nesprávný vzájemný posun. Proto jsem vždy ručně zvětšil/zmenšil hodnotu počátku výpočtu rámců a následně znovu kontroloval graf. Při tomto postupu se vyplatilo vždy zmenšit ruční posun na polovinu. Nakonec se povedlo velmi pěkně najít dva rámce, které si byly velmi podobné. Zpětně bych se ale asi přikláněl k implementaci nějakého algoritmu, který bude podobnosti (korelace) hledat sám.

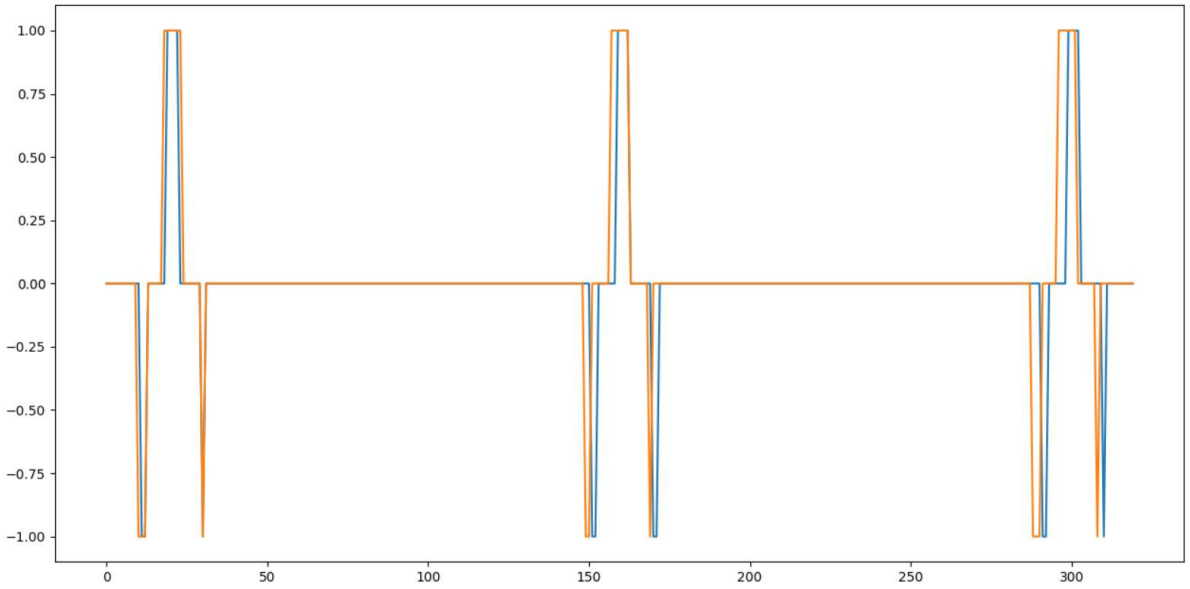
Na následujícím obrázku jsou vykresleny velmi podobné rámce.



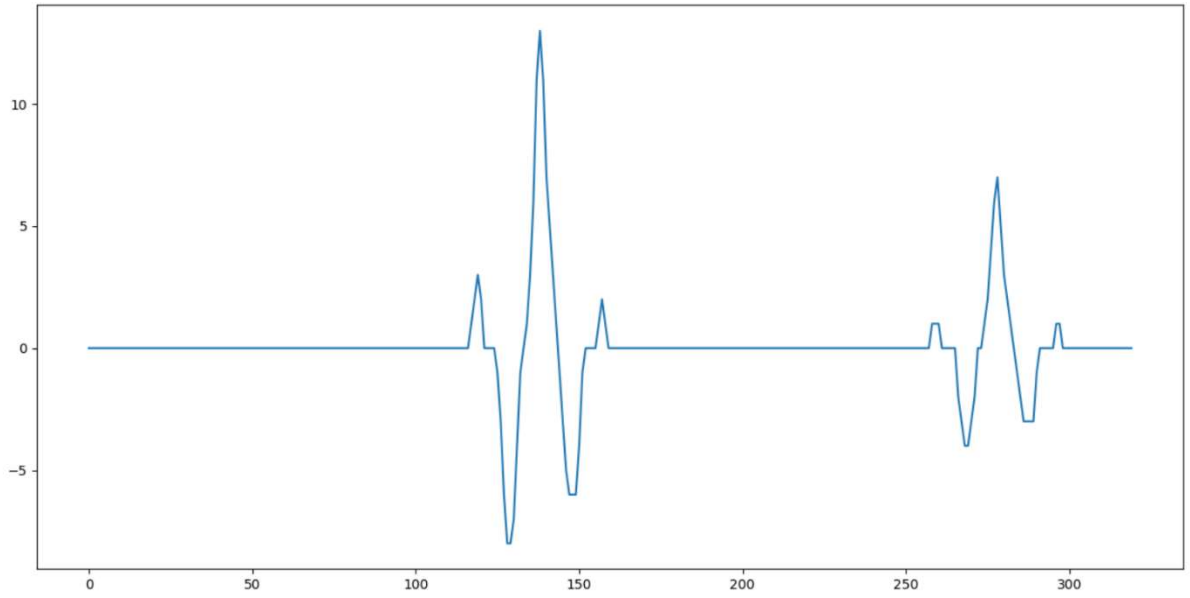
Obrázek 1 - dva velmi podobné rámce – modrá bez roušky, oranžová s

Úkol 4

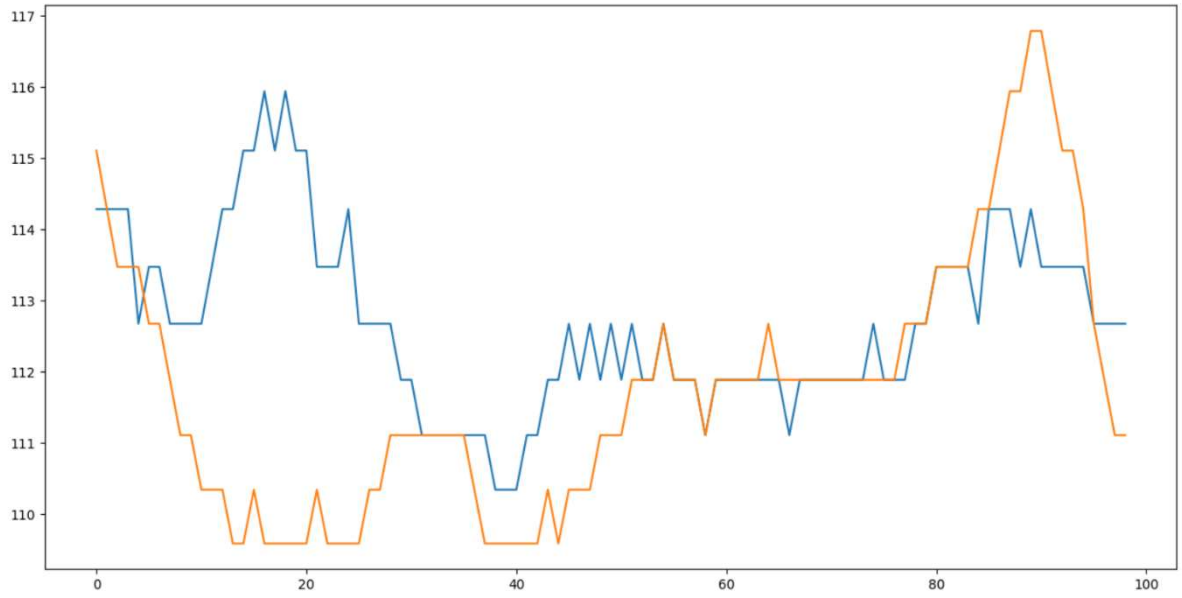
V předchozím úkolu byly nalezeny dva velmi podobné rámce a v této úloze byl proveden clipping, autokorelační analýza (práh byl zvolen od hodnoty 40, tedy frekvence nemůže nikdy přesáhnout 400 Hz). Na posledním grafu jsou zobrazeny zjištěné frekvence.



Obrázek 2- zclippované signály (bez roušky modře, s rouškou oranžově)



Obrázek 3 – výsledek dílčí autokorelace pro tón bez roušky



Obrázek 4 - frekvence bez roušky a s rouškou (rouška oranžově)

Po použití funkcí np.mean a np.var byly odhaleny následující hodnoty:

- Průměr a rozptyl bez roušky: 112.6975202980943, 1.5234747416073047
- Průměr a rozptyl s rouškou: 111.78215332011584, 3.1985816943643828

Odpověď na ošetření velkých rozdílů při rozdílech o 1 – možností je několik:

- Snížit počet vzorků pro výpočet lagu (třeba ze 320 na 160) – povede k menší přesnosti měření (bude hůře určená základní frekvence, ale současně bude mít lag menší tendence „přeskakovat“ na vzorky okolo).
- Zvýšit přesnost zvýšením vzorkovací frekvence a počtem vzorků – řešení bude přesnější, ale datově náročnější.
- Nějakým způsobem vázat lagy na sousedních rámcích (víme, že zvuk jen tak nepřeskakuje zničehonic, takže by se daly třeba nějakým způsobem váhově průměrovat atp.)

Úkol 5

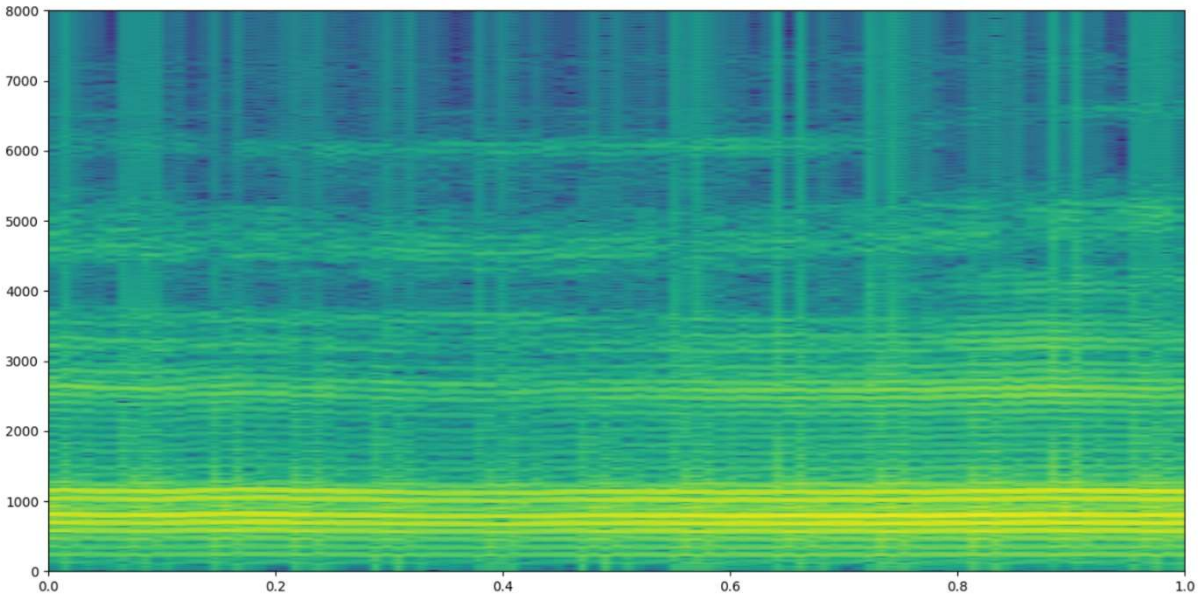
Zde je pythonovský kód implementující DFT:

```
zerosegment = np.append(segment, np.zeros(704))
for k in np.arange(1024):
    koef = 0
    for n in np.arange(1024):
        koef += zerosegment[n]*np.exp(-2j*pi*n*k/1024)
    res[k]=koef
```

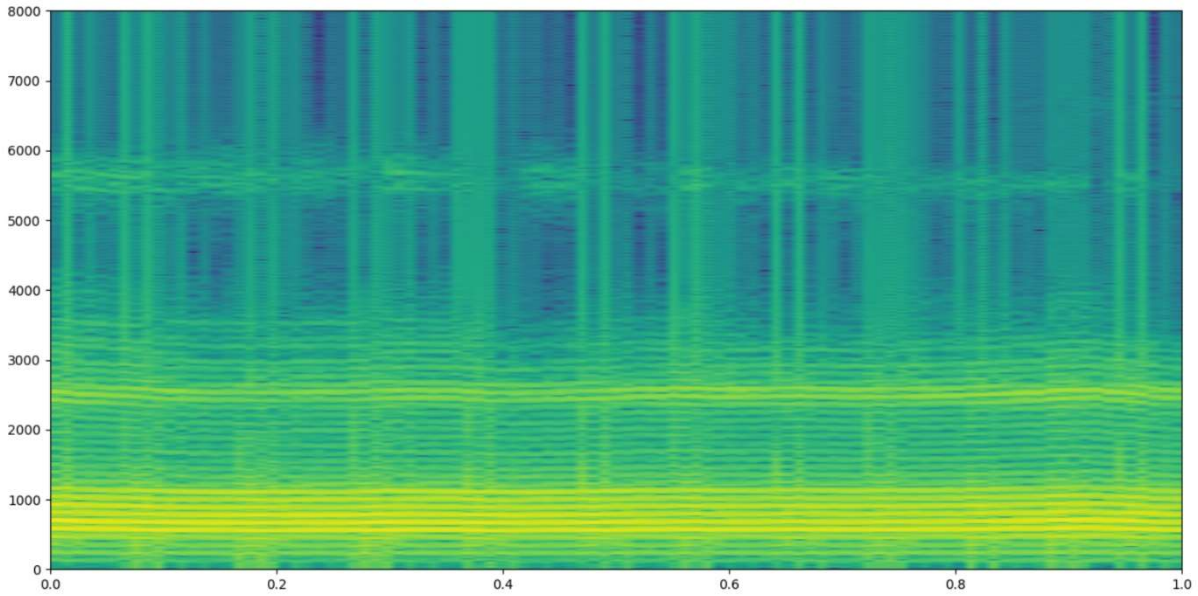
- segment obsahuje získané vzorky daného rámce (320 vzorků na rámec, k tomu je doplněno 704 nul)
- koef, tedy jednotlivý koeficent Fourierovy řady, je vypočítávaný dle vzorečku pro DFT (čerpáno ze slajdů):
  - $\sum_{n=0}^{N-1} x[n] e^{-j2\pi\frac{k}{N}n}$
  - Kde k je index koeficientu, x[n] je konkrétní vzorek z daného rámce, N počet vzorků a n je aktuální počítaný vzorek z rámce (k i n jdou od 0 do 1024).
- Pro výpočet všech rámců je nutné přidat cyklus procházející všechny rámce provádějící uvedený výpočet.

- Při srovnání s knihovní funkcí np.fft.fft došlo ke shodě hodnot, avšak také k zásadní neshodě u délky výpočtu. Osobně nedoporučuji tuto implementaci v kódu odkomentovávat.

Následují dva spektrogramy získané s použitím metody np.fft.fft() a tisknuté pomocí funkce imshow():



Obrázek 5 - spektrogram bez rouška



Obrázek 6 - spektrogram s rouškou

Úkol 6

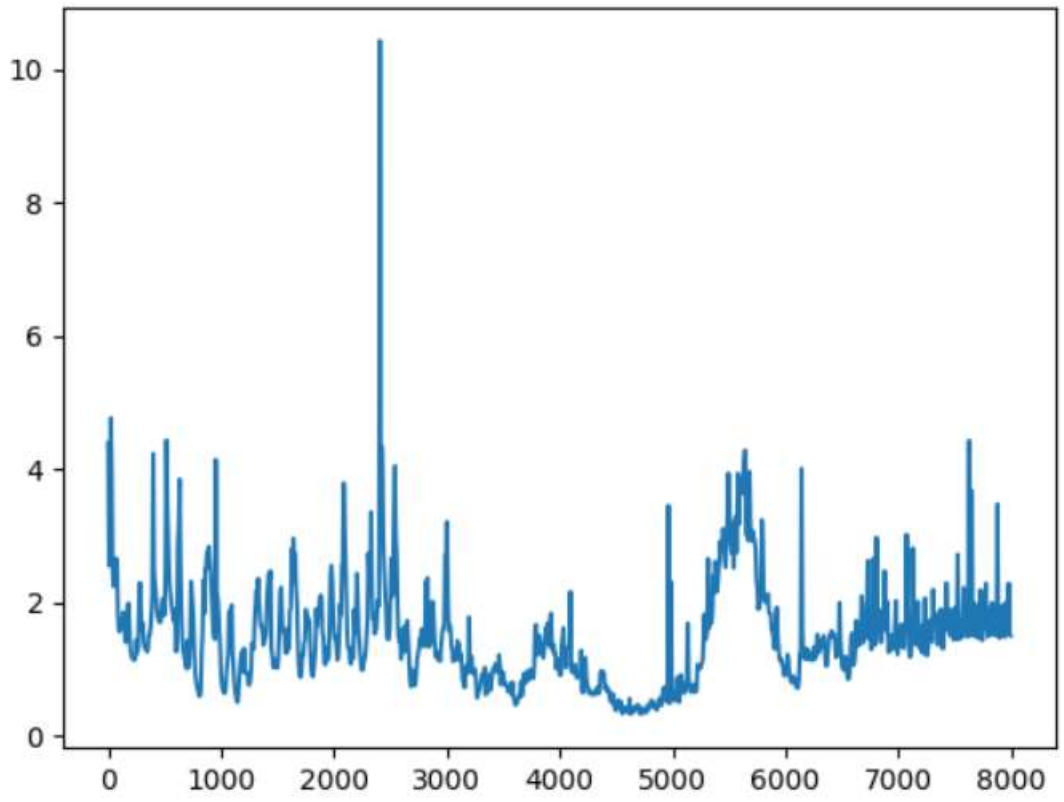
Výpočet frekvenční charakteristiky filtru se provádí jako podíl Fourierovy transformace nahrávky s rouškou a nahrávky bez roušky. Tato funkcionalita je ukryta i ve funkci freqz a v projektu je možné si vybrat, který výsledek je žádán (výpočet přes freqz ignoruje okénkovou funkci, zatímco výpočet přes podíl ji bere v potaz, pokud byla použita v implementace DFT).

Funkce freqz uvádí svou implementaci takto:

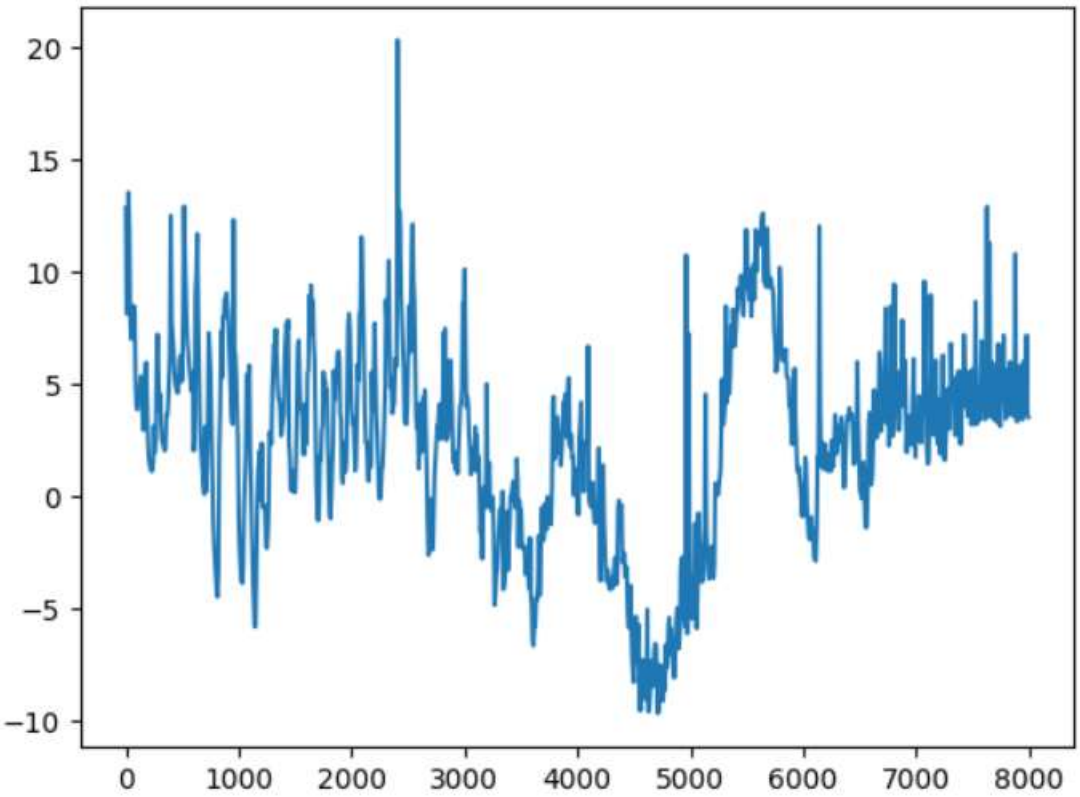
$$H(e^{jw}) = \frac{B(e^{jw})}{A(e^{jw})} = \frac{b[0] + b[1]e^{-jw} + \dots + b[M]e^{-jwM}}{a[0] + a[1]e^{-jw} + \dots + a[N]e^{-jwN}}$$

Tato implementace, jak si můžeme povšimnout, přímo odpovídá na otázku ze zadání.

Níže následuje frekvenční charakteristika roušky (modul a modul převedený na výkon dle stejného logaritmického vzorce, jako výsledek Fourierovy transformace v úkolu číslo 5).



Obrázek 7 - modul frekvenční charakteristiky roušky



Obrázek 8 - výkon roušky

Z grafů týkajících se filtru si můžeme povšimnout poměrně divokých grafů, jejichž význam ale v praxi není zas tak významný, jelikož nahrávka obsahuje hlavně nižší frekvence a u těch se průběh modulu odehrává kolem jedničky či u výkonu blízko 0. Vyšší frekvence pak vypadají poměrně divoce, protože ve vzorcích je k nim poměrně málo dat a i malá změna se tak projeví velmi silně (v nahrávce se klidně mohou z ničeho stát dvě nic). Mohlo by pak být zajímavé to pustit na zvuk, který obsahuje frekvence kolem 2200 Hz, jelikož u těch graf obsahuje špičku.

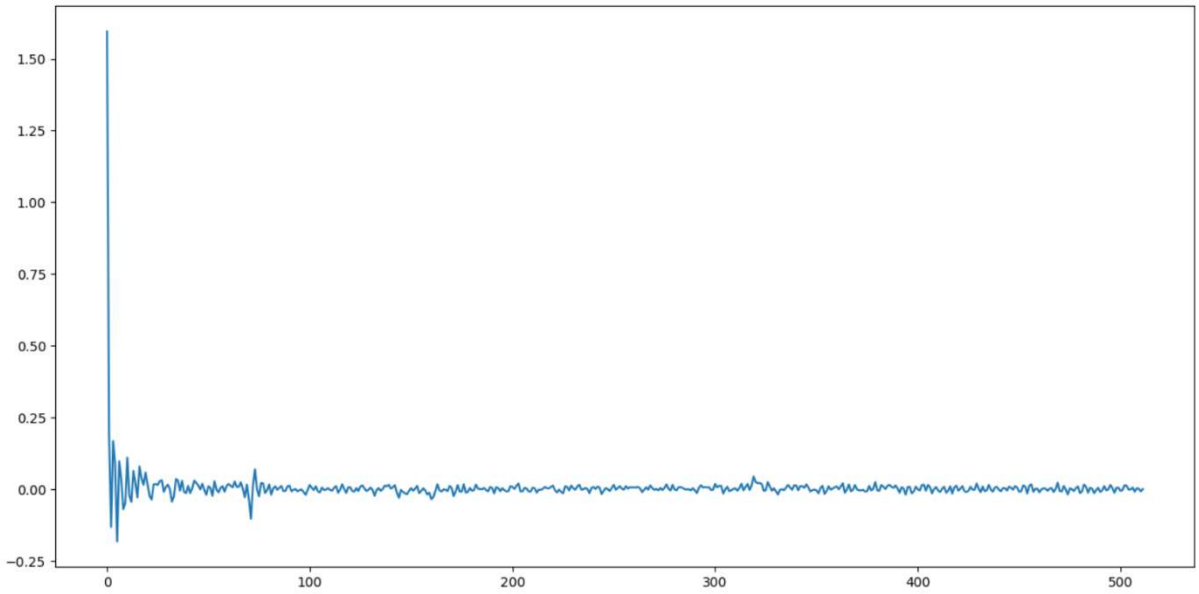
Úkol 7

Řešením tohoto úkolu je pouze implementace inverzní diskrétní Fourierovy transformace (IDFT), která je velmi podobná implementaci diskrétní Fourierovy transformace (DFT), tedy pythonovský kód:

```
xnarray = np.zeros(1024)
for n in np.arange(1024):
   for k in np.arange(1024):
       xnarray[n] += characteristic[k] * np.exp(+2j*pi*k/1024*n)
idftres = 1/1024 * xnarray
```

- $\frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{+j2\pi\frac{k}{N}n}$
- N je počet vzorků (1024), X[k] je k-tý koeficient Fourierovy řady, n je index počítaného vzorku, k je počítadlo koeficientů.
- Přestože není výrazně pomalejší než knihovní implementace, v projektu se používá rychlejší np.fft.ifft()

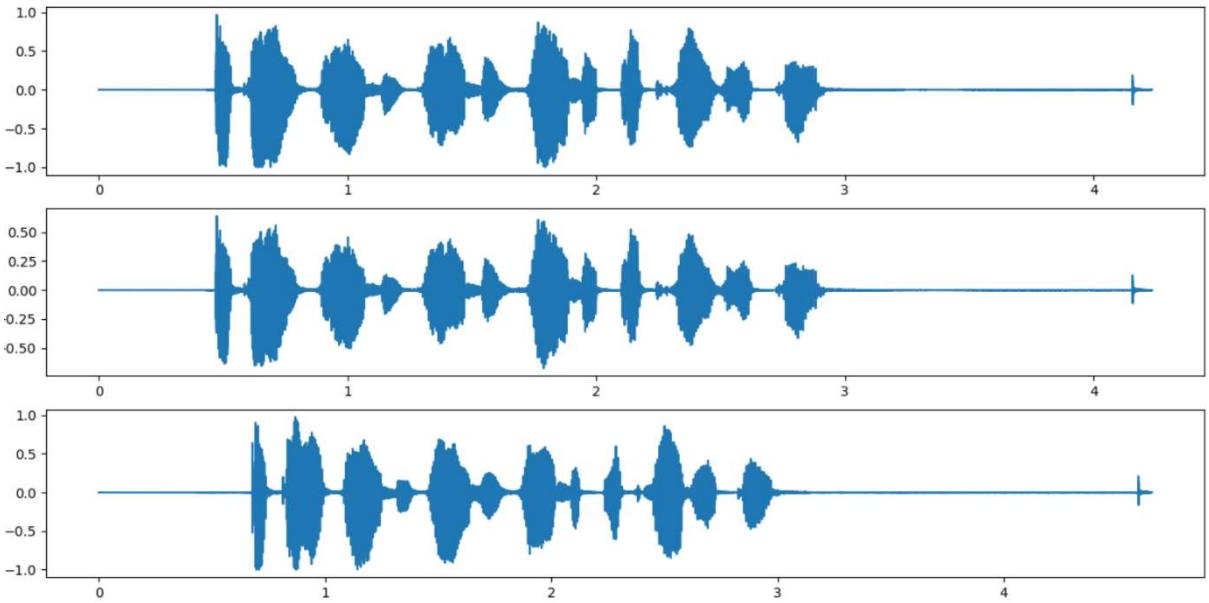
Níže uvádím výsledek IFFT (impulzní odezva):



Obrázek 9 - impulzní odezva

Úkol 8

Níže uvádím graf s aplikací filtru pomocí funkce lfilter:



Obrázek 10 - věta s rouškou, věta se simulovanou rouškou a věta s rouškou

Jediná významná změna je, že výsledek simulace je poměrně ztlumený oproti původnímu vstupu bez roušky. Jinak jsou vidět pouze malé změny u některých hrbolků a na poslech je rozdíl také opravdu jen minimální (zajímavé efekty probíhaly, když jsem zapomněl oříznout impulzní odezvu, pak tam byla ozvěna a do toho podivný kovový zvuk). Osobně si myslím, že možná jsem mohl trochu více „máznout“ první výběr dvou úseků, ze kterých se dělala DFT a následně filtr, čímž by došlo k zajímavějším posunům.

Úkol 9

Po provedení základních úloh jsem došel k tomu, že mnou vytvořené nahrávky a jejich následné zpracování na filtr nebylo příliš úspěšné, jelikož byly oba zvuky velmi podobné (a to i přesto, že jsem nepoužil roušku, nýbrž respirátor, u kterého bych právě očekával, že bude jako filtr silnější). Důvodů může být několik:

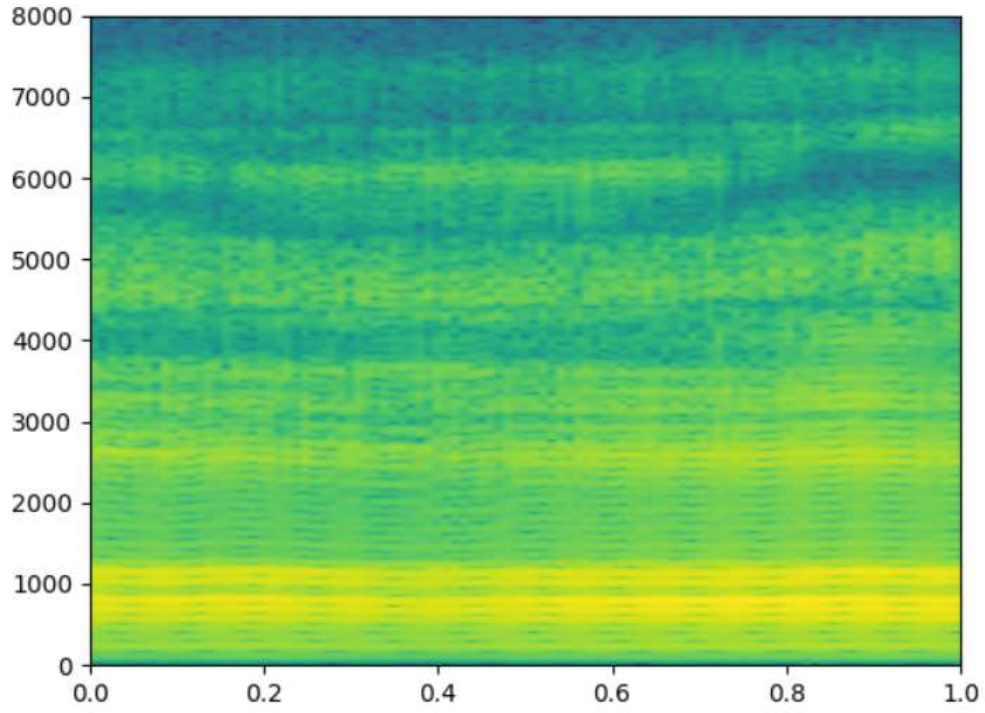
- Náhodou jsem trefil opravdu velmi dobré úseky.
- Při použití respirátoru podvědomě více tíhnu na čistší, silnější zvuk (hůře se v něm dýchá, tudíž více tlačím na bránici, abych rozdíl vyrovnal, nebo se už jenom psychologicky snažím to prostě udělat líp). Celkově by se dalo říct – lidský faktor.
- Špatné umístění mikrofonu – je možné, že mikrofon na náhlavní soustavě byl umístěn tak, že vykompenzovával velikost respirátoru, či na něj byl více natlačen (faktory měření). Otázka je, jestli by nebyly větší rozdíly, pokud by byl mikrofon umístěn v obou měřeních třeba metr od obličeje, kdy se více projeví rozostření způsobené respirátorem.
- Nedostatečné prostředky na záznam – jak mikrofon, tak použití AudaCity (AudaCity samo o sobě do velké míry ustředňovalo vstupní signál).
- Nedostatečná data obecně – pokud bych projekt prováděl znovu, nechal bych stát někoho dalšího v místnosti, kdo by řekl, jestli mnou vydávaný zvuk byl stejný (či do jaké míry podobný) s rouškou a bez.

Úkol 11

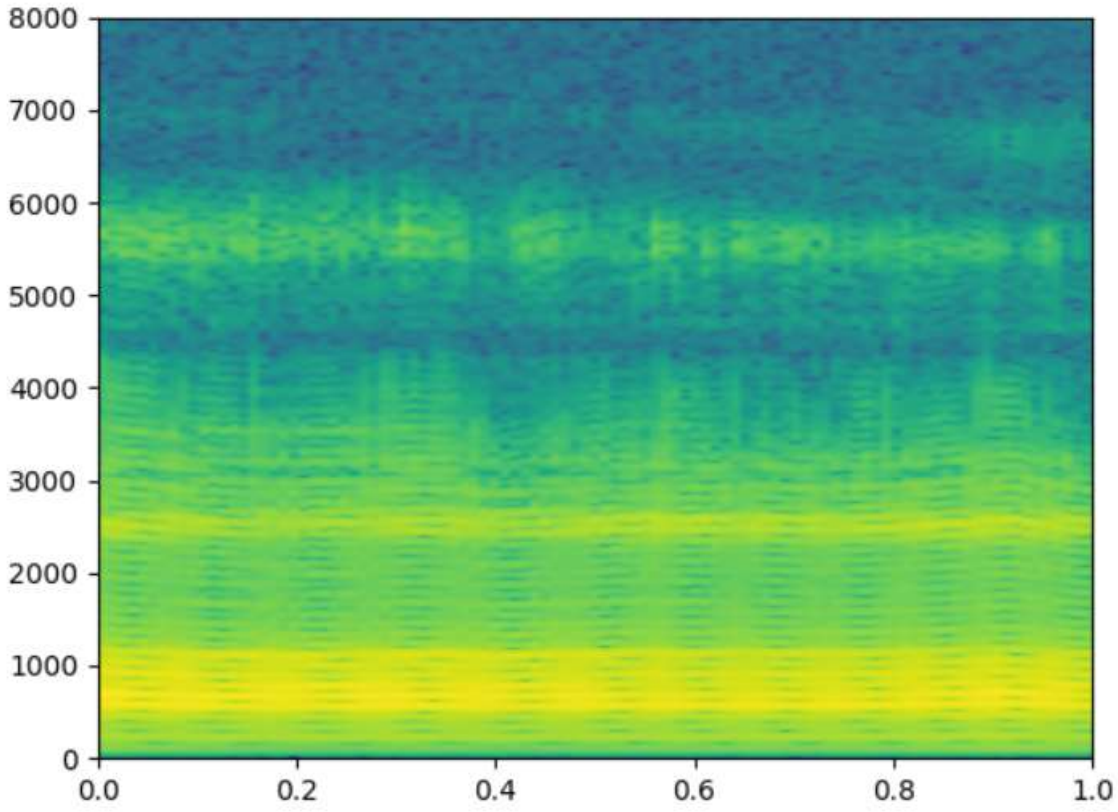
V tomto úkolu byla možnost si vybrat okénkovou funkci, a proto jsem si vybral Hanningovu okénkovou funkci (čistě designově se mi líbil její graf, který mi připomínal Gaussovu křivku).

Popis bodů takového okénka je definován dle vzorce (čerpáno z dokumentace numpy, odkaz v sekci zdroje):

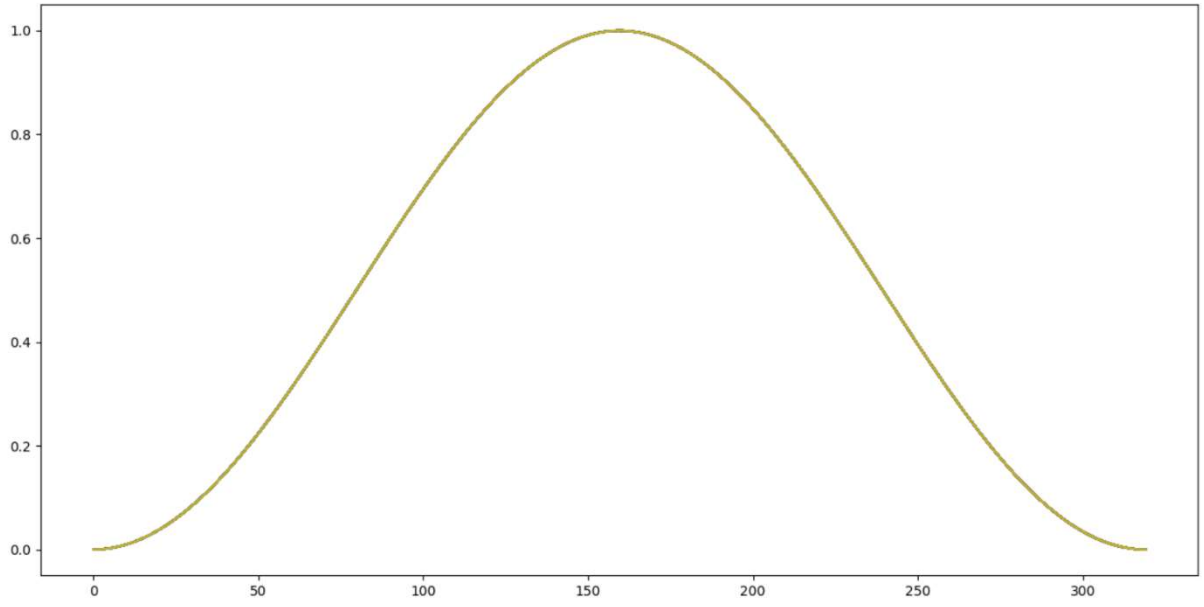
- $w(n) = 0.5 - 0.5\cos\left(\frac{2\pi n}{M-1}\right), 0 \leq n \leq M-1$
- Dále následují výsledky použití okénkové funkce



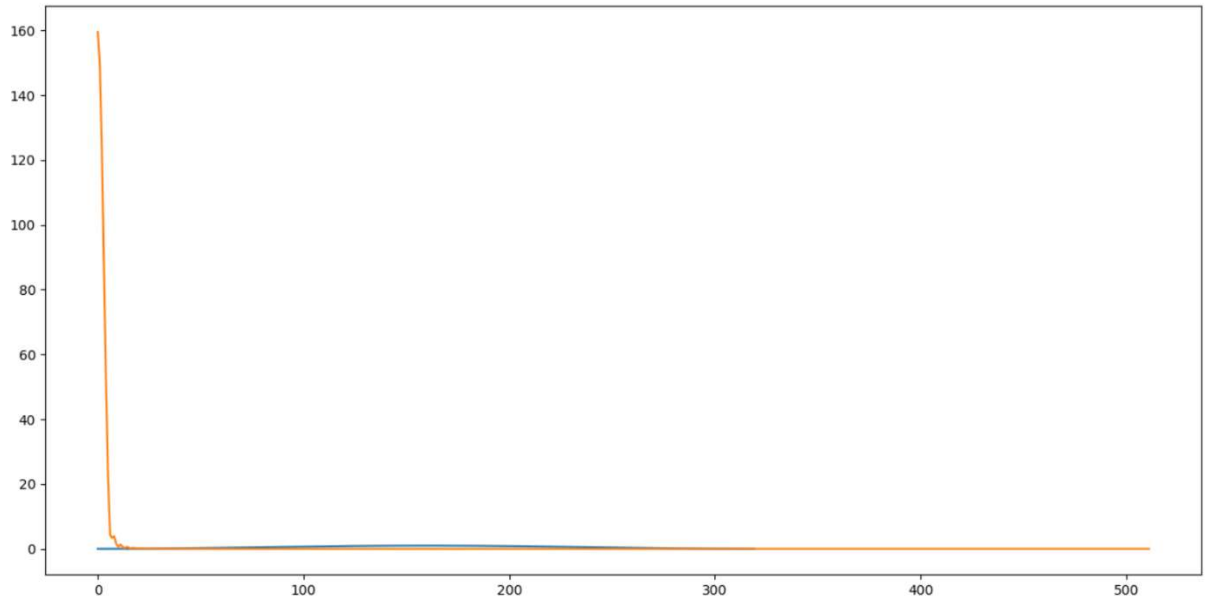
Obrázek 11 - spektrogram bez roušky po aplikaci Hanningovy okénkové funkce



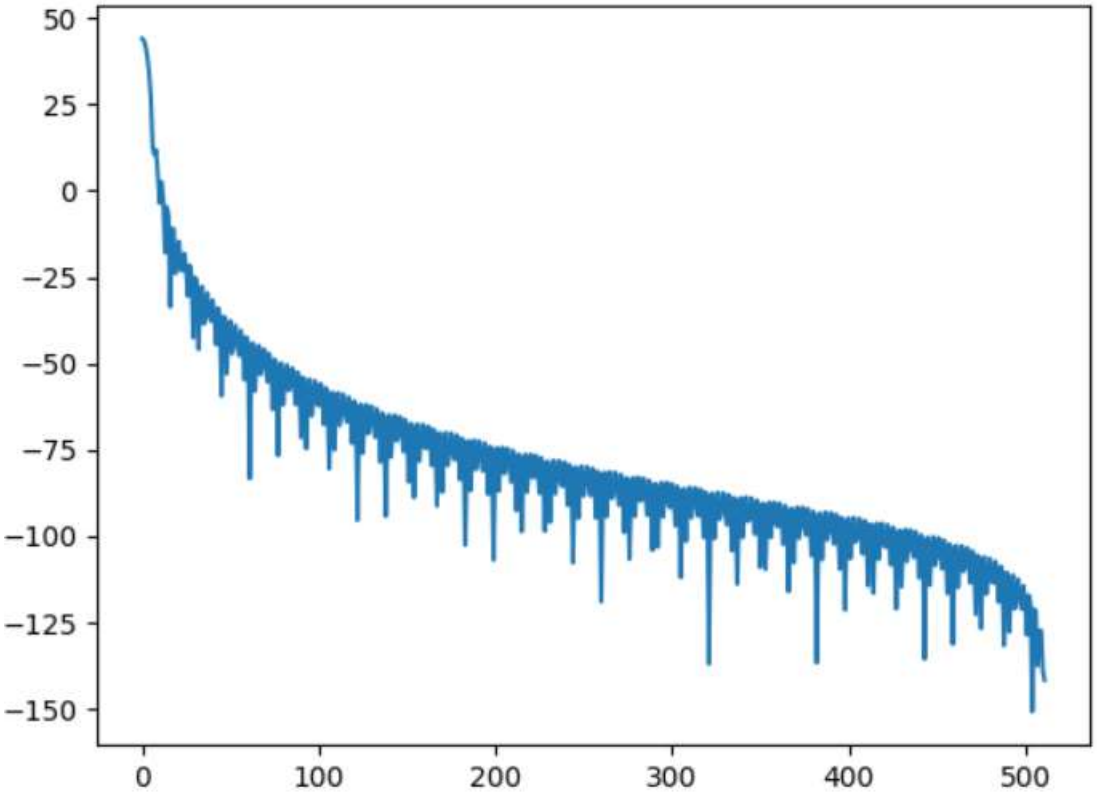
Obrázek 12 - spektrogram s rouškou po aplikaci Hanningovy okénkové funkce



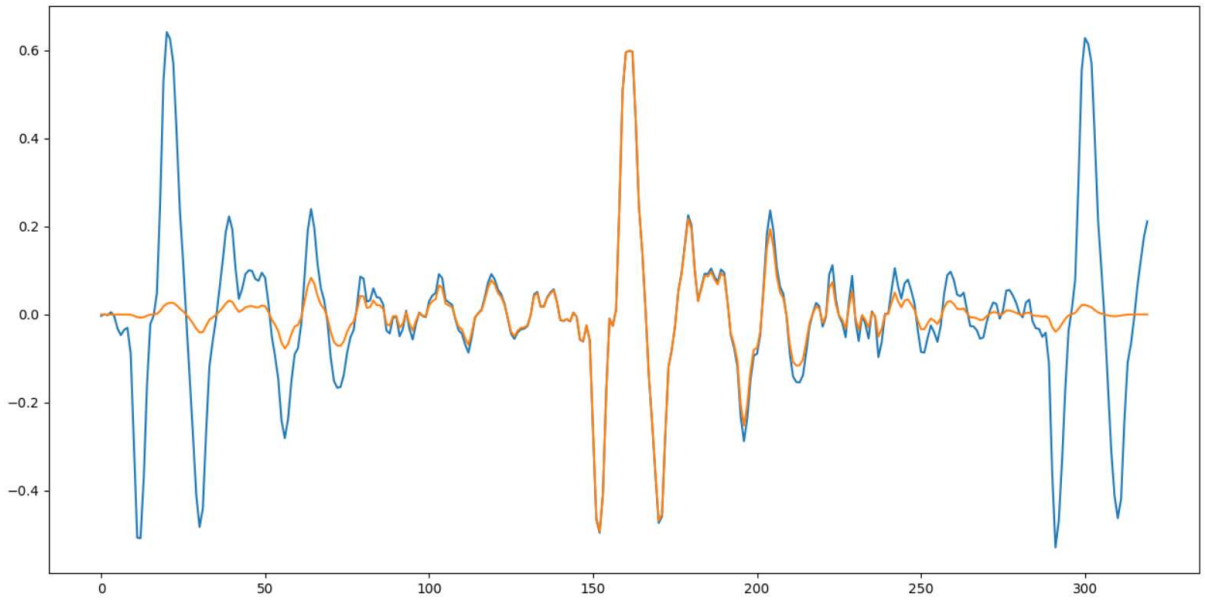
Obrázek 13 - graf Hanningovy okénkové funkce



Obrázek 14 - Hanningova okénková funkce po FFT



Obrázek 15 - výkonnostní DFT na Hanningově okénku (skutečné umělecké dílo)

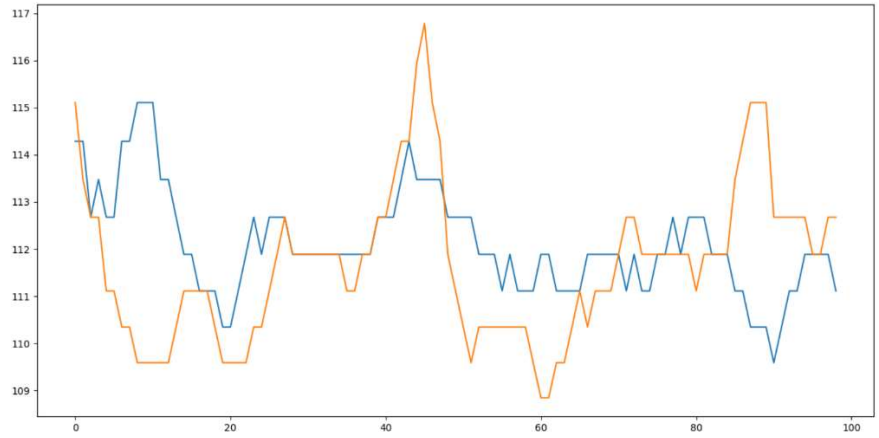


Obrázek 16 - modře rámec bez okénka, oranžově s okénkem

Krom toho, že se mi osobně esteticky Hanningova okénková funkce líbí (připomíná mi některé motivy ze hry Death Stranding), je užitečná v tom, že tlumí efekt okrajových částí rámce, kde bývají chyby. Co se pak týká spektrogramu, tak ten je poněkud více mázlý – nenachází se tam takové ostré hrany. Ve výsledku ale je nutné konstatovat, že v současném řešení projektu se okénková funkce nějak moc neprojevuje, možná akorát ještě více zpřesňuje simulaci, že je více správná, ale zde je obtížné pro mě najít rozdíl.

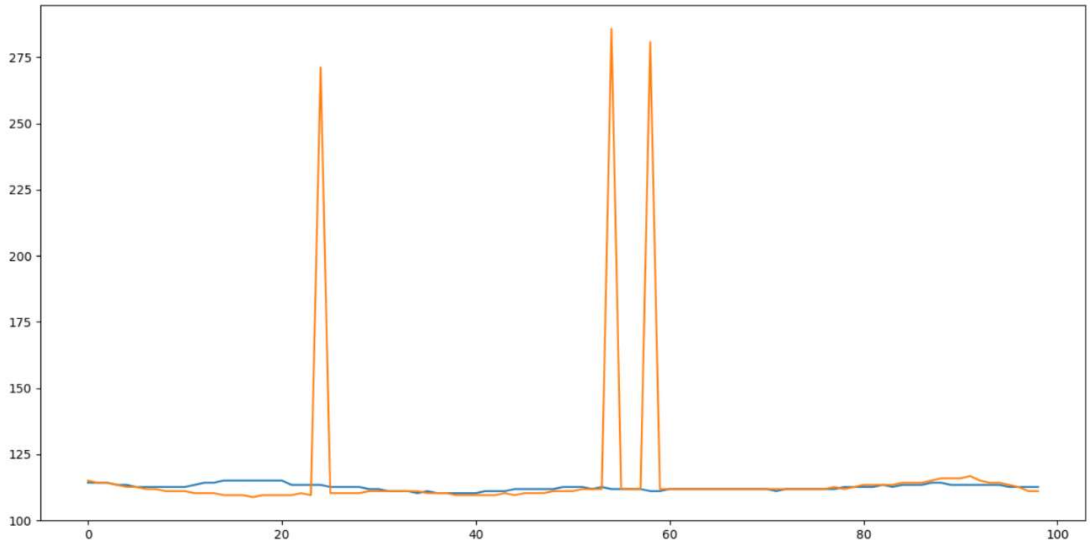
Úkol 12

Vyvolání popisované situace v zadání, tedy že se v autokorelačních koeficientech objeví větší extrém než je správný hledaný, se ukázalo jako velmi problematické. Prvním způsobem byla snaha vyvolat lag rozšířením rámců na řádově větší velikosti, což ale nakonec omezila celková délka nahrávky potřebná pro správný počet rámců.



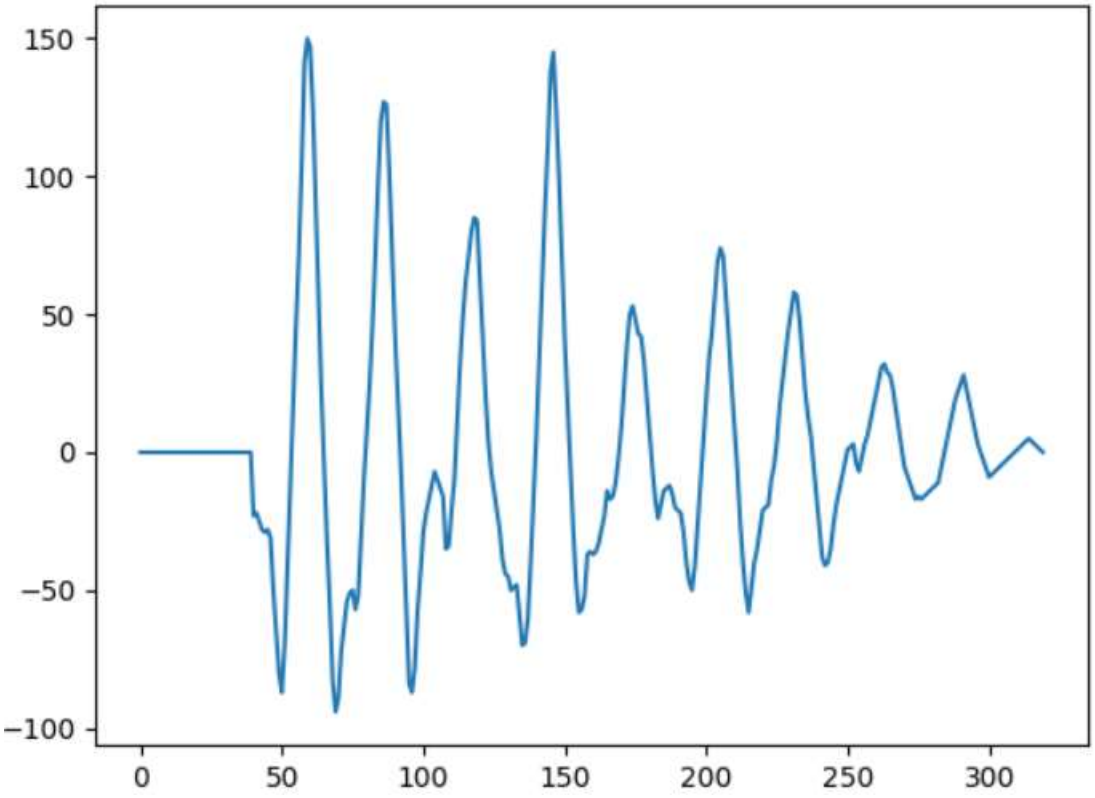
Obrázek 17 - extrémní snahy o vyvolání dvojnásobného lagu s pomocí zvětšování délky rámců – neúspěšné

Chybu se nakonec povedlo vyvolat omezením clippingu (přesněji řečeno jeho vypnutím). Můžeme ji vidět na následujícím grafu:



Obrázek 18 - chybné frekvence (projevily se pouze na rámcích s rouškou)

Autokorelační koeficienty v daném rámci pak vypadaly takto:

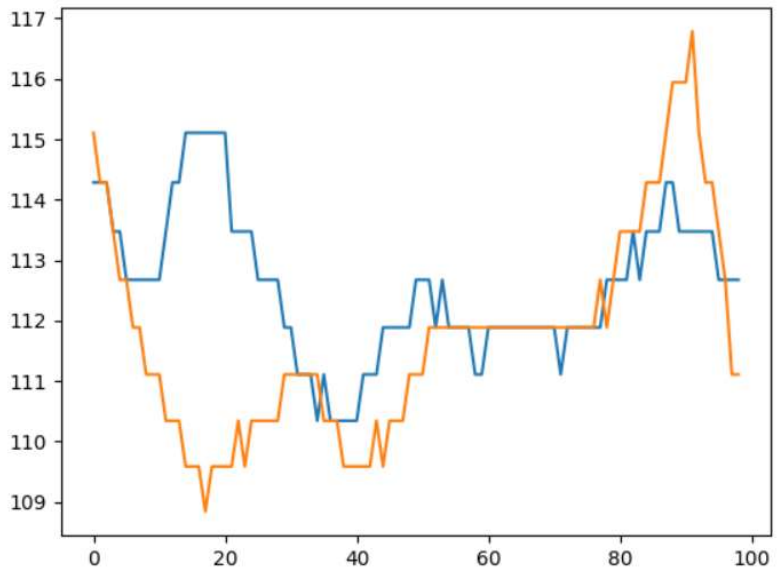


Obrázek 19 - autokorelační koeficienty u nulového clippování v rámci se špatným lagem

Nyní následuje krátký komentář/postup k dané chybě:

- Ve čtvrtém úkolu jsme zjistili, že lag bývá obvykle něco pod 150 a vidíme, že i zde je pod 150 velmi vysoký autokorelační koeficient. Ten je ale překonán autokorelačním koeficientem (velmi mírně) na hodnotě něco přes 50, tedy výsledná frekvence je kolem 275 (dle grafu výše).
- Z mého pohledu je řešení této chyby velmi jednoduché – obnovit clipping (tedy udělat prevenci chyby).

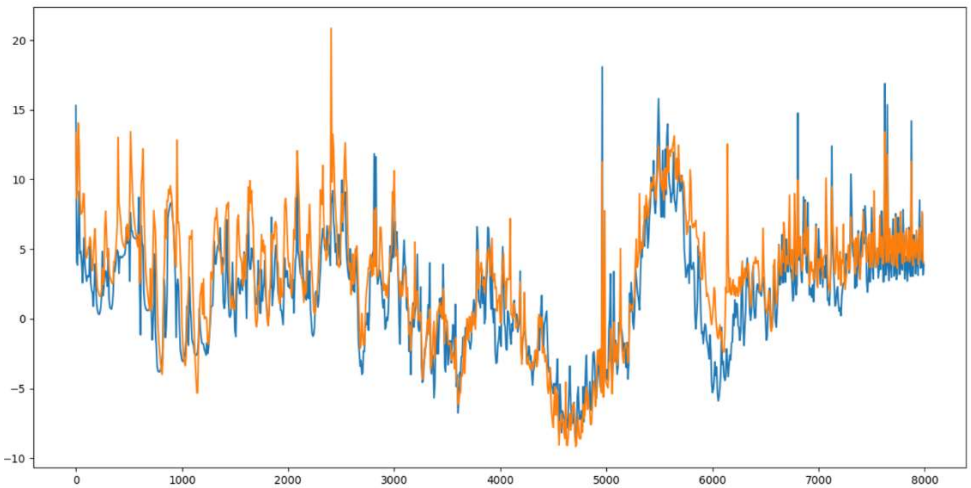
- Toto řešení však není to kýžené a v případě testování jiných nahrávek by se mohla chyba projevit i bez extrémních podmínek, tudíž je nutné implementovat nějakou metodu, která tuto chybu řeší.
- Je jí využívání mediánu z okolních hodnot – v grafu frekvencí vidíme, že extrémně vystupující frekvence jsou pouze ojedinělé a logicky můžeme usuzovat, že v případě stabilního tónu by se nejspíš neměly dít žádné skoky, tudíž nepěkné místo v grafu můžeme zahladit nahrazením skokové hodnoty mediánem okolí. Opět pak vychází očekávatelný graf. (Za skok se v mém řešení bere cokoliv, co přesahuje 1.5násobek střední hodnoty, medián se pak dělá z hodnot v okolí 3 doleva a 3 doprava.)



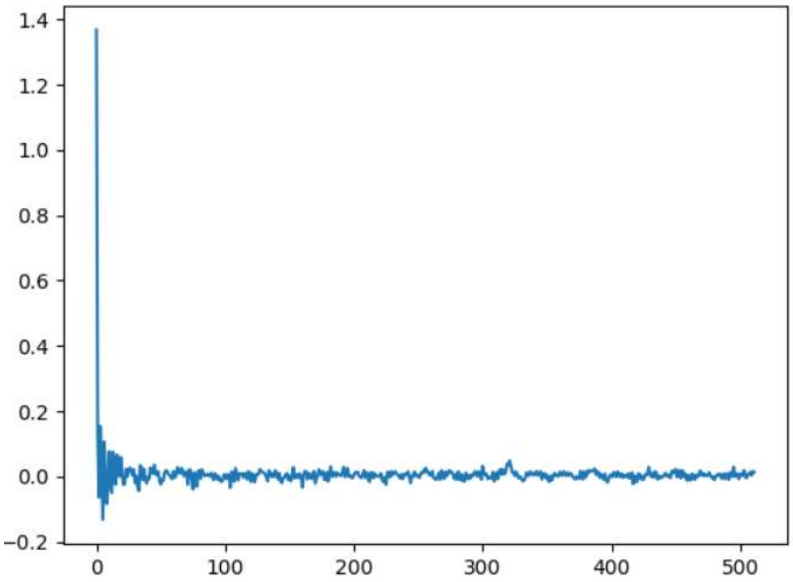
Obrázek 20 - opravené frekvence

Úkol 13

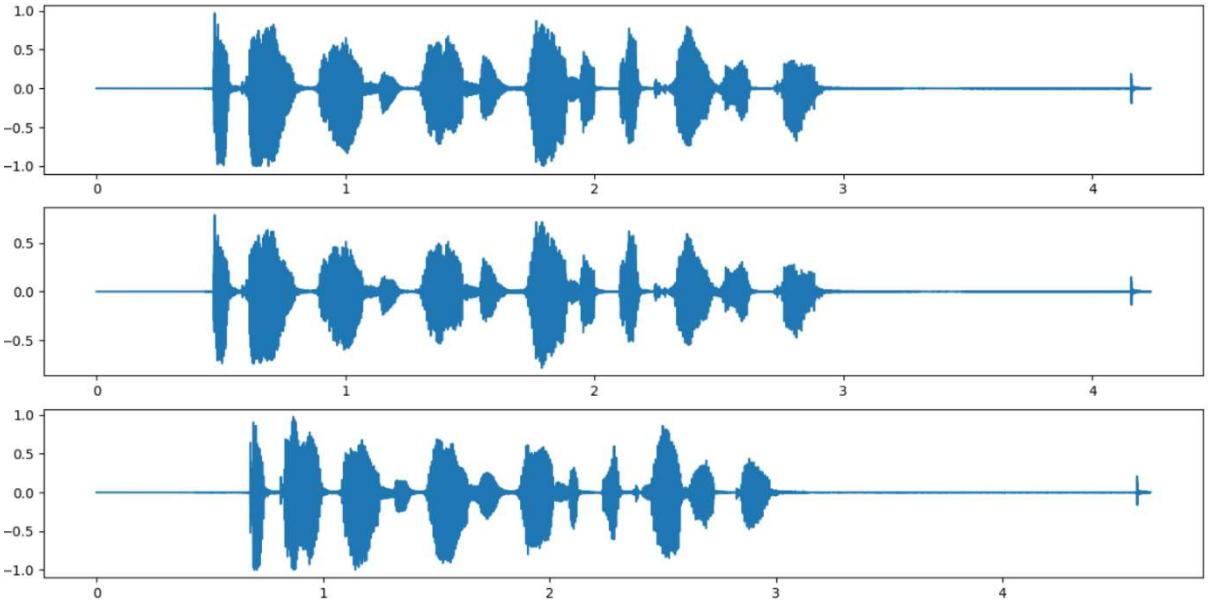
Princip, kterým se vyhazují jednotlivé nevhodné rámce je jednoduchý – po získání základních frekvencí ve 4. úkolu se nakopírují všechny rámce do pomocného pole, ze kterého se postupně budou s pomocí funkce np.delete() odstraňovat jednotlivé řádky s danými rámci, pokud si rámce nejsou (v rámci tolerance) rovny. Ty se pak v rámci freqz zpracovávají do frekvenční charakteristiky. Její výsledek viz níže:



Obrázek 21 - výkonnostní spektrum po odstranění rámců s různými frekvencemi (modrý odstraněné rámce, oranžový originál)



Obrázek 22 - impulzní odezva po odstranění rámců s různými frekvencemi



Obrázek 23 - věta bez roušky, věta se simulovanou rouškou, věta s rouškou

Podobně jako v předchozím úkolu, kde se aplikovala okénková funkce, i zde nedošlo k nějakým výrazným změnám. Vlastně by se akorát dalo říci, že se to vyplatí spíše než z hlediska přesnosti z hlediska rychlosti, jelikož místo 99 se zpracovává pouze 41 rámců (v mém případě, u jiných nahrávek to může být cokoliv od 0 do 99).

Úkol 15

- Bonus – vzájemná vzdálenost dvou takových vrcholků v rámci je vlastně perioda, takže pokud posuneme druhý rámec do levého vrcholku, získáme hodnotu, posuneme z původního místa do pravého vrcholku, získáme hodnotu, tak tento součet je logicky vzdálenost dvou vrcholků s největším významem (hlavním tónem) a tedy lag (perioda).

.

Závěr (2)

Po splnění některých bonusových úloh se v podstatě výsledek nemění oproti základnímu. Rozdíly mezi nefiltrovaným a filtrovaným záznamem se akorát ještě zmenšily (drobné hrbolky byly vyhlazeny atp.) Celkově se tedy výsledek dá zhodnotit jako mírné zklamání.

Zdroje

- Studijní materiály
- https://numpy.org/doc/stable/reference/generated/numpy.hanning.html