Fakulta informačních technologií
Vysoké učení technické v Brně

Signály a systémy

Projekt 2021/22

Maksim Tikhonov (xtikho00) 7. ledna 2022

1 Úvod

1.1 Struktura projektu

Ve adresáři `/src` se nachází soubor `project.ipynb`, který obsahuje zdrojový kod analyzující a vyčisťující pomocí filtrů vstupní signál ze souboru `https://www.fit.vutbr.cz/study/courses/ISS/public/proj2021-22/signals/xtikho00.wav` a generující grafy použité v tomto protokolu.

Ve adresáři `/audio` se nacházejí audio soubory, které jsou výsledky práce programu.

1.2 Použité prostředí

Projekt byl implementován v jazyce **Python** na základě platformy **Google Colab**.

2 Řešení projektu

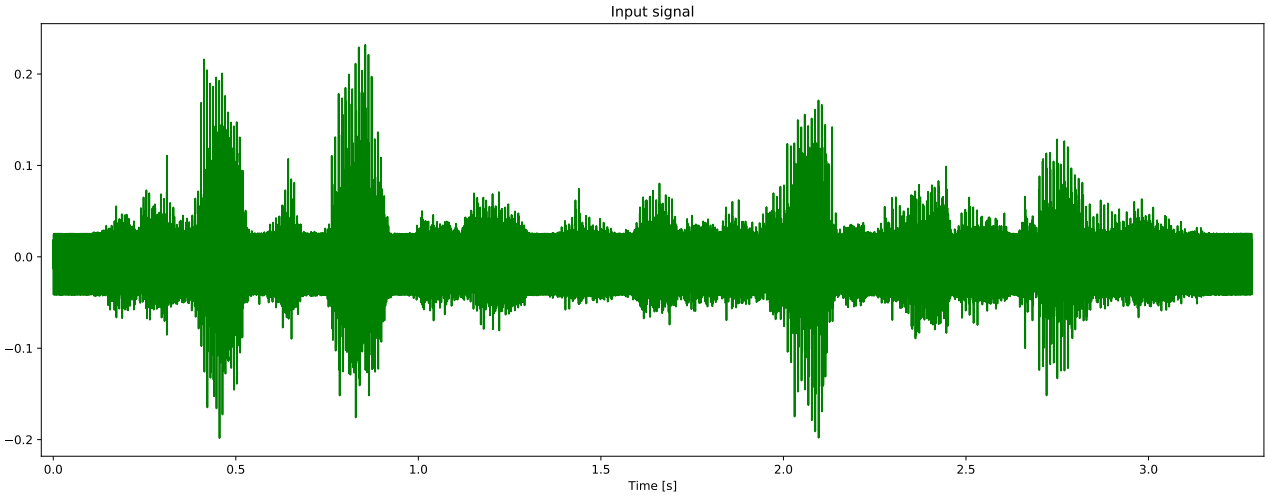
2.1 Standardní zadání

2.1.1 Základy

Prvotní analýza vstupního signálu (stanovení délky v sekundách, nalezení max. a min. hodnot signálu) byla implementována pomocí základních funkcí **Pythonu** `.min` a `.max` pro extrémní hodnoty signálu a dělením délky signálu ve vzorcících vzorkovací frekvencí pro nalezení délky signálu v sekundách.

Data	Délka		Hodnota		
Soubor	[s]	[ ]	Min.	Max.	Stř.
xtikho00.wav	3.28325	52532	-0.1983642578125	0.23193359375	-2.5046350096726756e-5

Tabulka 1: Základní údaje o signálu



Obrázek 1: Vstupní signál

2.1.2 Předzpracování a rámce

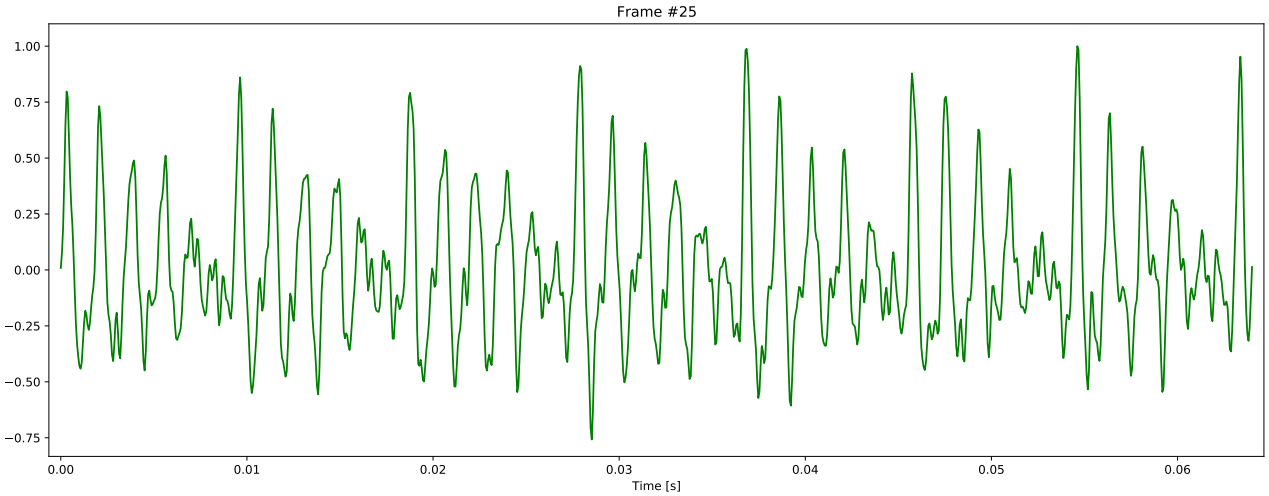
Předzpracování signálu (vyloučení stejnosměrného proudu a převod do rozsahu -1 až +1) bylo provedeno prostřednictvím odečítání jeho střední hodnoty a dělením maximém jeho absolutní hodnoty.

Pak pro **rozdělení původního signálu do rámců**, na jeho konec byly přidány vzorky s hodnotou nula tak, aby jeho délka byla dělitelná délkou rámců - 1024.

Pro generování matic rámců byla použita funkce `np.tile`. Vyšlo mi celkem 102 rámců.

```
def frame_signal ( signal = None, frameLength = 1024, overlapLength = 512 ) :

  signalLength = len(signal)
  stepLength = frameLength - overlapLength

  numberOfFrames = np.abs(signalLength - overlapLength) // np.abs(frameLength - overlapLength) + 1
  restLength = np.abs(signalLength - overlapLength) % np.abs(frameLength - overlapLength)

  addedZeroSignalLength = stepLength - restLength
  addedZeroSignal = np.zeros(addedZeroSignalLength)
  signal = np.append(signal, addedZeroSignal)

  index1 = np.tile(np.arange(0, frameLength), (numberOfFrames, 1))
  index2 = np.tile(np.arange(0, numberOfFrames * stepLength, stepLength), (frameLength, 1)).T
  indices = index1 + index2

  return signal[indices.astype(np.int32, copy=False)]
```

Po prohlednutí grafů všech těchto rámečků jsem si vybral ten nejvíce **znějící rámec číslo 25** reprezentující součást fonému `<a>` ve slově `dark`. Kritérii nejvhodnější znělosti jsou periodická příroda samotného signálu a to, že zvuk signálu reprezentuje nějakou samohlásku.



Obrázek 2: Rámec číslo 25

2.1.3 DFT

Diskrétní Fourierova Transformace je implementována dle vzorku:

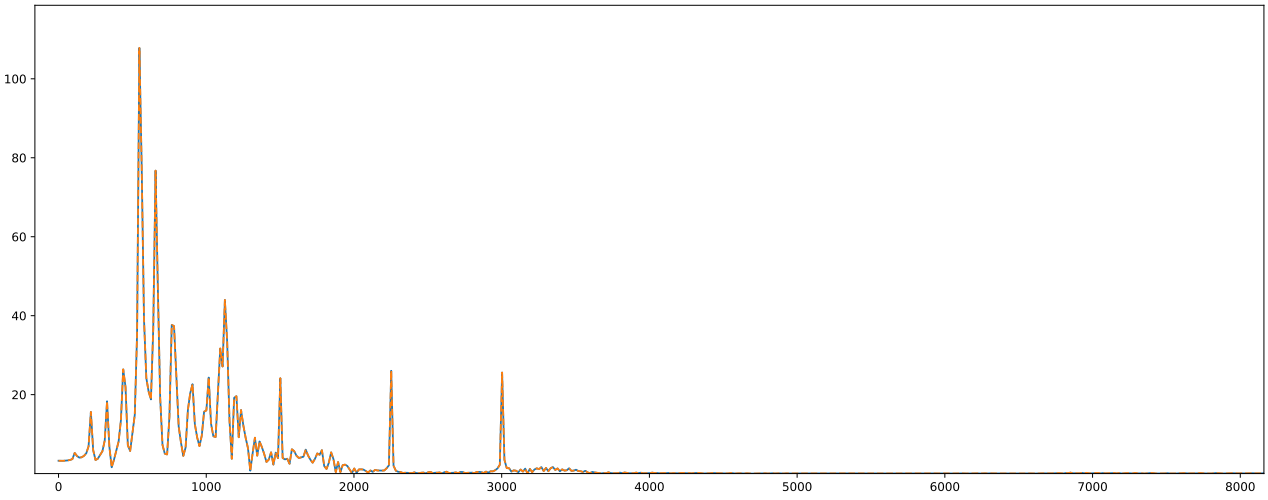
$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi nk}{N}}$$

$$\begin{bmatrix} X[0] \\ X[1] \\ \vdots \\ X[k-1] \end{bmatrix} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & e^{-j\frac{2\pi(N-1023)(k-1023)}{N}} & \dots & e^{-j\frac{2\pi(N-1023)(k-1)}{N}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & e^{-j\frac{2\pi(N-1023)(k-1)}{N}} & \dots & e^{-j\frac{2\pi(N-1)(k-1)}{N}} \end{bmatrix} \cdot \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix}$$

Funkce `myDFT` generuje matice tvaru k × N, kde N = k = 1024, a vektorově násobí ji původním signálem (rámcem). Výsledkem je jednorozměrný vektor koefficientů DFT.

```
def myDFT ( signal = None, numberOfSamples = 1024 ) :

  # generate matrix
  serieDeFourier = np.fromfunction( lambda n, k : np.exp(-1j * 2 * np.pi * n * k / (numberOfSamples)),
                        (numberOfSamples, numberOfSamples),
                        dtype = "complex128" )

  # dot product
  return np.dot(serieDeFourier, signal)
```

Ohledně rychlosti funkce, myslím, že lze říci, že je dostatečně rychlá (131 ms) - nejsou tam žádné cykly, ale jenom násobení vektorů. Očevidně, že nejde o žádné rychlostní srovnání s algoritmem `FFT` z knihovny `NumPy` (131 ms vs. 16.3 µs). Zase dle funkce `np.allclose` výsledky jsou přibližně stejné. Níže je graf, který představuje výsledky těchto dvou algoritmů (rozdíl není viditelný)



Obrázek 3: DFT

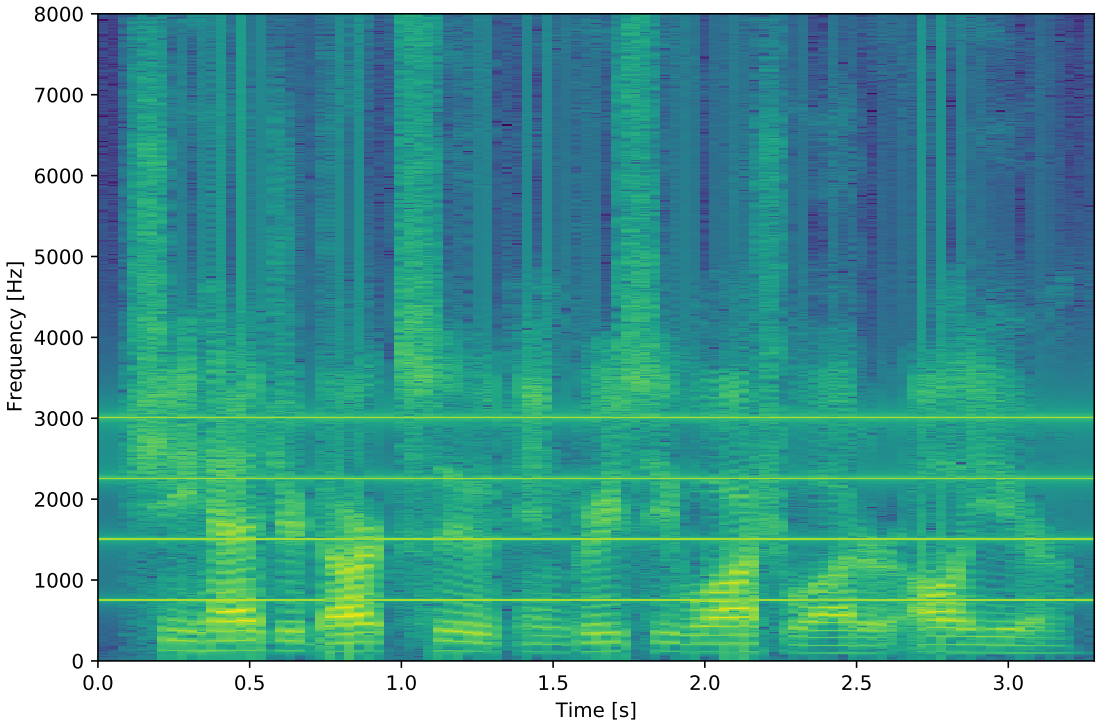
2.1.4 Spektrogram

Pro výpočet **spektrogramu** signálu a pro správné zobrazení času a frekvence na osách **spektrogramu** byla implementována funkce `spectrogram`, která vrací pole času `t` frekvence `f` a `první polovinu koeficientů DFT` (jsou symetrické vzhledem ke středu, nepotřebujeme zobrazovat signál dvakrát)

```
def spectrogram ( signal = None, frameLength = 1024, overlapLength = 512, sRate = 16000 ) :

  frames = frame_signal( signal, plotGraph = 'No' )
  dftCoefficients = np.array(list(map( np.fft.fft, frames )))

  dftCoefficients = dftCoefficients.T
  dftCoefficients = dftCoefficients[:len(dftCoefficients) // 2]

  spectreCoefficients = 20 * np.log10(np.abs(dftCoefficients))
  t = np.linspace(0, len(signal)/sRate, spectreCoefficients.shape[1])
  f = np.linspace(0, 8000, spectreCoefficients.shape[0])

  return t, f, spectreCoefficients
```

Funkce utilizuje předem definovanou funkce `frame_signal` pro rozdělení vstupního signálu do rámců, které pak budou konvertovány na `koeficienty DFT`. Potom z výše uvedených důvodů useká získané `pole DFT koeficientů` a upravuje jeho podle vzorku

$$P[k] = 10 \log_{10} |X[k]|^2 = 20 \log_{10} |X[k]|$$



Obrázek 4: Spektrogram původního signálu

2.1.5 Určení rušivých frekvencí

Ze spektrogramu původního signálu je jasně vidět frekvence čtyřech cosinusovek, můj předpoklad byl následující:

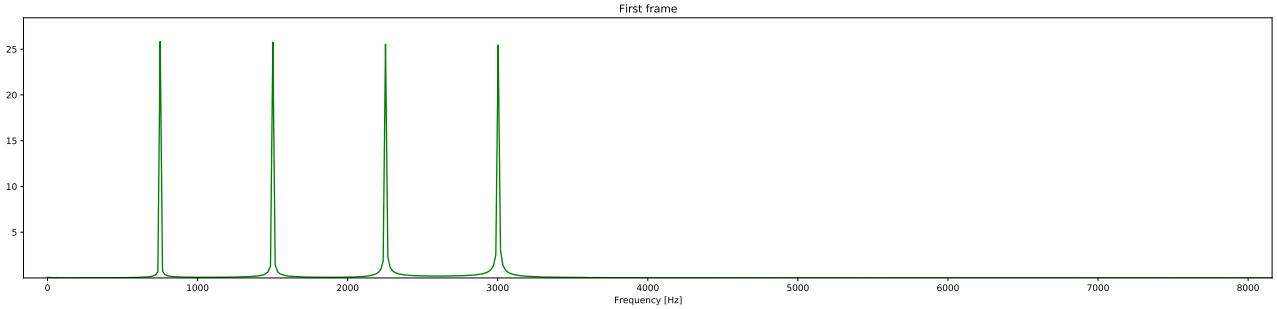
$f_1 = f_1 = 750$ Hz ≈ 48. koeficient
$f_2 = 2f_1 = 1500$ Hz ≈ 96. koeficient
$f_3 = 3f_1 = 2250$ Hz ≈ 144. koeficient
$f_4 = 4f_1 = 3000$ Hz ≈ 196. koeficient

Pro ověření této teorie jsem implementoval funkce `find_bad_frequencies`. Poprve jsem zkusil to ověřit pomocí vyhledavání indexů s nejmenšími amplitud hodnot uvnitř jednotlivých koeficientů spektra (dle předpokladu museli být 48, 96, 144, 192). Získané indexy můj předpoklad potvrdily jen částečně:
`[192 144 96 191 193 145 48 196]`

```
def find_bad_frequencies ( signal = None, sRate = 16000 ) :

  _, _, spectre = spectrogram( signal )

  maxs = spectre.max(axis = 1)
  mins = spectre.min(axis = 1)
  differences = maxs - mins
  bad_indeces = differences.argsort()[:8]

  ...
```

Druhou a mnohem přesnější metodou bylo získání frekvencí cosinusovek z rámcu, který neobsahuje žadné jiné frekvence (řeč). Takovým rámcem je první (nulový). Pomocí **DFT** získáme to na jakých frekvencích je v tomto rámci signál. Po useknutí poloviny koeficientu (nechceme získat stejné frekvence dvakrát) najdeme pomocí `np.argpartition` indexy (a tedy i frekvence) cosinusovek. Výsledek odpovídá mému předpokladu: `[48 96 144 192]`

```
  ...

  frames = frame_signal( signal, plotGraph = 'No' )
  dftCoefficients = np.abs(np.fft.fft( frames[0] ))

  plot_single_signal( dftCoefficients, xSize = 20, ySize = 5, plotLabel = 'First frame',
                      xLabel = 'Frequency [Hz]', DFT = 'yes', color = 'g' )

  dftCoefficients = dftCoefficients[:len(dftCoefficients)//2]
  badIndeces = np.argpartition(dftCoefficients, -4)[-4:]

  # Sort by value
  badIndeces[0], badIndeces[1], badIndeces[2], badIndeces[3] = \
  badIndeces[1], badIndeces[2], badIndeces[3], badIndeces[0]

  return badIndeces * 16000 // 1024
```

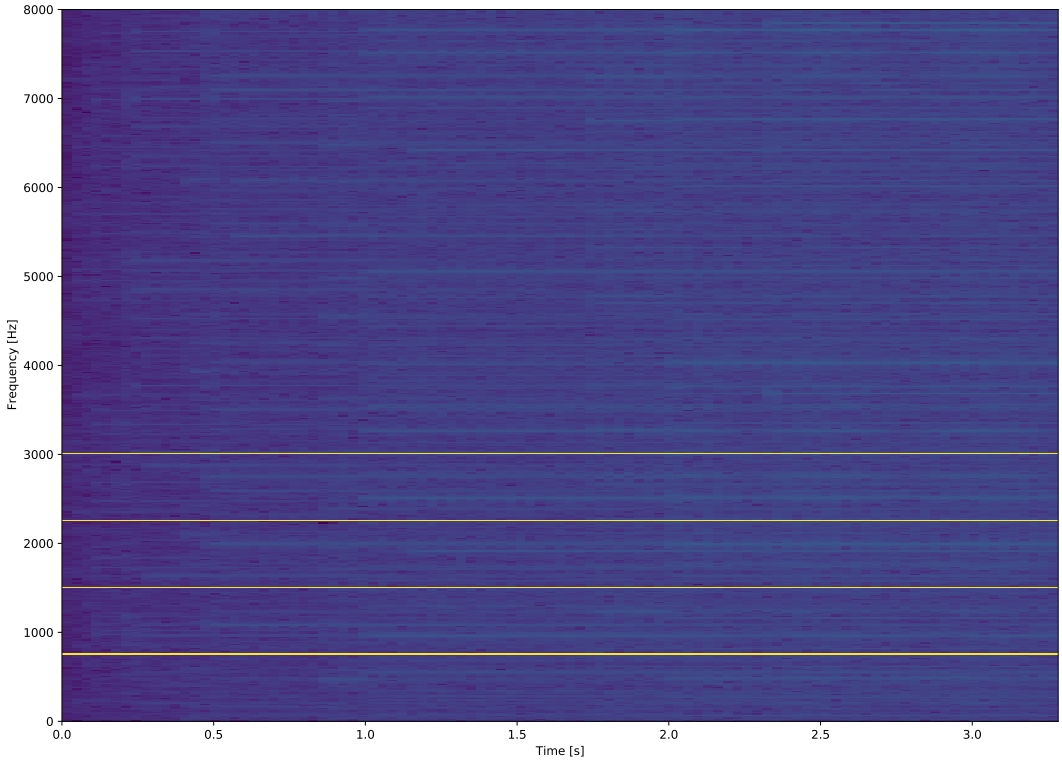


Obrázek 5: První rámec

2.1.6 Generování signálu

Generovaný signál se skládá z čtyřech cosinů s frekvencemi f_1, f_2, f_3, $f_4 = 750$ Hz, 1500 Hz, 2250 Hz, 3000 Hz. Jednotlivé cosinusovky byly vygenerovány funkcí `np.cos` a výsledný signal je jejich součtem. Frekvence vygenerovaného signálu korespondují frekvencím cosinů z původního signálu.

Vygenerovaný signál je uložen v souboru `audio/4cos.wav`



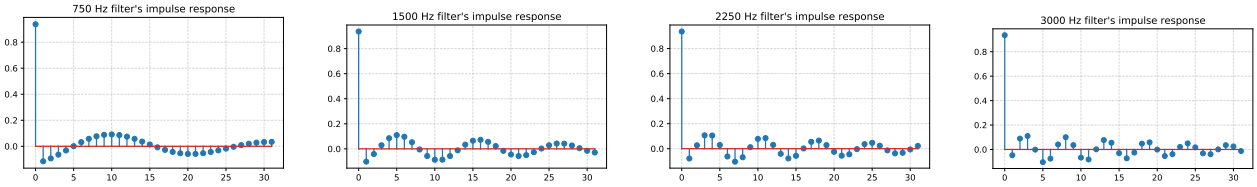
Obrázek 6: Vygenerovaný signál

2.1.7 Čisticí filtr

Čisticí filtr byl implementovan jako 4 pásmové zádrží.

Filtry		0	1	2	3	4	5	6	7	8
H_{f_1}	b	0.9378	-7.1796	24.3628	-47.8379	59.4337	-47.8379	24.3628	-7.1795	0.9378
	a	1.0000	-7.5328	25.1521	-48.5983	59.4157	-47.0628	23.5878	-6.8411	0.87949
H_{f_2}	b	0.9366	-6.2302	19.2874	-35.9200	43.8646	-35.9200	19.2874	-6.2302	0.9366
	a	1.0000	-6.5430	19.9245	-36.5014	43.8495	-35.3254	18.6614	-5.9308	0.8772
H_{f_3}	b	0.9361	-4.7510	12.7868	-21.9019	26.1276	-21.9019	12.7868	-4.7510	0.9361
	a	1.0000	-4.9915	13.2121	-22.2580	26.1170	-21.5354	12.3681	-4.5209	0.8763
H_{f_4}	b	0.9358	-2.8650	7.0326	-10.2735	12.5147	-10.2735	7.0326	-2.8650	0.9358
	a	1.0000	-3.0108	7.2670	-10.4407	12.5081	-10.1001	6.8006	-2.7256	0.8758

Tabulka 2: Koeficienty filtrů

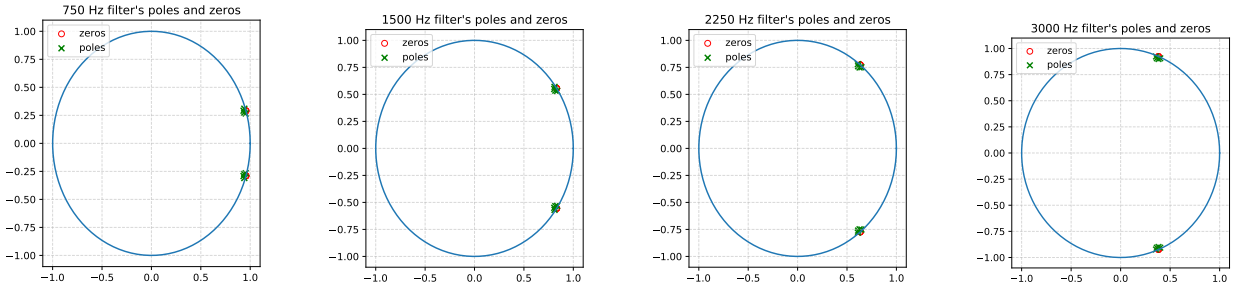


Obrázek 7: Impulsní odezvy filtrů

Každý z filtrů má šíři závěrného pásma (W_s) 30 Hz, šíři propustného pásma je (W_p) 50 Hz na každé ze stran závěrného pásma. Zvllnění (ripple) je 3 dB, potlačení (attenuation) je 40 dB.

```
for i in range(len(frequencies)):

    Wp = np.array([frequencies[i] - 65, frequencies[i] + 65]) * 2 / sRate
    Ws = np.array([frequencies[i] - 15, frequencies[i] + 15]) * 2 / sRate
    n, wNorm = sig.buttord( Wp, Ws, 3, 40 )
    b[i], a[i] = sig.butter( n, wNorm, btype = 'bandstop' )
```

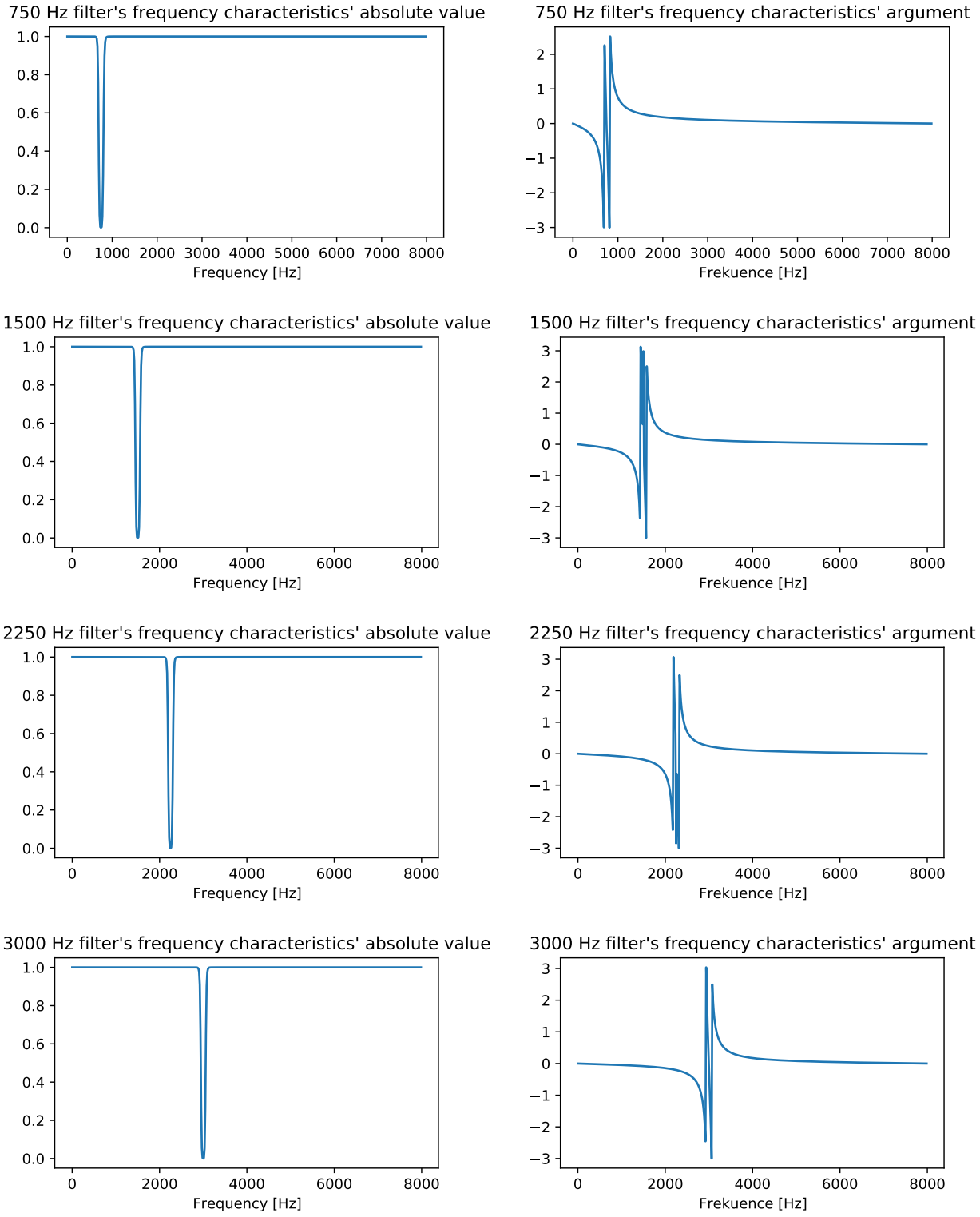
2.1.8 Nulové body a póly



Obrázek 8: Nulové body a póly filtrů

Z grafů je vidět, že všechny 4 filtry jsou stabilní (všechny nuly a póly leží přímo na jednotkové kružnici). To potvrzuje i kontrola převzata z příkladů Katky Zmolíkové.

2.1.9 Frekvenční charakteristika



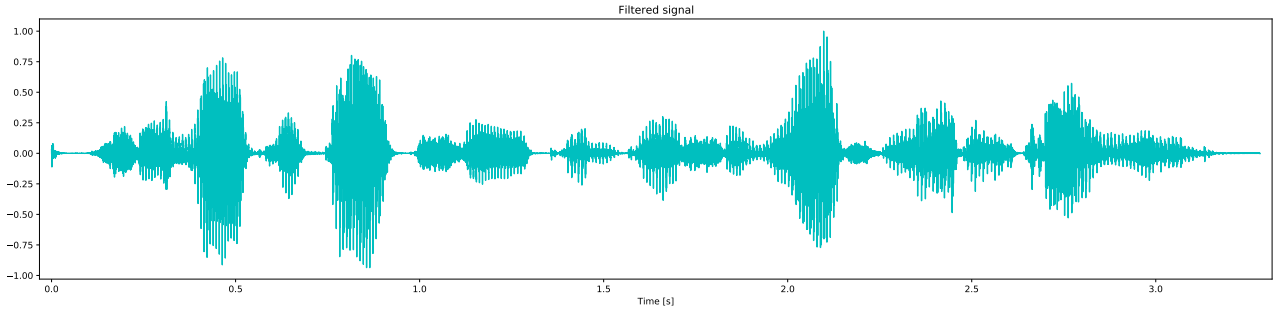
Obrázek 9: Nulové body a póly filtrů

Z grafů modulů frekvenční charakteristiky filtrů je zřejmé, že každý z nich odpovídá každé z rušivých frekvencí z původního signálu (750, 1500, 2250 a 3000 Hz)

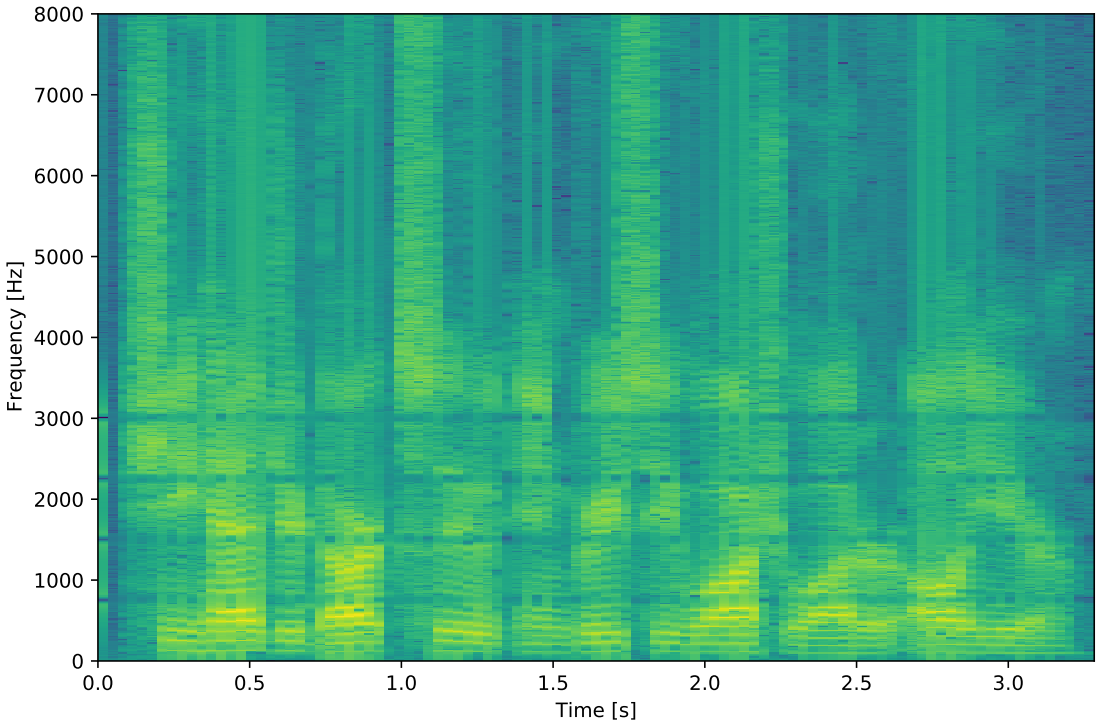
2.1.10 Filtrace

Filtrace signálu proběhla čtyřikrát (každým z filtrů) hned po vytvoření jejich koeficientů a vzhledem k zvuku výsledného signálu si myslím, že filtrování bylo úspěšné.

Během prvních několika milisekund je ale signál stále zašuměný. Důvodem je takzvaný "Edge effect", kvůli kterému se filtr aplikuje na prvních několik vzorků.



Obrázek 10: Filtrovaný signál



Obrázek 11: Filtrovaný signál

Reference

[1] Rozdělení signálu na rámce: https://superkogito.github.io/blog/SignalFraming.html#signal-framing

[2] Vykreslování signálu, frekvenční charakteristika signálu: https://nbviewer.org/github/zmolikova/ISS_project_study_phase/tree/master/

[3] SciPy dokumentace: https://docs.scipy.org/doc/

[4] NumPy dokumentace: https://numpy.org/doc/