Vysoké učení technické v Brně
Fakulta informačních technologií

Signály a systémy

# Projekt – analýza vlivu masky na řeč

31. prosince 2020 Ondřej Ondryáš (xondry02)

## Použité prostředí

Projekt byl vypracován s použitím Python 3.8.5 a Jupyter Notebook 6.1.6. V adresáři `src/` se nachází soubor `project.ipynb` s Jupyter notebookem obsahujícím kód s výpočty, který generuje grafy použité v této dokumentaci do adresáře `src/plots/`. Doplňující úkoly 11, 13, 13+15 jsou v samostatných kopiích základního projektu, pouze úkol 12 je zařazen ve všech. Projekt vyžaduje kromě `jupyter` také knihovny `numpy`, `scipy`, `matplotlib`, `soundfile` a `IPython`.

# Základní zadání

## 1 Testovací nahrávky (tóny)

Nahrávky byly pořízeny telefonem Xiaomi Mi 8 s připojeným klipovým kondenzátorovým mikrofonem MAONO AU-100. Tón na nahrávce by měl přibližně odpovídat tónu A3 (220 Hz).

| soubor | délka [vzorky] | délka [s] |
|---|---|---|
| maskoff_tone.wav | 73 250 | 4,58 |
| maskon_tone.wav | 82 896 | 5,18 |

Tabulka 1: Informace o nahrávkách tónu

## 2 Testovací nahrávky (Věty)

| soubor | délka [vzorky] | délka [s] |
|---|---|---|
| maskoff_sentence.wav | 42 486 | 2,66 |
| maskon_sentence.wav | 43 209 | 2,70 |

Tabulka 2: Informace o nahrávkách věty

## 3 Rámce

### 3.1 Výběr podobných úseků

Z nahrávky *s maskou* jsem se pokusil vybrat podobný jednosekundový úsek s použitím cross-korelace. Z načtené nahrávky *bez masky* jsem zcela náhodně vybral jeden vzorek a z dat jsem ponechal pouze část od tohoto vzorku s délkou 16 000 (zůstalo mi tedy 16 000 vzorků, což odpovídá jedné sekundě). Poté jsem využil funkci `numpy.correlate`, která provedla korelaci $bez\_masky \star s\_maskou$, nejlepší shodu se signálem *bez masky* pak v signálu *s maskou* najdeme na indexu, který odpovídá indexu položky v poli výsledků korelace s nejvyšší hodnotou. Nyní můžu vzít z původních dat *s maskou* 16 000 vzorků od nalezeného indexu a dále pracovat jen s těmito daty.

Z nahrávky *bez masky* jsem (zcela náhodně) zvolil vzorky 16 625–32 624, pro tyto vzorky byla v nahrávce *s maskou* nalezena nejvyšší korelace na vzorcích 32 366–48 365.

V kódu je korelace provedena jednoduše použitím funkce knihovny numpy:

```
correlated = np.correlate(mask_data, nomask_data, mode='valid')
max_corr_index = np.argmax(correlated)
```

### 3.2 Rozdělení na rámce

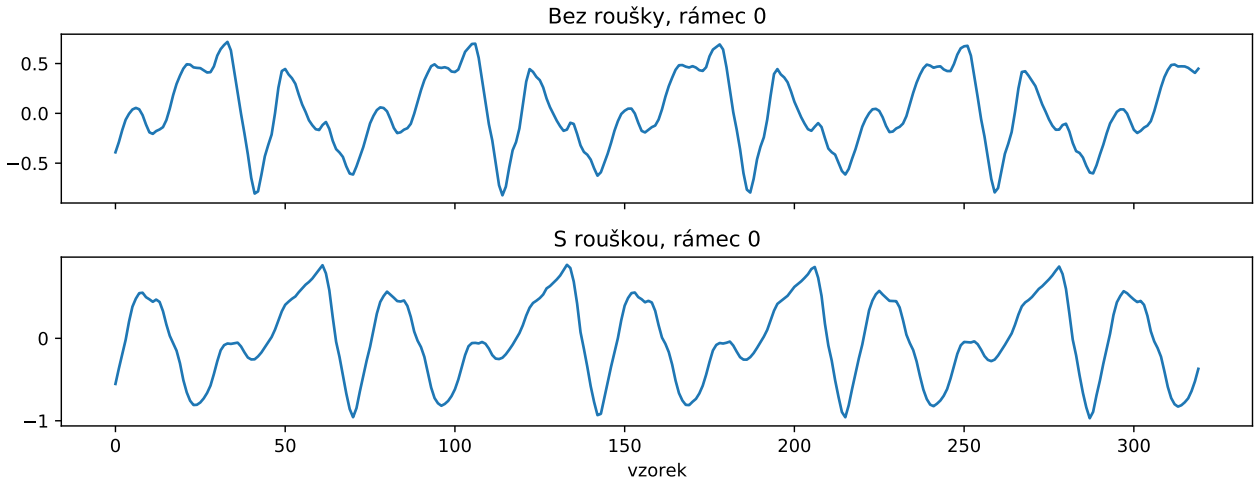
Nechť je $\tau$ délka rámce v ms a $F_s$ vzorkovací frekvence, pak je délka $a$ jednoho rámce ve vzorcích:

$$a = \frac{\tau}{1000} F_s$$

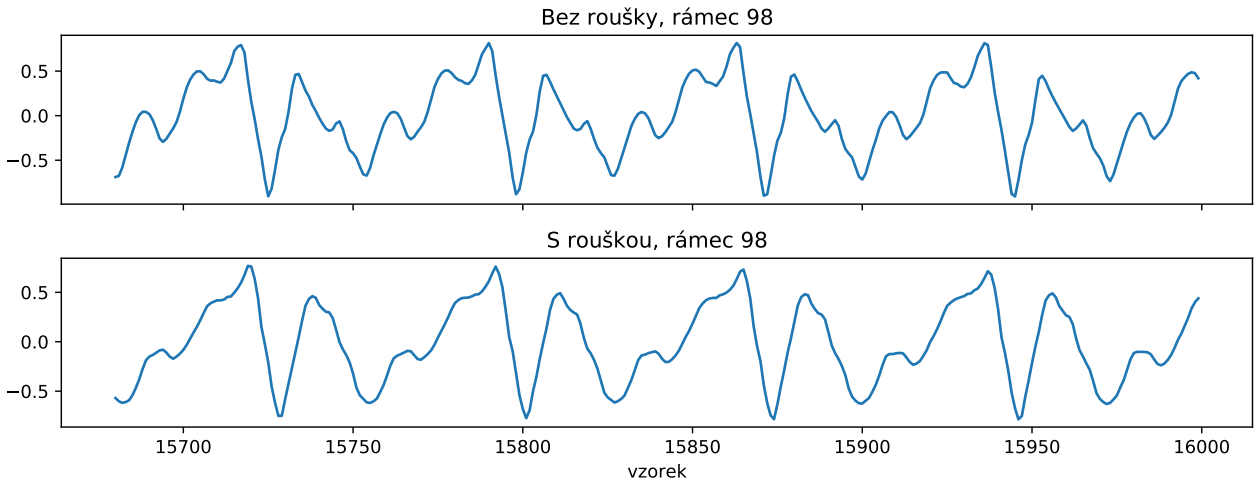
Naše rámce tak budou mít délku 320 vzorků. K rozdělení na rámce jsem použil trik s funkcí numpy, která vytváří pohled na původní data jako pole zadaného tvaru se zadaným krokem (viz [1]):

```
def framing(a, stride_length = 320, stride_step = 160):
    nrows = ((a.size - stride_length) // stride_step) + 1
    n = a.strides[0]
    return np.lib.stride_tricks.as_strided(a,
        shape=(nrows, stride_length), strides=(stride_step*n, n))
```

Rozhodl jsem se pracovat pouze s původně zadaným úsekem o velikosti jedné sekundy, poslední rámec, který by měl pouze poloviční velikost, tedy zahazuji a používám pouze 99 rámců. Při prvním pohledu mě zaujalo, že si rámce skutečně vcelku odpovídají (např. rámec 98, viz obr. 2), ale u některých je velmi zjevný posun (např. rámec 0, viz obr. 1).




Obrázek 1: Rámec 0




Obrázek 2: Rámec 98

# 4 Zjištění základního tónu

Autokorelace byla implementována podle základního vztahu (viz např. [2, str. 62]):

$$R[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n]\, x[n+k]$$

Mé řešení využívá některých funkcí a vlastností knihovny numpy (např. násobení polí po položkách), které oproti naivnímu vnořenému cyklu výpočet urychlují; výrazně efektivnější by ovšem bylo implementovat korelaci například s využitím DFT (FFT). Výsledky jsem srovnal také s výsledky korelační funkce `numpy.correlate`, shodují se.

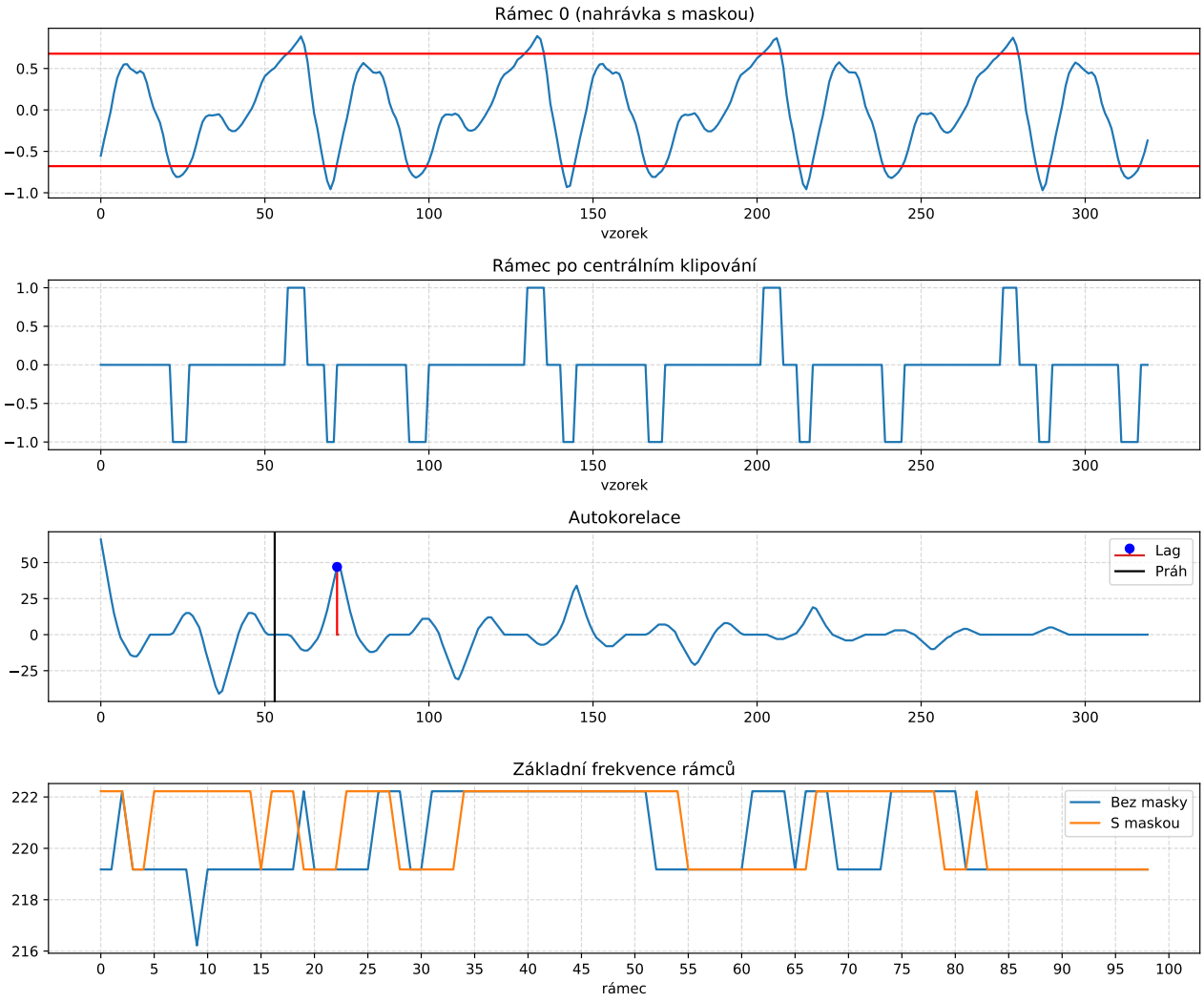
Jako práh jsem zvolil hodnotu 300 Hz, protože vzhledem ke znalosti vstupních signálů očekávám lag na frekvenci cca 220 Hz – volba vyššího prahu by byla zbytečná (a případně mohla vést ke zvolení koeficientu ze začátku posloupnosti, kterých se prahováním chceme zbavit), volba nižší hodnoty by už vedla k označení nižšího koeficientu, než je hledaný maximální.

Index maximálního koeficientu, lag, se dá převést na frekvenci podle vztahu $f_0 = \frac{F_s}{L}$, kde $L$ je index vzorku [2, str. 61]. Při použité vzorkovací frekvenci $F_s$ = 16 000 Hz se tak výrazně projeví i ±1 chyba hodnoty lagu – např. v rámci 0 je hodnota lagu 73 (viz obr. 3), rozdíly mezi podíly $\frac{16000}{73}$, $\frac{16000}{72}$ a $\frac{16000}{74}$ jsou 3,04 Hz, resp. 2,96 Hz, což jsou nezanedbatelné chyby (více než dvakrát větší než je směrodatná odchylka napříč zjištěnými základními frekvencemi všech rámců, viz tab. 3). Tuto chybu by tedy mohlo omezit zejména zvýšení vzorkovací frekvence – spolu s tím by se úměrně zvýšila i hodnota lagu a chyba ± 1 by tak vedla k nižšímu rozdílu výsledku (například při dvojnásobném zvýšení v původním příkladu: $\frac{16000 \cdot 2}{73 \cdot 2} - \frac{16000 \cdot 2}{73 \cdot 2 + 1} = 1{,}491$, což je výrazně menší rozdíl než původních 2,96 Hz). To je ovšem v praxi často nereálné, mohli bychom tak signál třeba nadvzorkovat a interpolovat.

| nahrávka | střední hodnota $f_0$ [Hz] | rozptyl $f_0$ [Hz$^2$] | směrodatná odchylka $f_0$ [Hz] |
|---|---|---|---|
| bez masky | 220,378 | 1,546 829 | 1,243 715 |
| s maskou | 220,869 | 1,512 645 | 1,229 896 |

Tabulka 3: Zjištěné údaje o základní frekvenci testovacích nahrávek

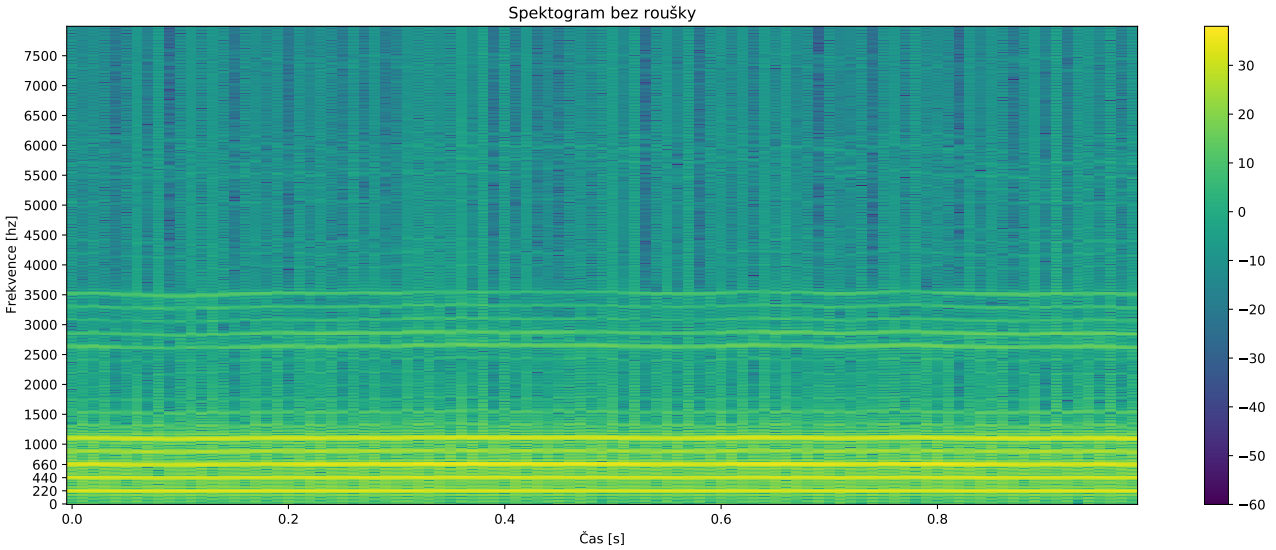
Zjištěné údaje ukazují, že jsem na obou nahrávkách vcelku spolehlivě udržel stejný tón, jehož střední hodnotou je přibližně cílových 220 Hz, ačkoliv přesně tuto frekvenci v žádném z rámců nezískám.



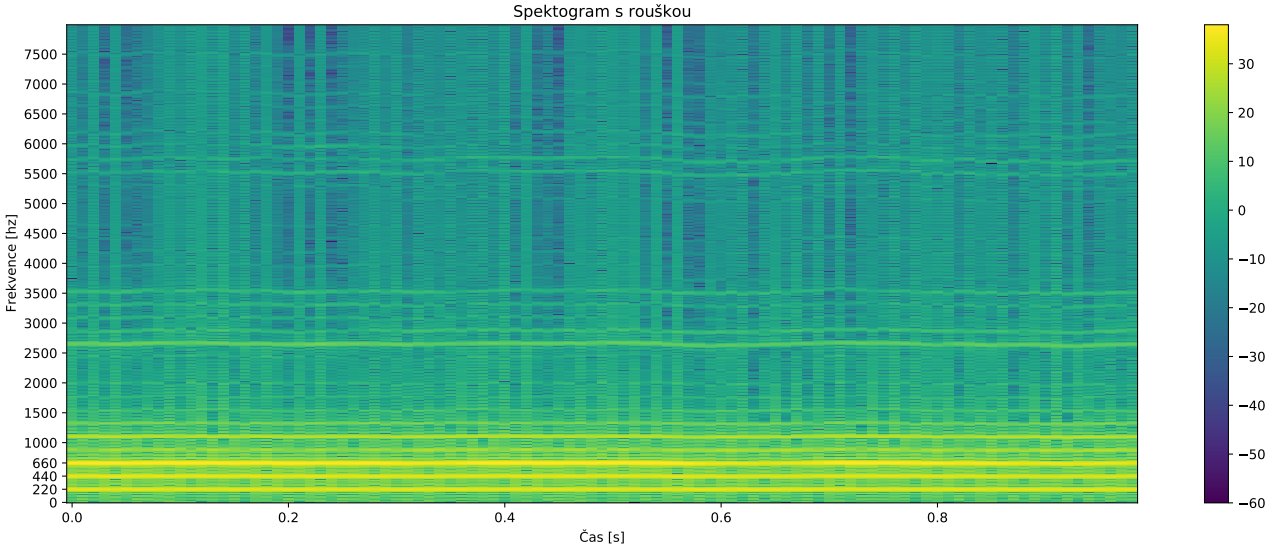

Obrázek 3: Ukázka klipování a autokorelace rámce 0; základní frekvence všech rámců. Červené přímky v prvním grafu naznačují 70% hranice pro klipování.

# 5 Spektrogramy

Na spektrogramech (obrázky 4 a 5) jsou velmi zřetelně vidět vrcholy výkonu signálu na frekvencích, které jsou násobky základní frekvence ~220 Hz (první tři násobky jsem explicitně uvedl i v popiscích osy $y$). Také je vidět, že signál z nahrávky *s maskou* je o něco zašuměnější – rozdíly mezi hodnotami na násobcích základní frekvence a jejich těsným okolím jsou méně výrazné, méně výrazné jsou také vrcholy na vyšších násobcích ($12 \ldots 16 \times f_0$).




Obrázek 4: spektrogram úseku nahrávky tónu bez masky




Obrázek 5: spektrogram úseku nahrávky tónu s maskou

## 5.1 Implementace DFT

Vlastní DFT jsem implementoval podle základního vztahu pro její výpočet:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j2\pi \frac{n}{N} k}$$

Taková implementace je samozřejmě velmi neefektivní, pokusil jsem se ji tedy alespoň trochu urychlit předpočítáním hodnot $e^{-j2\pi \frac{n}{N} k}$. Implementace tímto způsobem vypočítá DFT na mém počítači za přibližně 3 sekundy,

zatímco FFT z numpy zabere méně než 10 milisekund. Výsledky jsou každopádně totožné (ověřeno použitím funkce `numpy.allclose`). Správný počet vzorků je zajištěn vytvořením kopie vstupního pole, která je přitom napravo vyplněna nulami až do požadované velikosti. Funkce přebírá rovnou dvourozměrné pole, které reprezentuje jednotlivé rámce, a vypočítá DFT pro všechny najednou.

```
def dft(frames, n=1024):
    # Zarovnani vstupnich poli nulami (pouze ve druhe dimenzi obsahujici data ramcu)
    padded = np.pad(frames, ((0,0), (0, n - frames.shape[1])),
        constant_values=((0,0), (0,0)))
    # Vystupni pole
    ret = np.zeros(padded.size, dtype='complex128').reshape(padded.shape)

    # Predpocitani mocnin e pro vsechny dvojice "k, n"
    es = np.zeros(n**2, dtype='complex128').reshape(n, n)
    for k in range(0, n):
        for na in range(0, n):
            es[k][na] = np.exp(-2j*np.pi*k*na/n)

    # Vypocet DFT
    for frame_i in range(0, len(padded)):
        frame = padded[frame_i]
        ret_frame = ret[frame_i]

        for k in range(0, n):
            ret_frame[k] = np.sum(frame * es[k])

    return ret
```

## 6 Frekvenční charakteristika

Vypočítané DFT signálů můžeme brát jako $z$-transformaci $x[n] \xrightarrow{\mathcal{F}} X(z)$. Frekvenční charakteristika $H(e^{j\omega})$ se pak dá vyjádřit pomocí přenosové funkce:

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

V kódu nám $Y(z)$ a $X(z)$ představují pole koeficientů, které jsme vypočítali pomocí DFT. Hledanou frekvenční charakteristiku (což je prakticky spektrum impulsní odezvy filtru) tak získáme pouhým vydělením jednotlivých položek těchto polí. Následně hodnoty upravíme do podoby výkonového spektra podle vztahu ze zadání.

```
H = mask_fr/nomask_fr # Najednou vypocita podily vsech odpovidajicich hodnot v ramcich
H_mean = np.mean(H, axis=0) # Spocita prumery odpovidajicich polozek napric ramci
H_log = 10*np.log10(np.abs(H_mean)**2)
```

Na první pohled je zřejmé, že filtr lehce tlumí signál v téměř celém jeho rozsahu, výrazněji však na frekvencích kolem 1 100 Hz a zejména kolem 2 800 Hz, funguje tedy zhruba jako *pásmová zádrž*. Pro zajímavost jsem nechal část frekvenční charakteristiky vyhladit filtrem *Savitzky–Golay* s polynomem 11. řádu [3] (viz obr. 7), nejnižší hodnotu tato vyhlazená frekvenční charakteristika nabývá na frekvenci cca 2 921 Hz. Jedná se o FIR filtr.

## 7 Implusní odezva

Implusní odezvu, která bude později použita pro aplikaci filtru na signál, spočítáme použitím inverzní DFT na frekvenční charakteristiku.

### 7.1 Implementace IDFT

IDFT je implementována s využitím stejné metody, kterou používá má implementace DFT (viz kap. 5.1). Opět tedy vychází ze základního vztahu

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \cdot e^{i\frac{2\pi}{N}kn}$$

V tomto případě se akorát pracuje rovnou s jednorozměrným polem koeficientů DFT. Opět jsem provedl srovnání s funkcí `numpy.fft.ifft`, výsledky jsou stejné, ale trvání knihovní funkce je téměř neměřitelné (pod 1 ms), zatímco použití mé implementace zabere asi 2 sekundy.

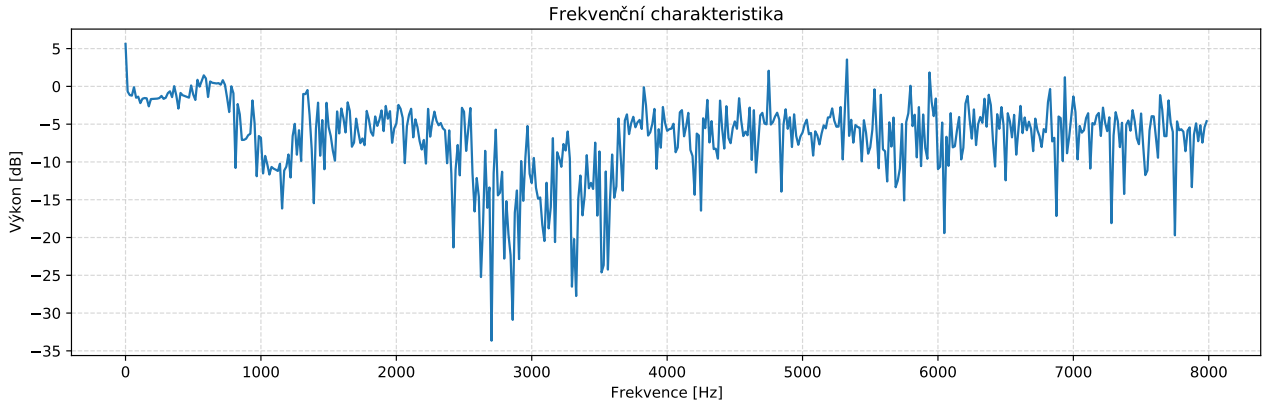
```
def idft(X, n=1024):
    # V nasem pripade se nepouzije, vstupni pole bude mit 1024 prvku,
    # ale pro ucely aspon nejake obecnosti...
    padded = np.pad(X, (0, n - len(X)), constant_values=((0, 0), ))
    # Vystupni pole
    ret = np.zeros(padded.size, dtype='complex128')

    # Predpocitani mocnin e pro vsechny dvojice "k, n"
    es = np.zeros(n**2, dtype='complex128').reshape(n, n)
    for k in range(0, n):
        for na in range(0, n):
            es[k][na] = np.exp(2j*np.pi*k*na/n) # kladne znamenko!

    # Vypocet IDFT
    for k in range(0, n):
        ret[k] = np.sum(padded * es[k]) / n # delime N!

    return ret
```

Impulsní odezvu filtru vidíme na obrázku 8. Hodnota prvního vzorku (a tedy prvního koeficientu $b_0$ výsledného FIR filtru) je řádově vyšší než všechny další hodnoty, což (z pohledu intuice) naznačuje, že se výstupní signál bude vstupnímu signálu „celkem podobat“.

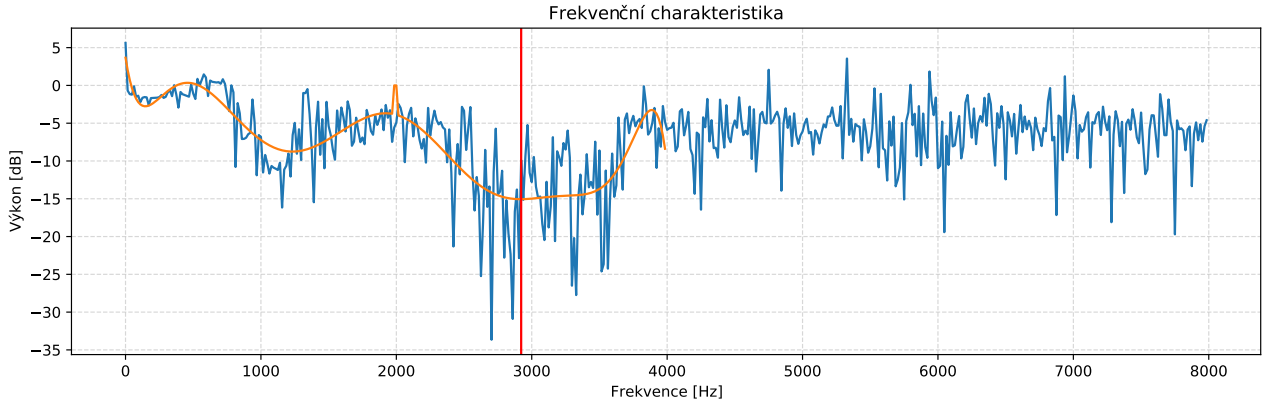
## 8 Simulace masky

Příjemnou vlastností FIR filtrů je, že hodnoty vzorků jejich impulsní odezvy odpovídají koeficientům $b_i$ diferenční rovnice, která daný filtr popisuje. S využitím funkce z knihovny scipy tak můžeme jednoduše filtr aplikovat na nějaký signál:

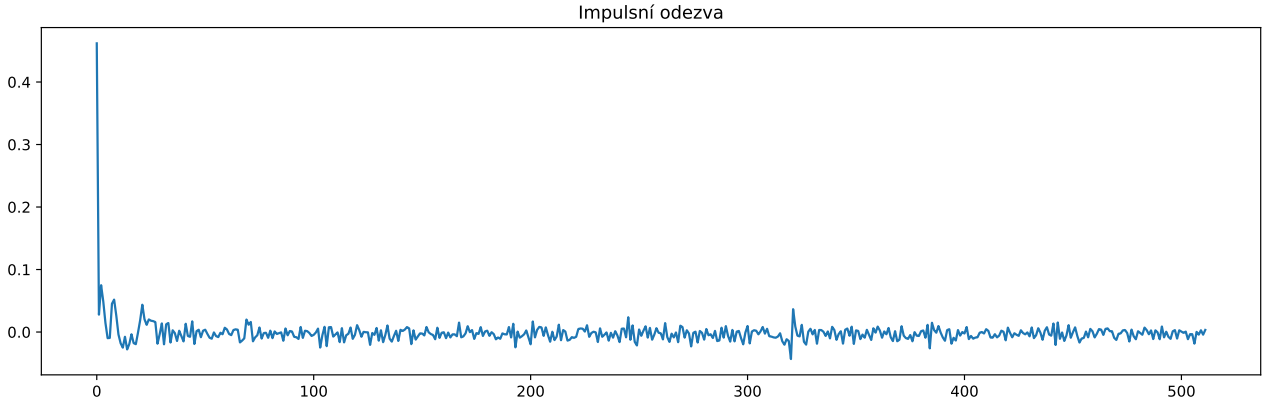
```
mask_simulation = signal.lfilter(impulse_resp, [1.0], nomask_sent_data)
```




Obrázek 6: Frekvenční charakteristika výsledného filtru




Obrázek 7: Frekvenční charakteristika výsledného filtru s vyhlazením



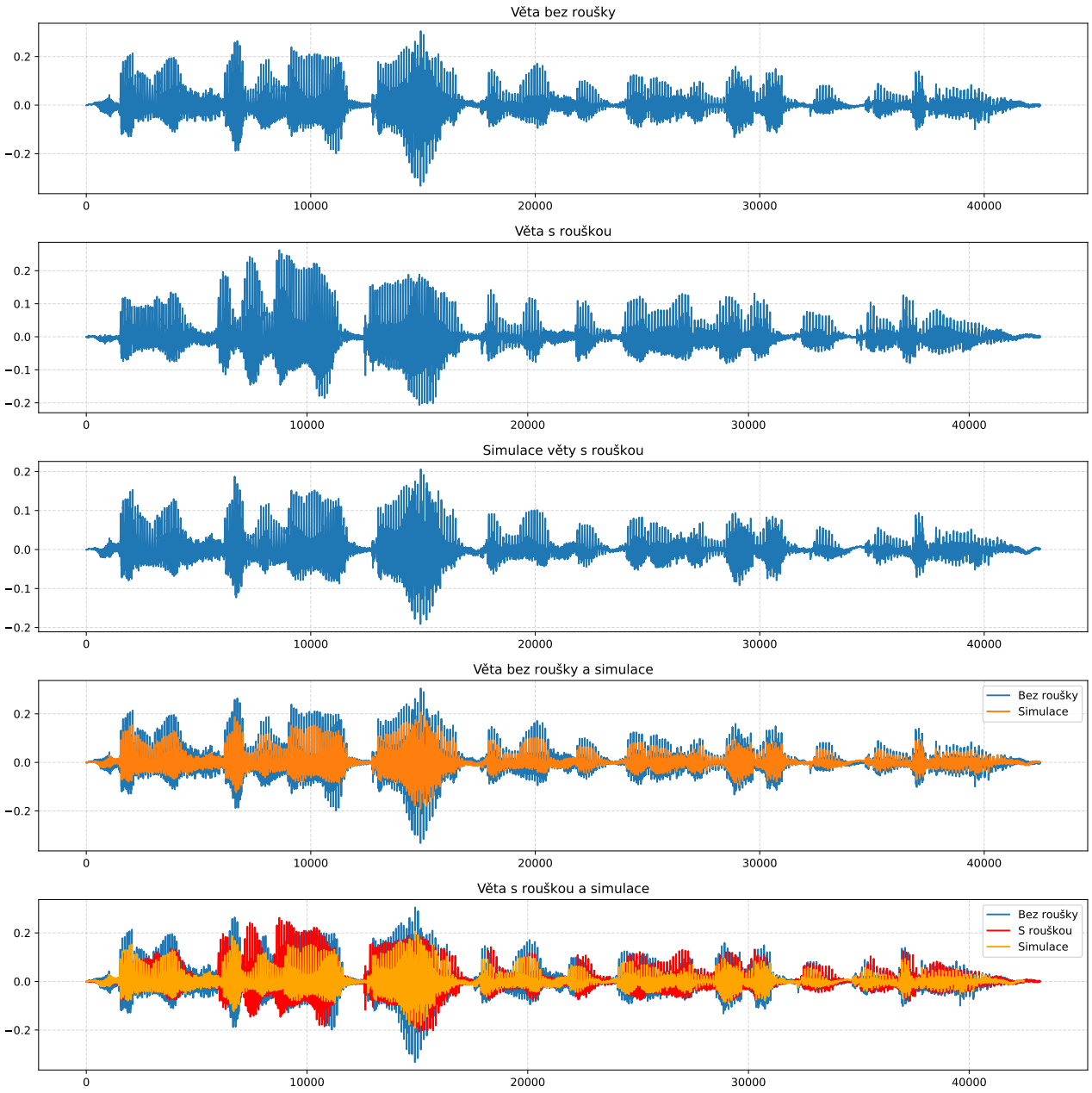

Obrázek 8: Impulsní odezva výsledného filtru

Na obrázku 9 můžeme vidět grafy vstupní nahrané věty bez masky, věty s maskou a simulace masky pomocí získaného filtru (zvlášť a překrytá se vstupními nahrávkami). Simulace se očekávatelně blíží spíše vstupnímu signálu než skutečné nahrávce s maskou – na čtvrtém grafu je vidět, že největším dopadem filtru je celkové utlumení signálu (jak bylo naznačeno v kap. 6). Zaujalo mě chování filtrů v částech, kde má vstupní signál nižší hodnotu – filtr v nich signál jaksi „rozkmitává“ (je vidět například úplně na konci, kolem vzorků ± 17 300, 21 100 nebo 34 100; výstupní signál zde dělá viditelné „obloučky“). Zajímavé také je, že simulovaný signál je na většině míst dokonce utlumenější než skutečný signál s maskou. Nejvíce se signály liší v části mezi vzorky cca 6 000–12 000, v této části se od sebe ovšem výrazněji odchylují i vstupní nahrávky, není tedy překvapivé, že zde simulace neodpovídá skutečné nahrávce. Naopak se zdá, že simulace zafungovala vcelku spolehlivě na úseku mezi vzorky cca 12 000–17 000.

Pro zajímavost jsem mezi jednotlivými nahrávkami a simulací (ustředněnými a normalizovanými) provedl korelaci. Maximální nalezené hodnoty jsou přibližně následující:

- Autokorelace nahrávky s maskou: 964
- Korelace simulace a nahrávky bez masky: 784
- Korelace simulace a nahrávky s maskou: 148
- Korelace nahrávky bez masky a nahrávky s maskou: 137

Nejsem si jistý, jak moc relevantní je porovnávat mezi sebou hodnoty korelace, ale přesto považuji za úspěšné, že korelace simulace a nahrávky s maskou vychází alespoň o něco vyšší než korelace zdrojové nahrávky bez masky a nahrávky s maskou.




Obrázek 9: Simulace filtru na nahrávce věty

# 9 Závěr

Analyzační část projektu považuji za úspěšnou: podařilo se mi najít podobné (významně korelované) úseky nahrávek, rozdělit je na rámce a zjistit jejich základní frekvence a z dat sestavit spektrogramy, které potvrzují, že se v signálu nejvíce objevují frekvence na násobcích frekvence základní. Frekvenční charakteristika ukázala, že filtr představující masku do jisté míry vykazuje podobu pásmové zádrže na frekvencích kolem 1 kHz a 2,8 kHz, veskrze ale signál především celý zeslabuje.

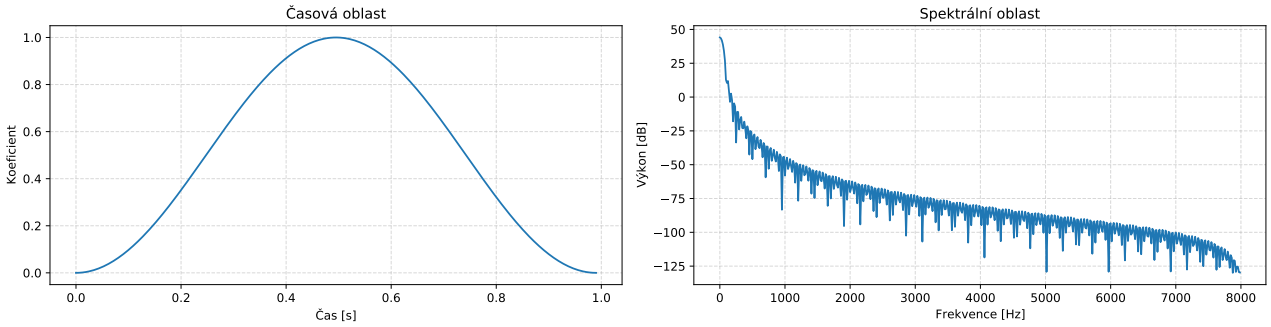
Simulace ukázala, že tímto způsobem vytvořený filtr není ani zdaleka dokonalý. Reálné utlumení signálu s maskou oproti signálu bez masky je menší než to, které způsobí filtr. Domnívám se, že by to mohlo být – kromě celkově nedokonalé metody odhadu – způsobeno příliš nízkým počtem vstupních dat (do rámců máme rozdělených pouze 0,99 sekund z původních nahrávek); také si myslím, že schopnost filtru spolehlivěji replikovat vliv masky na běžnou větu mohl ovlivnit fakt, že dvojice tónů, podle kterých jsem filtr vytvořil, je na poněkud vyšší frekvenci, než je základní frekvence mého hlasu v běžné větě (tou je v mém případě cca 130 Hz).

# Doplňující úkoly

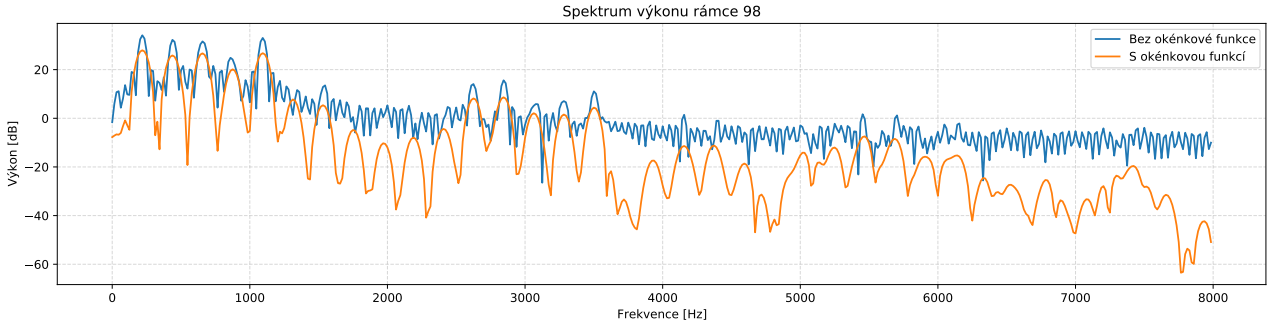
## 10 Okénková funkce

Použil jsem Hannovu okénkovou funkci, která by podle různých zdrojů ([4], [5]) měla být poměrně vhodnou funkcí pro neznámé náhodné signály. Okénková funkce upravuje tvar vstupního signálu způsobem, který jej na jeho krajích přibližuje k nule (v případě Hannovy funkce bude nultý a poslední vzorek určitě roven nule), čímž potlačuje vznik falešných hodnot na vyšších frekvencích při výpočtu DFT – na vstupní signál se po jejím použití dá nahlížet „jako na periodický“. Na obrázku 10 jsou vidět charakteristiky použité okénkové funkce, obrázek 11 pak zachycuje spektrum rámce 98 před a po použití této funkce. Okénková funkce je na rámce aplikována v časové oblasti, tedy prostým pronásobením hodnot:

```
win = signal.windows.hann(320) # Vytvori okenkovou funkci na 320 vzorcich
frame_windowed = nomask_frames[fr_i] * win # Pronasobime puvodni ramec okenkovou fci
fr_windowed_dft = np.fft.fft(frame_windowed, n=1024) # DFT pro plotnuti spektra
```



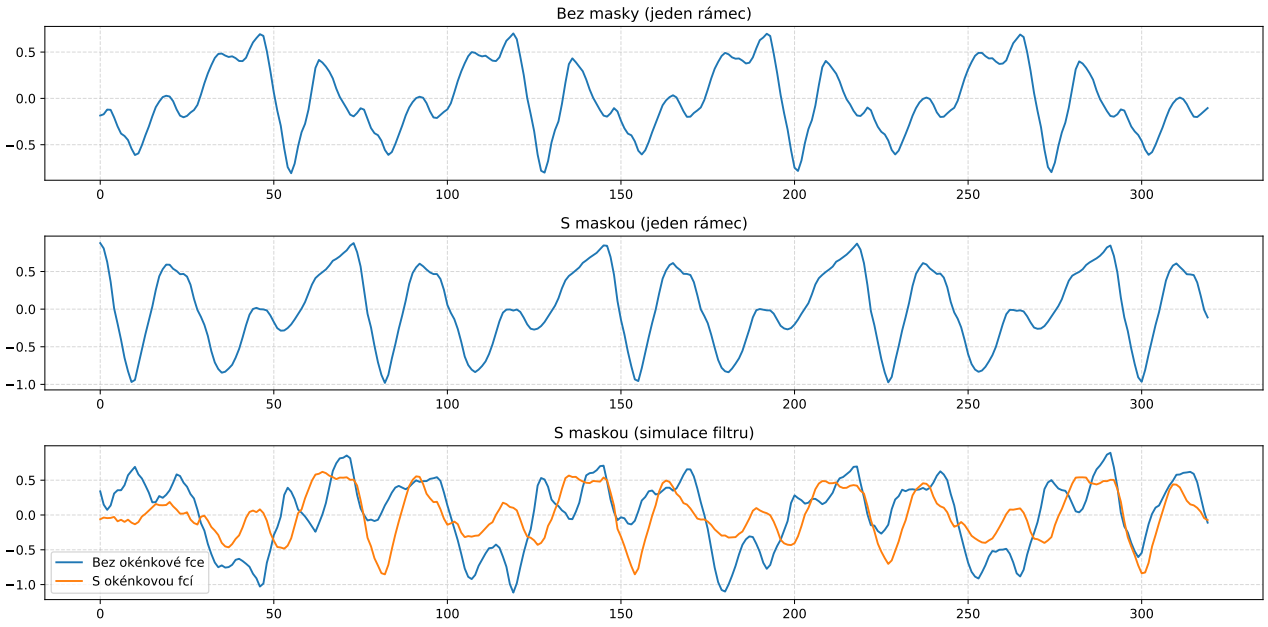

Obrázek 10: Hannova okénková funkce




Obrázek 11: Spektrum rámce před a po použití okénkové funkce

Na spektru rámce je vidět, že tato okénková funkce na nejvíce zastoupených frekvencích „rozšíří obloučky“, čímž signál *vyhladí*, a „zvýrazní průběh“ signálu – na místech, kde jeho hodnota klesá, po použití okénkové funkce klesá výrazněji. Tvar signálu je každopádně celkem výrazně pozměněn.

Zajímavý je dopad při použití následně vytvořeného filtru. Na obrázku 12 můžeme srovnat jeden rámec z obou původních nahrávek a poté rámec se simulací masky pomocí filtru, který byl vytvořený z tohoto rámce bez použití a s použitím okénkové funkce. Je vidět, že výsledek s použitím filtru vytvořeného s pomocí okénkové funkce mnohem více odpovídá skutečnému signálu. Na úrovni jednoho rámce jsme tedy simulaci vylepšili, pokud ale pokračujeme dál a vytvoříme podle dvojic rámců zprůměrovanou frekvenční charakteristiku, vliv okénkování je spíše negativní – frekvenční charakteristika i impulsní odezva tohoto filtru jsou „rozlítané“ a z výsledného filtru není příliš jasné, jaké frekvence má potlačovat. Výsledná simulace masky na nahrávce s větou zní spíše plechově.




Obrázek 12: Porovnání rámce z původních nahrávek a simulovaného rámce

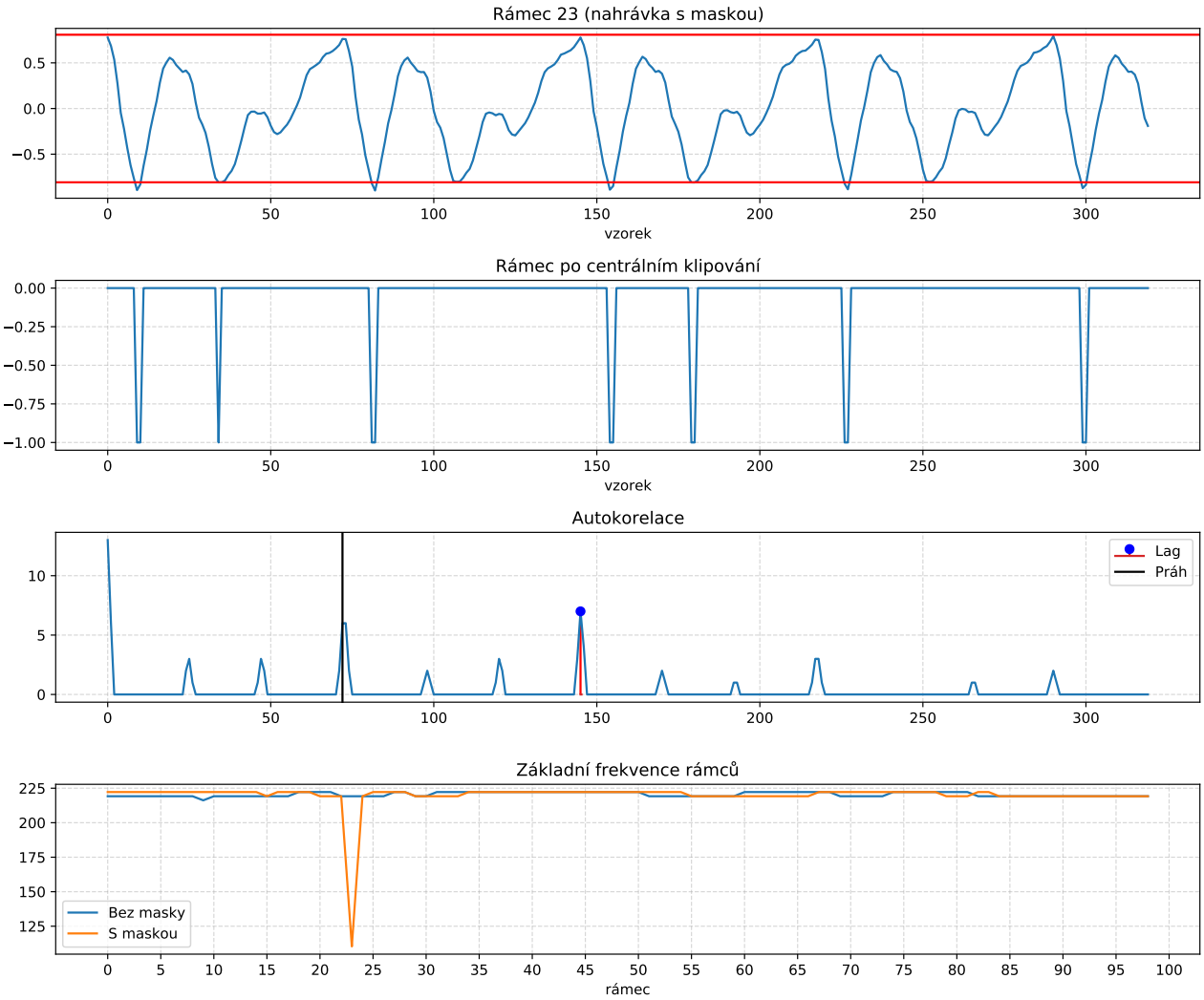
# 11 Detekce vícenásobného lagu

Pravděpodobně jsem měl štěstí s kvalitou vstupních nahrávek, protože problém s detekcí $n$-násobného lagu jsem vůbec nezaznamenal (viz obr. 3, základní frekvence všech rámců se drží na hodnotách odpovídajících třem po sobě jdoucím velikostem lagu). Problém se při použití 70% prahu klipování neprojevil dokonce ani s prahem frekvence 220 Hz, který je vzhledem k zjištěné $f_0$ velmi těsný. Problém se mi podařilo navodit až zvednutím prahu klipování na 90 %. Při použití těchto hodnot se vyskytla detekce dvojnásobného lagu v rámci 23 z nahrávky s maskou (viz obr. 13).

Problém jsem ošetřil velmi jednoduchým způsobem:

```
def fix_lags(arr):
    med = np.median(arr)
    thresh = 2
    wrong_indices = np.argwhere(np.abs(arr-med) > thresh)
    arr[wrong_indices] = arr[wrong_indices] // 2
```

Tento kód spočítá medián hodnot lagu napříč všemi rámci a poté zjišťuje, jestli se hodnota některé z položek liší od mediánu o více než nastavenou konstantní hranici `thresh`. Protože se pohybujeme na malém počtu vzorků a pracujeme s nahrávkou, která má v celém rozsahu přibližně konstantní základní frekvenci, tato hranice může být nastavena velmi nízko; předpokládáme, že se od sebe lagy nebudou velmi lišit – a chyba lagu ± 2 už by znamenala velmi podstatný rozdíl (jak bylo popsáno v kap. 4). Hranici jsem tedy nastavil na hodnotu 2. Funkce poté na pozice, kde byly nalezeny příliš rozdílné hodnoty lagu, nastaví poloviční hodnoty (čímž dvakrát zvýší zjištěnou $f_0$). To znamená, že tento přístup správně opraví pouze detekovaný dvojnásobný lag. V našem případě je to přístup zcela postačující, u jiných vstupních dat by tomu tak ale nemuselo být. Nabízí se také například nastavit lag na hodnotu mediánu.



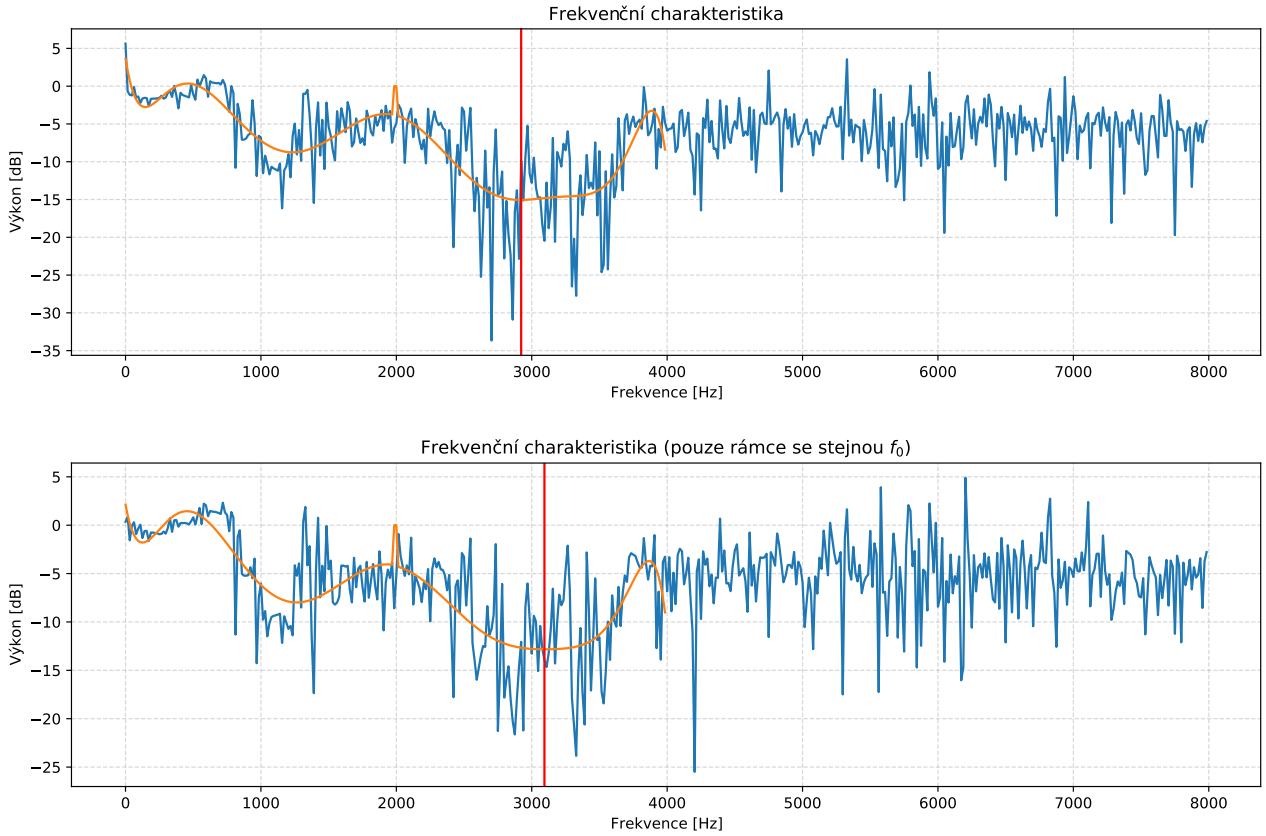

Obrázek 13: Klipování (90%) a autokorelace rámce 23, srovnání se základními frekvencemi ostatních rámců

# 12 Využití základní frekvence

V tomto úkolu jsem vytvořil filtr jen podle dvojic rámců, které měly stejnou základní frekvenci (respektive stejný lag). Takových rámců je v mých datech 60 z 99, celkem výrazně tak klesl počet vstupních dat pro tvorbu filtru. Provedení tohoto kroku je triviální:

```
lag_indices = np.argwhere(nomask_lags == mask_lags).flatten()
nomask_frames = nomask_frames[lag_indices]
mask_frames = mask_frames[lag_indices]
```

Obrázek 14 srovnává frekvenční charakteristiku původního filtru a filtru vytvořeného pouze z rámců se stejnou $f_0$. Můžeme vidět, že použití takových rámců zvýraznilo průběh frekvenční charakteristiky, zejména jejího tlumení na frekvencích kolem 1 100 Hz. Zároveň ale vidíme, že je frekvenční charakteristika výrazně zašuměnější, což přisuzuji právě snížení počtu vstupních rámců.



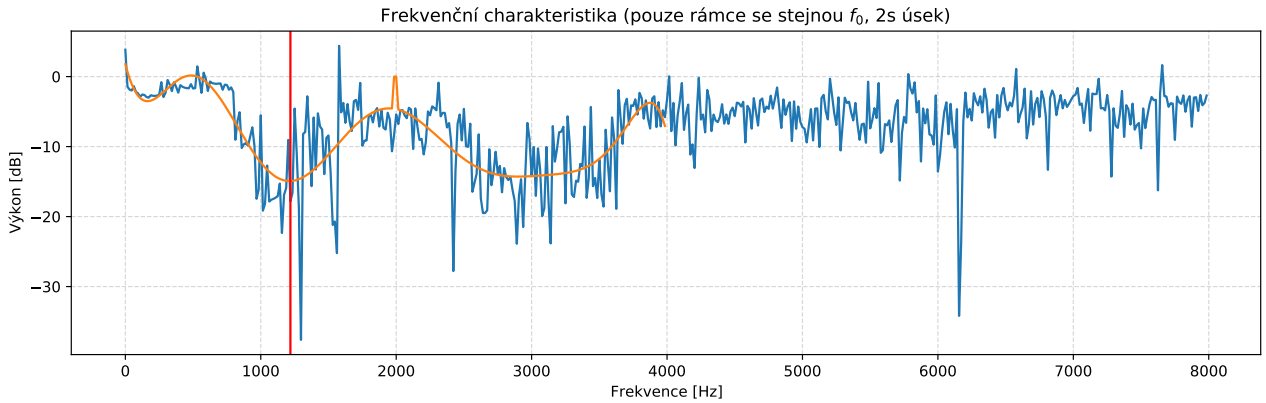

Obrázek 14: Frekvenční charakteristiky, proložené výstupem vyhlazovacího filtru

Pro zajímavost jsem tedy vzal spustil proces s dvojnásobným počtem rámců (z původních nahrávek jsem místo jedné sekundy vzal dvě). Počet dvojic rámců se stejnou $f_0$ se zvýšil na 101. Frekvenční charakteristika tohoto filtru je na obrázku 15.

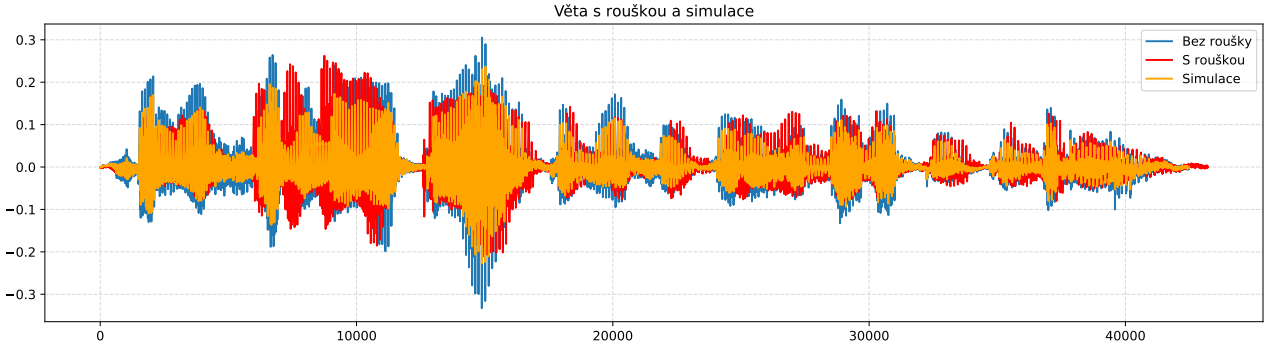
Na této frekvenční charakteristice je pěkně vidět, že zvýšení objemu vstupních dat omezí míru šumu – vyšší frekvence vypadají dokonce o něco hladčeji než ty z původního filtru. Zajímavé je, že v tomto filtru je také mnohem výraznější zádrž na frekvencích kolem 1 200 Hz, dokonce tak, že minimum průběhu vyhlazeného Savitzky-Golay filtrem se přesunulo právě na frekvenci 1 218 Hz.

Dalším dopadem tohoto kroku, který jsem zaznamenal, je, že při simulaci masky na nahrávce s větou tento filtr nezpůsobuje „obloučky“ v částech s nižšími hodnotami signálu, které jsem popisoval v kapitole 8. Můžeme to vidět na obrázku 16 (srovnejte s obr. 9).

Zajímavé jsou hodnoty korelace simulovaného a skutečného signálu s maskou. Zatímco při použití stejné sady rámců se stejnou $f_0$ se z původních 148 snížila na 140, při použití dvojnásobného počtu rámců se stejnou $f_0$ se o




Obrázek 15: Frekvenční charakteristika filtru vytvořeného podle dvojnásobného počtu rámců, proložená výstupem vyhlazovacího filtru



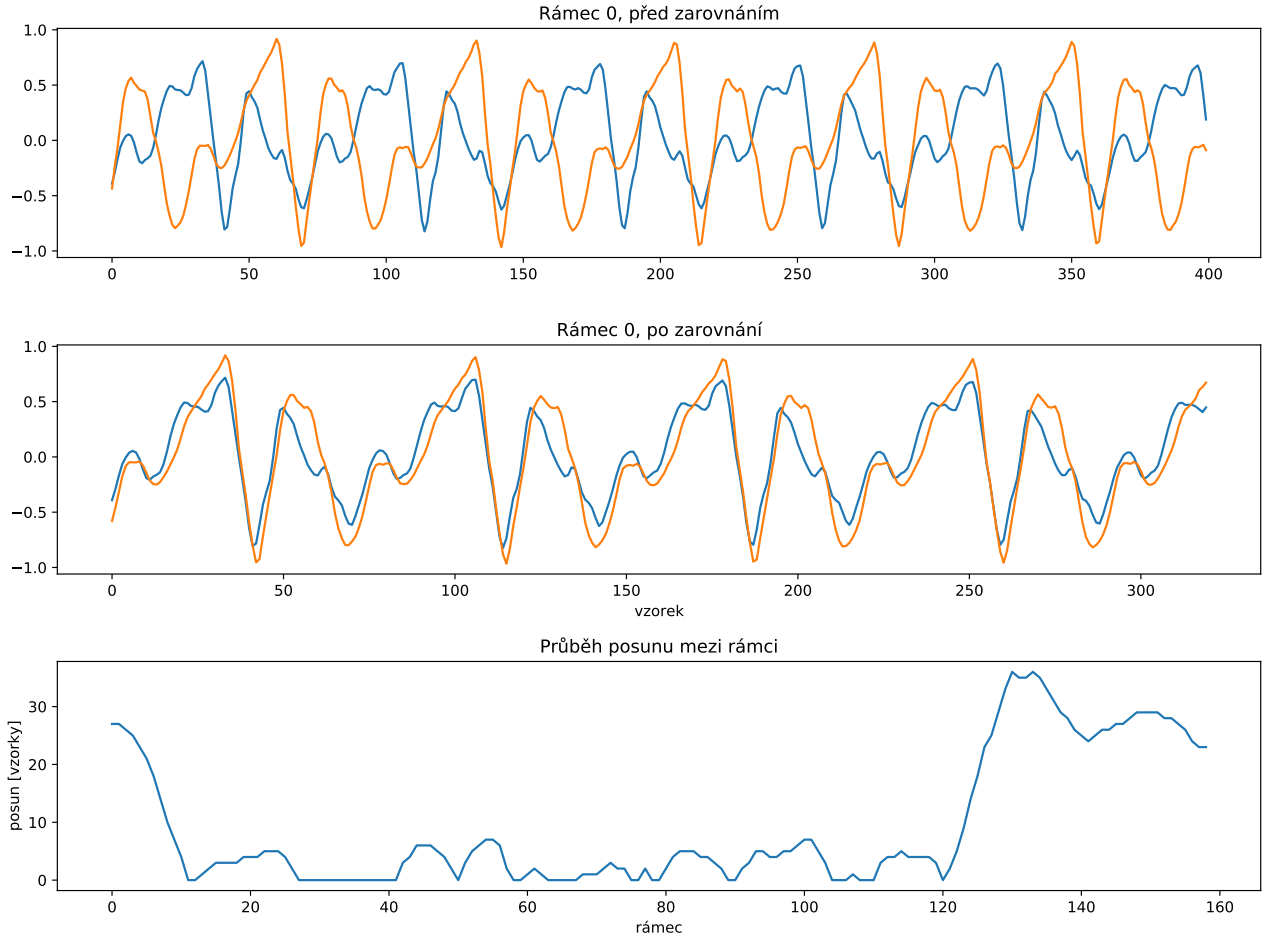

Obrázek 16: Simulace filtru na nahrávce věty

něco zvýšila na 152. To naznačuje, že snížení počtu vstupních rámců a výsledný šum má na celkový výsledek negativní dopad, pokud ale počet vstupních rámců zůstane přibližně stejný, použití rámců se stejnou $f_0$ náš filtr vylepší.

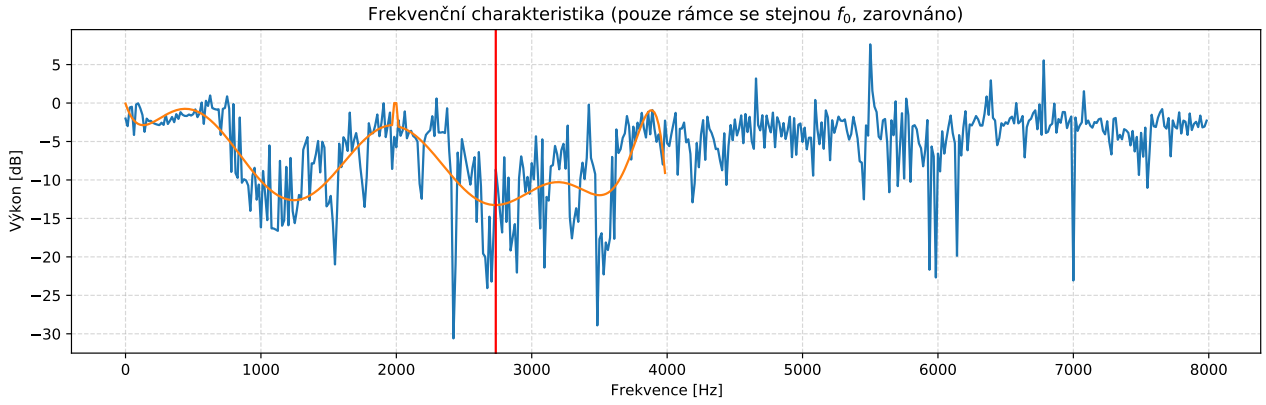
# 13 Zarovnání rámců

Zarovnávání rámců jsem prováděl ve stejném kódu jako omezování na dvojice rámců se stejnou $f_0$ a abych zamezil příliš velkému omezení počtu rámců, výpočty jsem prováděl na 2s úsecích vstupních nahrávek. Výsledný počet zarovnaných rámců se stejnou $f_0$ je pak 78. Na obrázku 17 jsou vidět grafy znázorňující zarovnání rámců a fázový posun mezi jednotlivými rámci. Je vidět, že největší posun mají rámce na začátcích úseků, ale u mnoha rámců je posun zanedbatelný už ve zdrojových nahrávkách – při výběru jiných dat by tak dopad zarovnání mohl být výraznější.

Zaujala mě frekvenční charakteristika filtru – jak je vidět na obrázku 18, charakteristika je o něco „vyprofilovanější“, je na ní více vidět, které části spektra potlačuje a které ne.




Obrázek 17: Zarovnání rámců




Obrázek 18: Frekvenční charakteristika filtru vytvořeného podle zarovnaných rámců, proložená výstupem vyhlazovacího filtru

# Reference


1. MALEK, Ayoub. *Signal framing* [online]. 2020-01-25 [cit. 2020-12-28]. Dostupné z: https://superkogito.github.io/blog/SignalFraming.html.
2. ČERNOCKÝ, Jan. *Zpracování řečových signálů – studijní opora* [online]. 2006-12-06 [cit. 2020-12-28]. Dostupné z: https://www.fit.vutbr.cz/study/courses/ZRE/public/opora/zre_opora.pdf.
3. WIKIPEDIA CONTRIBUTORS. *Savitzky–Golay filter — Wikipedia, The Free Encyclopedia* [online]. 2020 [cit. 2020-12-29]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Savitzky%E2%80%93Golay_filter&oldid=992823142.
4. *Understanding FFTs and Windowing* [online]. 2015 [cit. 2020-12-30]. Dostupné z: https://download.ni.com/evaluation/pxi/Understanding%20FFTs%20and%20Windowing.pdf.
5. *Understanding FFT Windows* [online]. 2003 [cit. 2020-12-30]. Dostupné z: https://www.egr.msu.edu/classes/me451/me451_labs/Fall_2013/Understanding_FFT_Windows.pdf.