ISS Projekt 2020/2021

Jakub Hlava (xhlava52)

4. ledna 2021

Úkol 1 a úkol 2

Informace o souborech s nahrávkami

Název souboru	Délka [vzorky]	Délka [s]
`maskoff_tone.wav`	51270	3.20
`maskon_tone.wav`	65016	4.06

Tabulka 1: Informace o nahrávkách tónu

Název souboru	Délka [vzorky]	Délka [s]
`maskoff_sentence.wav`	77987	4.87
`maskon_sentence.wav`	77276	4.83

Tabulka 2: Informace o nahrávkách věty

Postup

Získáno ze souborů příkazem `soxi ./audio/*`

Úkol 3

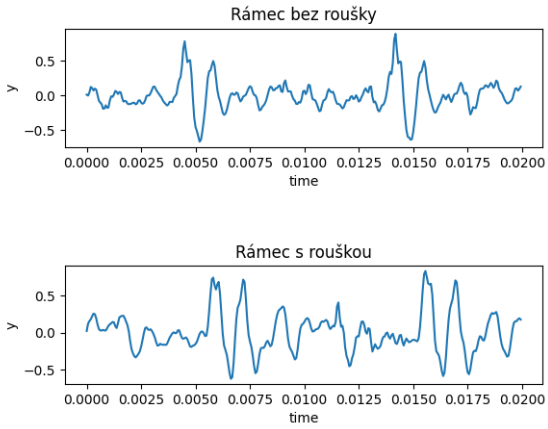
Výpočet velikosti rámce ve vzorcích

$$V_{Rvz} = D_R * F_v$$

Kde: V_{Rvz} je velikost rámce ve vzorcích
D_R je délka rámce v sekundách
F_v je vzorkovací frekvence signálu

V našem případě tedy konkrétně $0.02 * 16000 = 320$ vzorků na rámec.

Srovnání rámců



Obrázek 1: Srovnání druhého rámce nahrávky s rouškou a nahrávky bez ní

Metoda výběru úseků nahrávek

Při převodu jsem manuálně vybral úsek o délce 1s z nahrávky s rouškou a následně jsem prováděl korelaci všech možných sekundových úseků nahrávky bez roušky a vybral ten, který měl největší koeficient korelace.

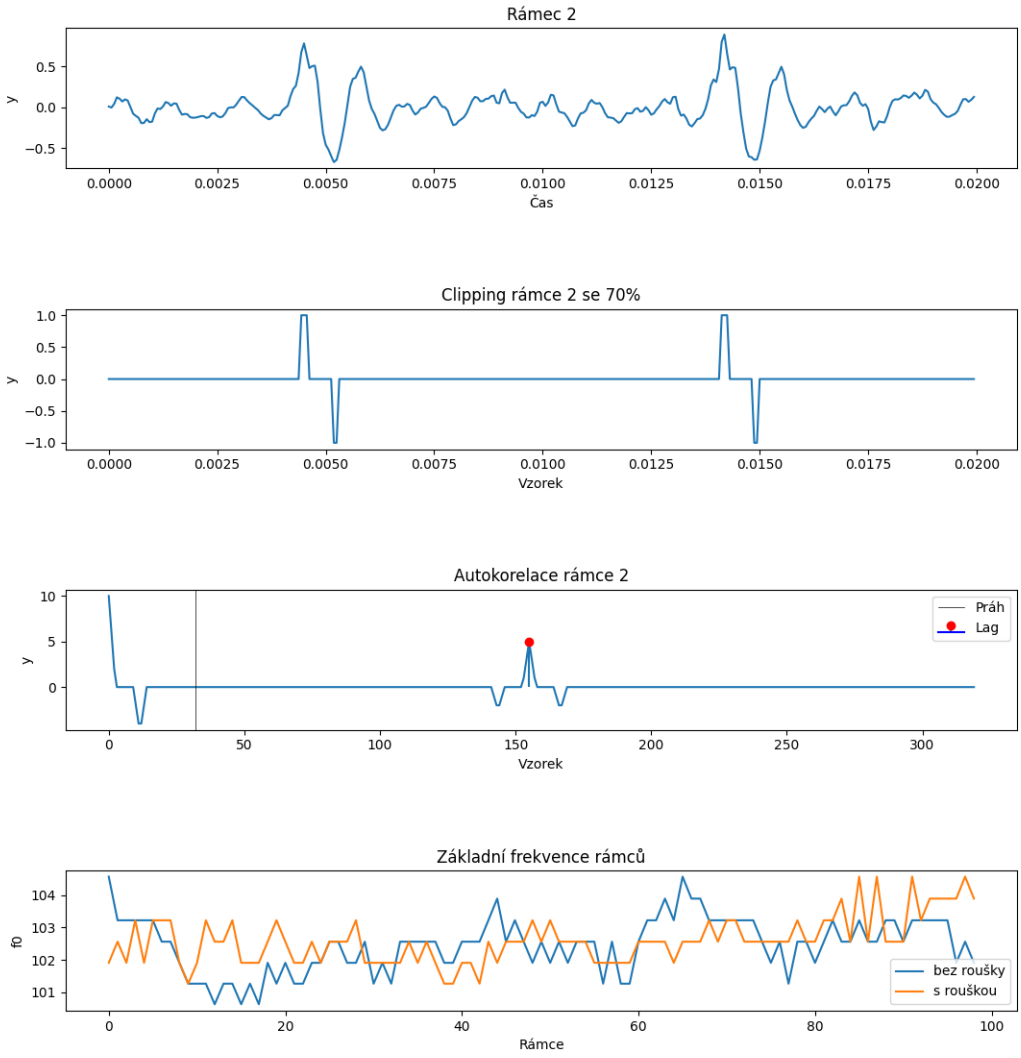
Postup

```
# Ruční výběr 1s vzorku signálu
hpon = np.array(maskon[16000:32000]) # 1.0 - 2.0s

#Výběr druhého vzorku pomocí korelace
cors = []
for i in range(0,len(maskoff)-16000):
    cors.append(np.correlate(hpon, maskoff[i:i+16000]))
startindex = cors.index(max(cors))
hpoff = np.array(maskoff[startindex:startindex+16000])

# Převod na floaty
hpoff = hpoff.astype(np.float)
hpon = hpon.astype(np.float)

# Ustřednění
hpoff -= np.mean(hpoff)
hpon -= np.mean(hpon)

# Normalizace
hpoff /= np.abs(hpoff).max()
hpon /= np.abs(hpon).max()

ramce_mon = list()
ramce_moff = list()

# rozčlenění na rámce
# 20ms rámce == 320 vzorků => 10ms posun = 160 vzorků
for i in range(99):
    ramce_mon.append(hpon[i*160:(i+2)*160])
    ramce_moff.append(hpoff[i*160:(i+2)*160])
```

Zdrojový kód 1: Vyběr, úprava a členění vtupního signálu

Úkol 4

Grafy výsledků operací nad rámci



Obrázek 2: Graf rámce, klipování, autokorelace a srovnání základních frekvencí nahrávek

Srovnání rozptylu a středních hodnot

Nahrávka	Rozptyl	Střední hodnota
Bez roušky	0.69487412	102.45811
S rouškou	0.52859397	102.60909

Tabulka 3: Srovnání středních hodnot (průměrů) a rozptylů základních frekvencí obou nahrávek

Postup

Klipování, autokorelace a výpočet základní frekvence

```
# Centrální klipování
clippedon = []
clippedoff = []

for r in ramce_moff:
    max70 = max([abs(x) for x in r])*0.7
    clippedoff.append(np.array([1 if x >= max70 else (-1 if x <= -max70 else 0) for x in r]))

for r in ramce_mon:
    max70 = max([abs(x) for x in r])*0.7
    clippedon.append(np.array([1 if x >= max70 else (-1 if x <= -max70 else 0) for x in r]))

clippedon = np.array(clippedon)
clippedoff = np.array(clippedoff)

prah = 32 # Práh pro lagy 500 MHz

# Autokorelace rámců
for r in clippedoff:
    lag = autocorrelate(r)[prah:].argmax()+prah
    freq = 1/lag * moff_sr # výpočet f0
    lagsoff.append(freq)

for r in clippedon:
    lag = autocorrelate(r)[prah:].argmax()+prah
    freq = 1/lag * mon_sr
    lagson.append(freq)
```

Zdrojový kód 2: Centrální klipování a výpočet základní frekvence

Vlastní implementace autokorelace rámců

```
def autocorrelate(sig):
    nulls = np.array([0 for x in range(sig.size)])
    b = np.concatenate((sig,nulls))
    results = []
    for i in range(sig.size):
        results.append(np.sum(sig*b[i:sig.size+i]))
    return np.array(results)
```

Zdrojový kód 3: Funkce pro autokorelaci

Výpočet průměrné hodnoty a rozptylu základní frekvence

Pro tyto výpočty jsem využil funkce `np.mean()` a `np.var()`

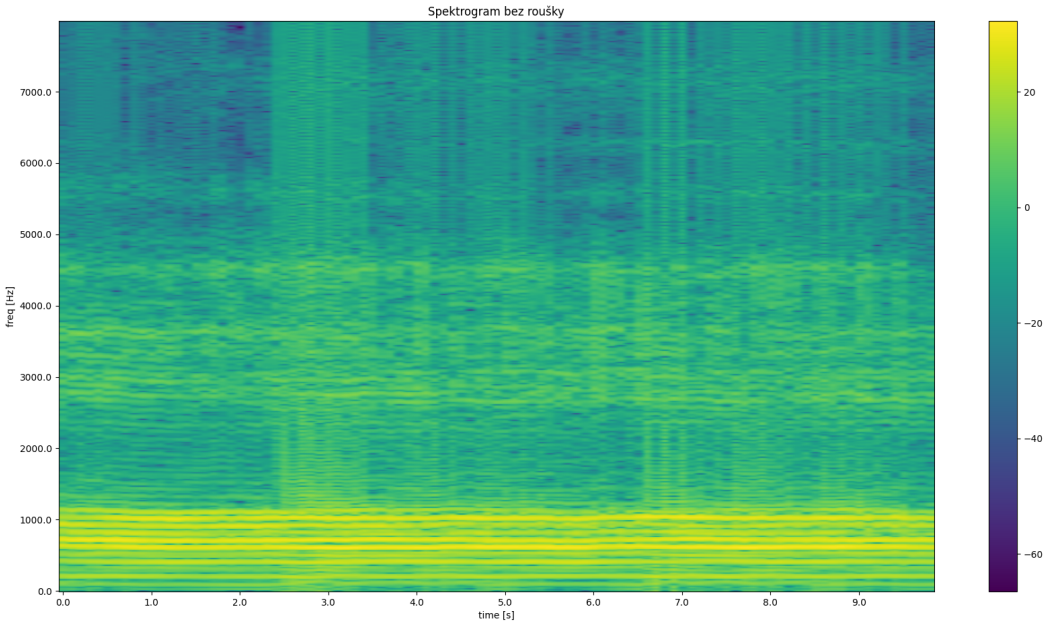
Úkol 5

Vlastní implementace DFT

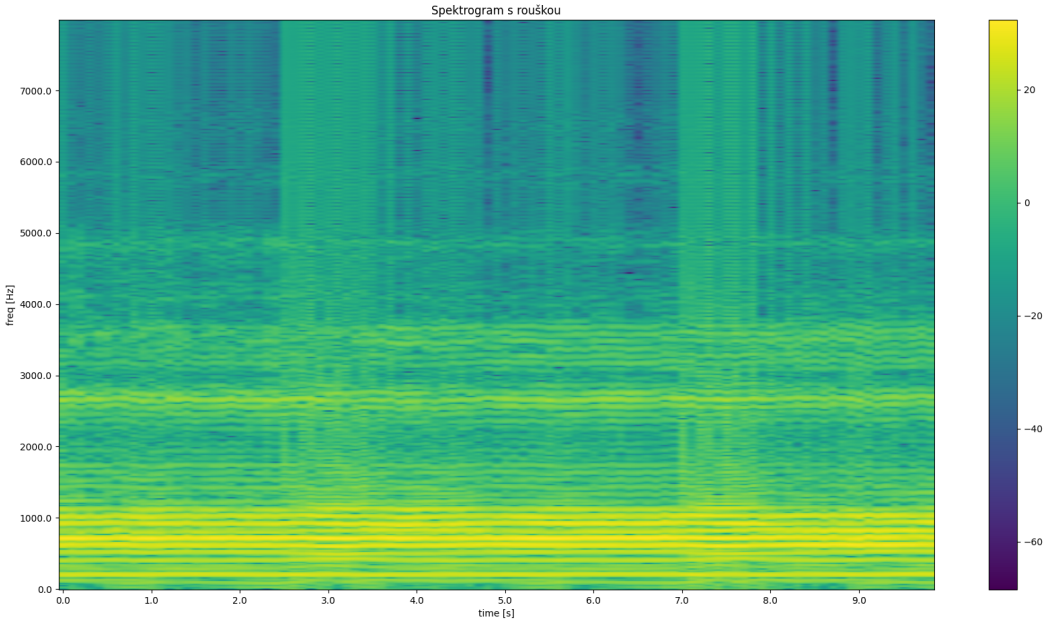
```
def myDFT(sig):
    result = []
    for k in range(sig.size):
        s = 0
        for n in range(sig.size):
            s += sig[n]*np.power(np.e,2*np.pi*(-1j)*k*n/sig.size)
        result.append(s)
    return np.array(result)
```

Zdrojový kód 4: Vlastní implementace DFT

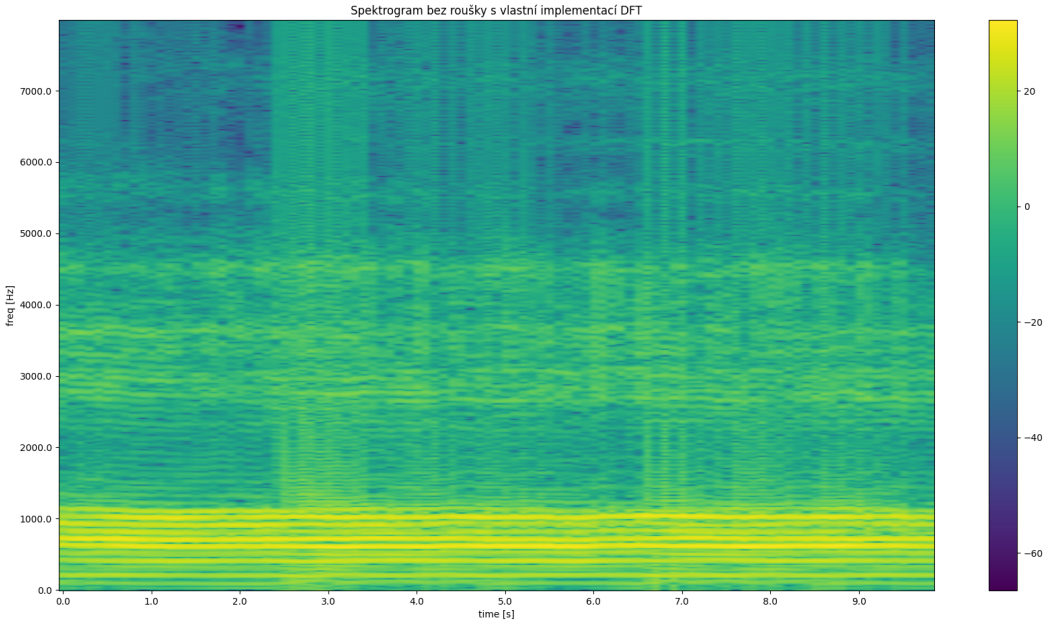
Spektrogramy nahrávek



Obrázek 3: Spektrogram nahrávky tónu bez roušky



Obrázek 4: Spektrogram nahrávky tónu s rouškou



Obrázek 5: Spektrogram nahrávky tónu bez roušky vypočítaný vlastní implementací

Srovnání vlastní a knihovní implementace DFT

Pro výpočty DFT jsem použil knihovní funkci numpy `np.fft.fft`. Moje implementace DFT, jak je vidět na obrázku 5, produkuje shodné koeficienty, pouze je mnohonásobně pomalejší, a to ze dvou důvodů - nejedná se o FFT implementaci a interpretovaný kód Pythonu je mnohem pomalejší než knihovní funkce.

Postup

Jak bylo zmíněno výše, pro výpočet DFT používám knihovní funkci `np.fft.fft`, volitelně (kvůli úspoře času) pro vytvoření srovnávacího spektrogramu i vlastní funkci `myDFT` (zdrojový kód 4). Protože je kód pro úkol 5 spojený s úkolem 13 a poměrně nepřehledný, ale v jádru jednoduchý, tak pouze ve stručnosti: spočítám DFT pro každý rámec, upravím hodnoty všech koeficientů dle vzorce ze zadání a na závěr vše vykreslím pomocí knihovní funkce `plt.imshow()`.

Úkol 6

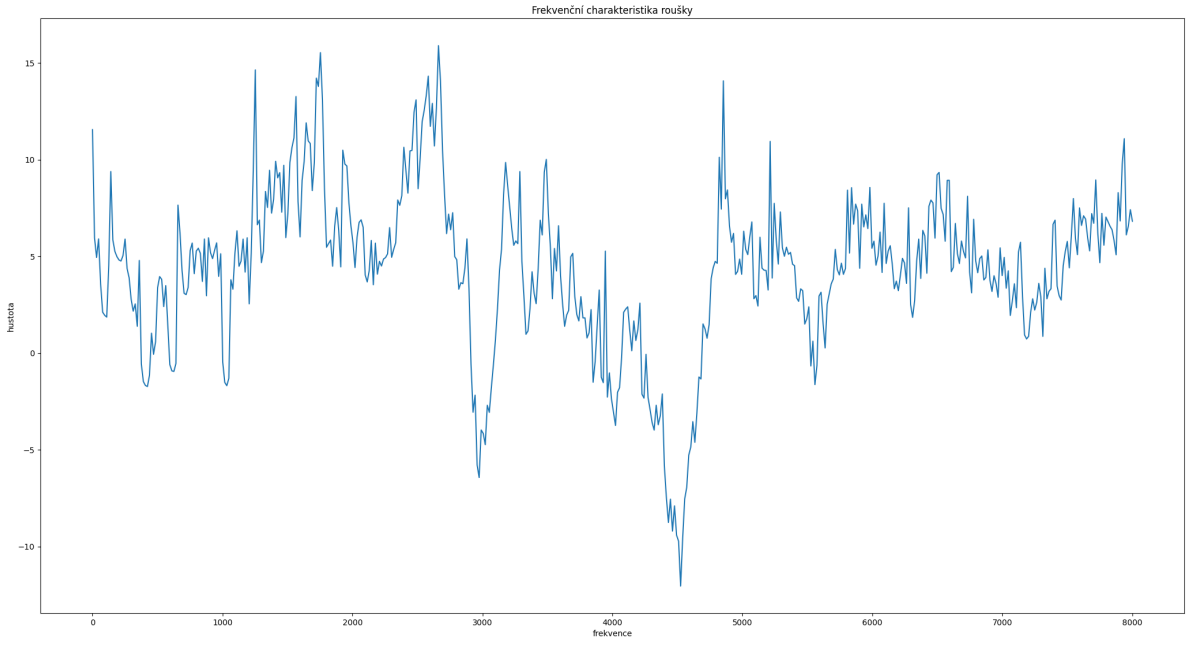
Vztah pro výpočet $H(e^{j\omega})$

Frekvenční charakteristiku spočítáme dělením všech koeficientů DFT všech rámců nahrávky s rouškou a nahrávky bez roušky, obecně tedy:

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

Kde: $H(e^{j\omega})$ je frekvenční charakteristika
A jsou koeficienty spektra nahrávky bez roušky
B jsou koeficienty spektra nahrávky s rouškou

Graf frekvenční charakteristiky roušky



Obrázek 6: Frekvenční charakteristika roušky

Postup

Výpočet frekvenční charakteristiky

```
H = []
mon = np.array(maskondft)
moff = np.array(maskoffdft)
for x in range(len(maskoffdft)): # Výpočet H
    H.append(mon[x]/moff[x])
H = np.array(H)

# zprůměrování absolutních hodnot komplexních koeficientů DFT
havg = np.average(np.abs(H),axis=0)

# úprava na výkonové spektrum před vykreslením
havgpow = 10*np.log10(np.power(np.real(havg[:512]),2))
```

Zdrojový kód 5: Výpočet $H(e^{j\omega})$

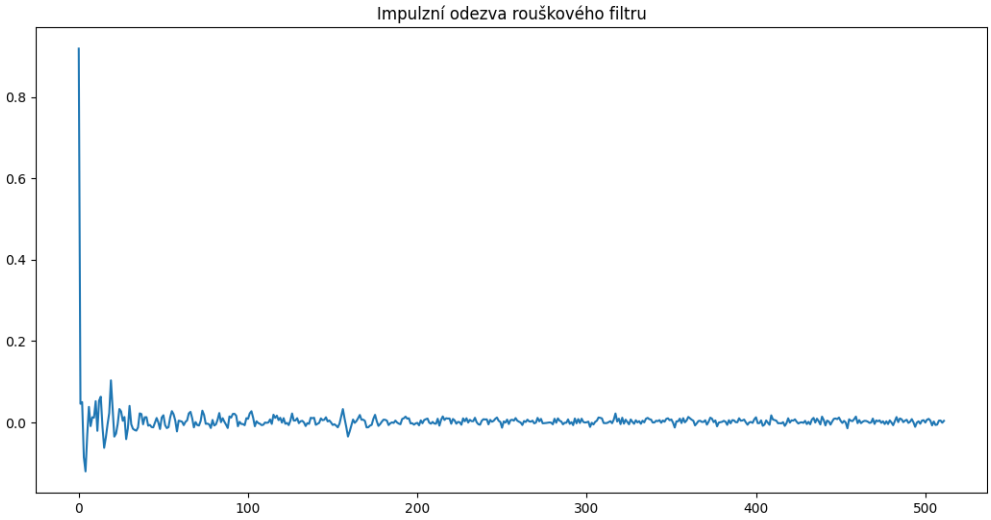
Úkol 7

Vlastní implementace IDFT

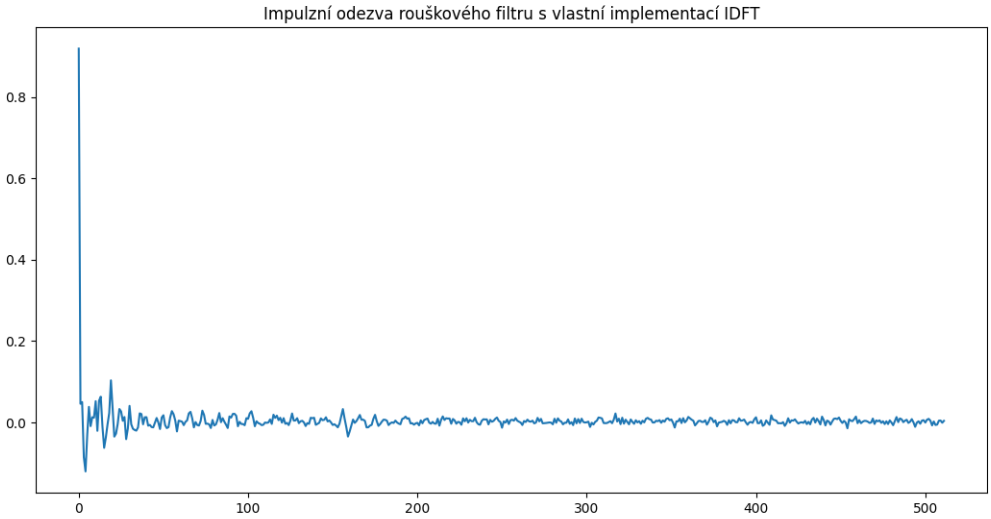
```
def myIDFT(sig):
    result = []
    for k in range(sig.size):
        s = 0
        for n in range(sig.size):
            s += sig[n]*np.power(np.e,2*np.pi*(1j)*n*k/sig.size)
        s /= sig.size
        result.append(s)
    return np.array(result)
```

Zdrojový kód 6: Vlastní implementace IDFT

Impulzní odezva roušky



Obrázek 7: Impulzní odezva roušky



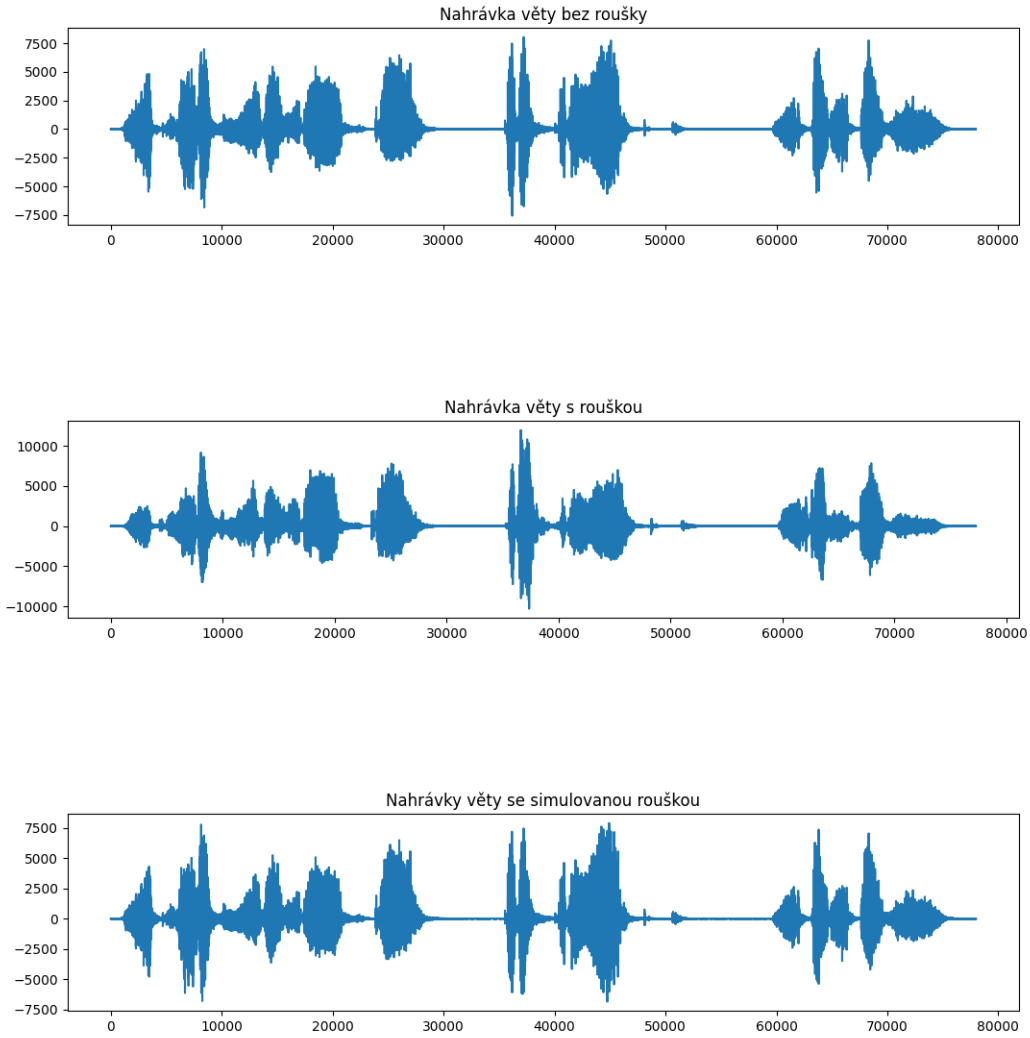
Obrázek 8: Impulzní odezva roušky s vlastní implementací IDFT

Srovnání vlastní a knihovní implementace IDFT a postup

Podobně jako u DFT jsem využil knihovní funkci Numpy, tentokrát ale `np.fft.ifft`, volitelně poté pro srovnání počítám IDFT vlastní implementací. Výsledky knihovní a vlastní funkce jsou shodné, nicméně knihovní funkce jich dosahuje v řádově nižším čase, proto ve výchozím stavu používám pro úsporu času ji namísto vlastní implementace.

Úkol 8

Grafy nahrávek



Obrázek 9: Grafy vět

Zhodnocení grafů

Simulovaný signál je vizuálně i poslechově podobný spíše nahrávce bez roušky, protože filtr (a ani rozdíl v originálních nahrávkách) nemá příliš intenzivní efekt. Největší podobnost simulovaného s „rouškovým“ signálem je slyšet ve znění sykavek.

Postup

Aplikace filtru a export do WAV

```
sf = scipy.signal.lfilter(odezva.real,[1.0],sentence) # aplikování filtru na nahrávku věty
sf = np.real(sf) # odstranění případné komplexní složky
sf = np.int16(sf) # převod na int16 pro export wav
wavfile.write("./audio/sim_maskon_sentence.wav",s_sr,sf) # export wav
```

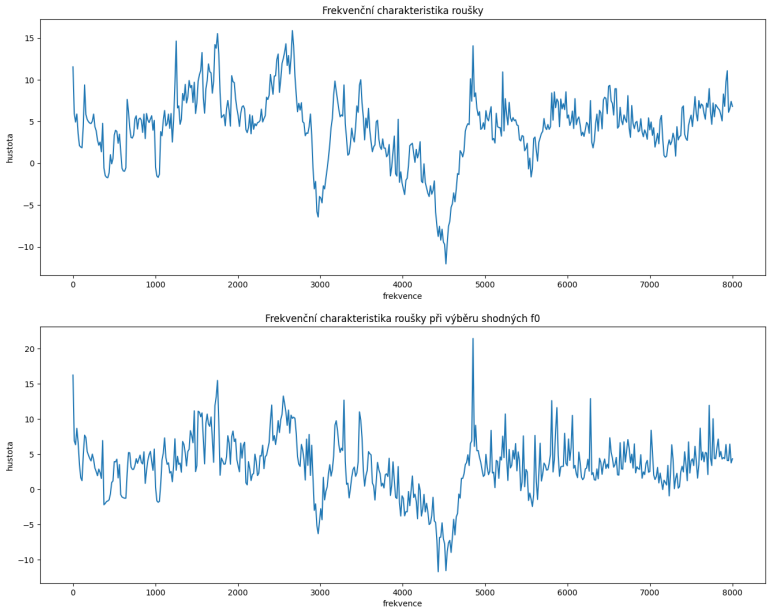
Úkol 9

Závěr prvotního odhadu

Jak jsem zmínil v předchozím bodě, moje řešení nemá příliš výrazný efekt na výsledném signálu. Při kontrole poslechem jsou rozdíly stěží zaznamenatelné, ale ani rozdíly mezi originálními nahrávkami s rouškou a bez ní nejsou příliš velké. Domnívám se, že pouze na hlásce „á“ není u mojí roušky tak výrazný rozdíl ve výsledné nahrávce jako např. u sykavek.

Úkol 13

Srovnání frekvenčních charakteristik



Obrázek 10: Frekvenční charakteristiky

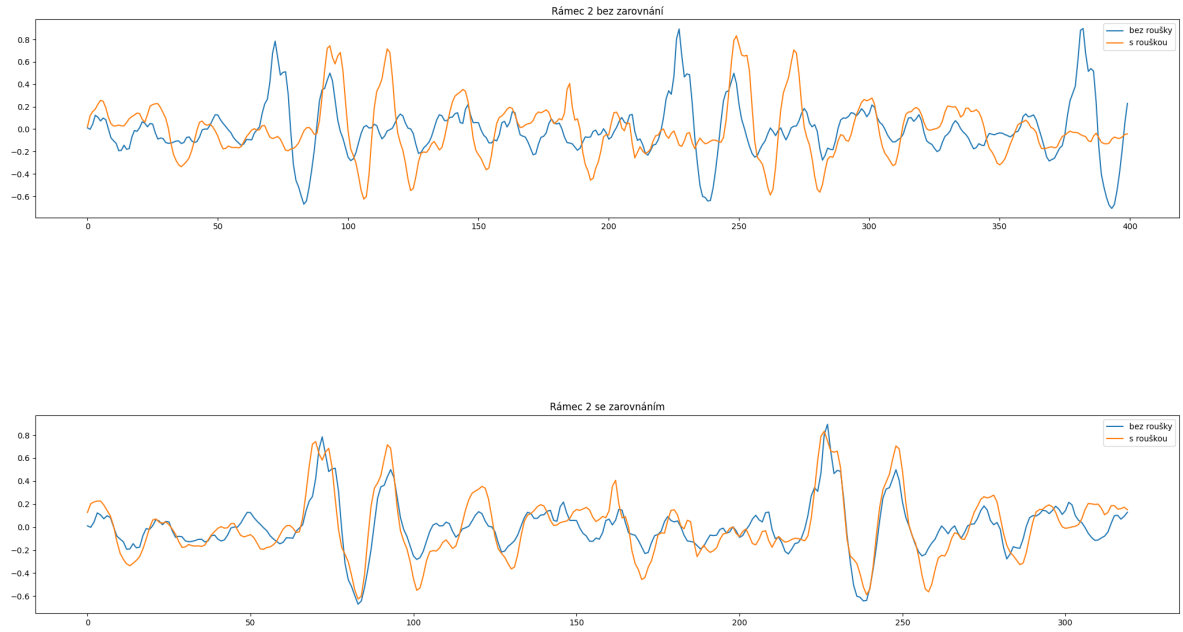
Tímto výběrem eliminujeme vliv rozdílných základních frekvencí, které i přes veškerou snahu jsou u nahrávek stále přítomny, na filtr. Zároveň ale průměrujeme méně hodnot, protože rámců se shodnou základní frekvencí není mnoho.

Postup

Používám kód shodný s úkolem 5, pouze s rozdílem, že do seznamu DFT rámců pro další zpracování přidávám pouze rámce nahrávky s rouškou a bez ní, které mají shodou základní frekvenci.

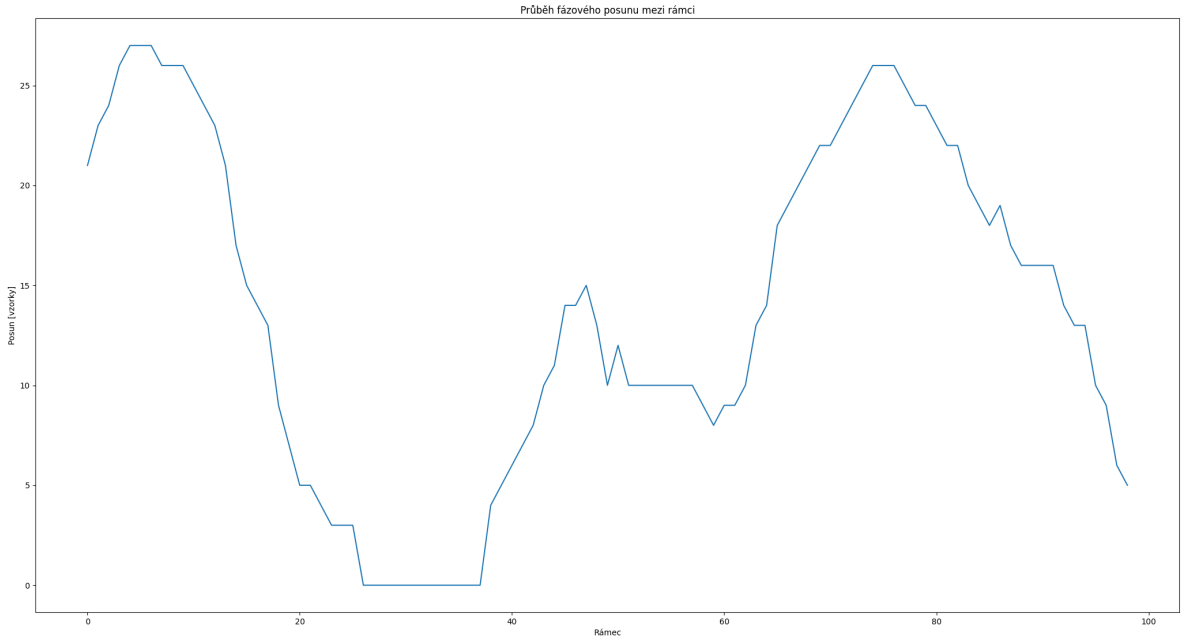
Úkol 15

Srovnání rámců při zarovnávání



Obrázek 11: Zarovnávání rámců

Průběh posunu rámců



Obrázek 12: Fázový posun rámců

Postup

```
    # Nové rámce - 25ms
    dr_off = list()
    dr_on = list()
    for i in range(99):
        dr_on.append(hpon[i*160:int((i+2.5)*160)])
        dr_off.append(hpoff[i*160:int((i+2.5)*160)])

    cutoff = []
    cuton = []
    posuny = []

    for i in range(len(dr_off)):
        # klipování na 70% kvůli korelaci
        clipon = clip(dr_on[i])
        clipoff = clip(dr_off[i])

        #korelace
        w = np.concatenate((clipon,[0 for x in range(clipon.size)]))
        posunoffon = []
        for j in range(clipon.size):
             posunoffon.append(np.correlate(clipoff,w[j:j+clipon.size])[0])
        w = np.concatenate((clipoff,[0 for x in range(clipoff.size)]))
        posunonoff = []
        for j in range(clipoff.size):
            posunonoff.append(np.correlate(clipon,w[j:j+clipoff.size])[0])
        #nalezení posunu ve vzorcích
        posunoff = np.array(posunoffon).argmax()
        posunon = np.array(posunonoff).argmax()

        # srovnání rámců
        if posunoff < posunon:
            posuny.append(posunoff)
            if posunoff == 0:
                cutoff.append(dr_off[i])
                cuton.append(dr_on[i])
            cutoff.append(dr_off[i][:-posunoff])
            cuton.append(dr_on[i][posunoff:])
        else:
            posuny.append(-posunon)
            if posunon == 0:
                cutoff.append(dr_off[i])
                cuton.append(dr_on[i])
            cutoff.append(dr_off[i][posunon:])
            cuton.append(dr_on[i][:-posunon])

    # ořez na 20ms rámce
    for x in range(len(cutoff)):
        cutoff[x] = cutoff[x][:320]
        cuton[x] = cuton[x][:320]
```

Zdrojový kód 7: Klipování, korelace a zarovnání rámců

```
    def clip(frame):
        max70 = max([abs(x) for x in frame])*0.7
        return np.array([1 if x >= max70 else (-1 if x <= -max70 else 0) for x in frame])
```

Zdrojový kód 8: Pomocná funkce pro klipování rámců

Poznámky ke kódu

Kód byl spouštěn a testován v prostředí pipenv, přiložený Pipfile lze využít k automatické instalaci závislostí projektu.
Závislosti: `numpy==1.19.3, scipy, matplotlib`
Testováno na Python 3.8