VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Projekt

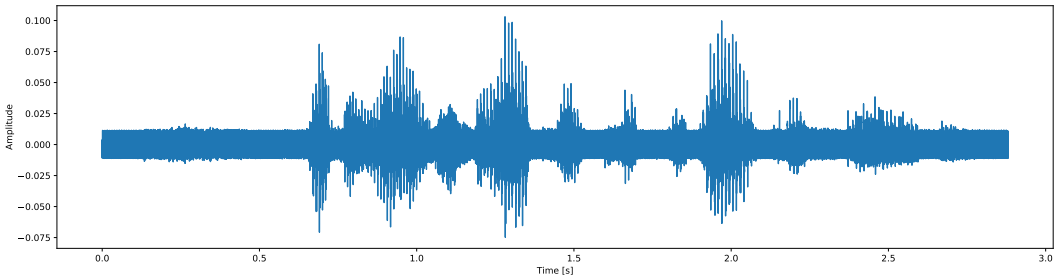
ISS – Signály a systémy

26. prosince 2021 Andrei Shchapaniak, xshcha00

Úvod

Máme na vstupu signál zarušený nějakými artefakty a je potřeba jej vyfiltrovat. V tomto projektu těmito artefakty budou 4 harmonicky vztažené cosinusovky. Potřebujeme signál analyzovat, najít, na kterých frekvencích cosinusovky jsou, navrhnout filtr pro čištění signálu a pak signál vyčistit.
Tento projekt byl řešen v programovacím jazyce `Python`. Veškeré výpočty jsou v odevzdaném souboru `solution.py`.

1 Základy



Tento signál byl načten pomocí funkce `soundfile.read()`[1]. K němu byly nalezeny následující údaje:

1. Délka ve vzorcích: **46080**
2. Délka ve sekundách: **2.88** [s]
3. Minimální hodnota: **-0.0748**
4. Maximální hodnota: **0.1031**

2 Předzpracování a rámce

Rozdělení signálu bylo provedeno pomocí následujícího příkazu:

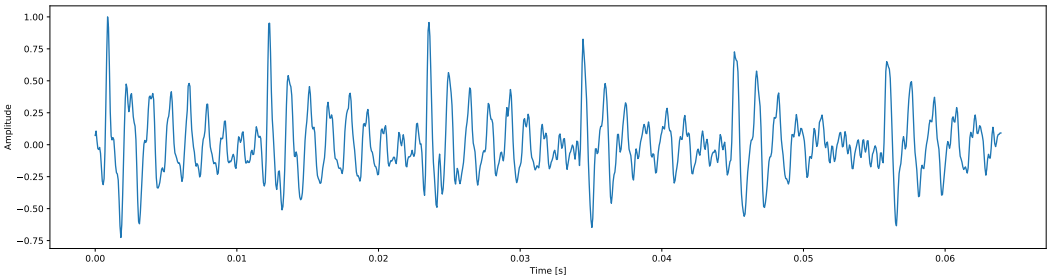
```
import numpy.lib.stride_tricks as stride
#normal_arr       - normalizovany signal
#length           - delka ramce 1024
#size_overlapping - delka prekryti 512
#pad_size         - pocet ramcu 90
frames_2D = stride.as_strided(normal_arr,
            (math.ceil(pad_size*2), length),
            (size_overlapping*normal_arr.itemsize, normal_arr.itemsize))

```

kde: `as_strided()`[2] - funkce pro rozdělení pole na překrývající rámce.

[1] https://pysoundfile.readthedocs.io/en/latest/
[2] https://numpy.org/doc/stable/reference/generated/numpy.lib.stride_tricks.as_strided.html

Následující obrázek je 40. rámec ustředněného a normlaizováného signálu. Vybral jsem ho brute forcem, protože vypadal pěkneji, než ostatní a přišel mi dost vhodný pro další ukázku.



3 DFT

Dalším úkolem bylo naimplementovat diskrétní Fourierovou Transformaci pro rámec z 2. úlohy a porovnat výsledek s funkcí z Pythonu `numpy.fft.fft()`[3]. Podle dálšího vzorce

$$\begin{bmatrix} e^{-j\cdot\omega\cdot\frac{0\cdot1}{N}} & e^{-j\cdot\omega\cdot\frac{0\cdot2}{N}} & \dots & e^{-j\cdot\omega\cdot\frac{0\cdot(N-1)}{N}} \\ e^{-j\cdot\omega\cdot\frac{1\cdot1}{N}} & e^{-j\cdot\omega\cdot\frac{1\cdot2}{N}} & \dots & e^{-j\cdot\omega\cdot\frac{1\cdot(N-1)}{N}} \\ \vdots & \vdots & \ddots & \vdots \\ e^{-j\cdot\omega\cdot\frac{(N-1)\cdot1}{N}} & e^{-j\cdot\omega\cdot\frac{(N-1)\cdot2}{N}} & \dots & e^{-j\cdot\omega\cdot\frac{(N-1)\cdot(N-1)}{N}} \end{bmatrix} \cdot \vec{\mathbf{x}} = \vec{\mathbf{X}}$$

jsem naimplementoval v Pythonu vlastní DFT:

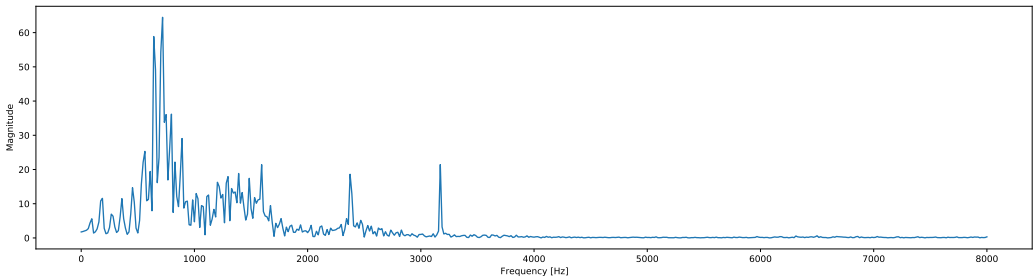
```
#x je vstupni signal
#X je vystupni DFT
N = len(x)
n = np.arange(N)
k = n.reshape((N, 1))
e = np.exp(-2j * np.pi * k * n / N)
X = np.dot(e, x)

```

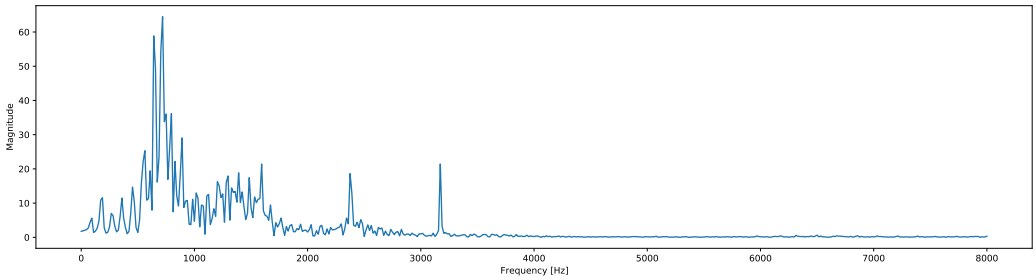
Pak jsem zobrazil modul vlastní DFT a modul FFT pro frekvence od 0 do 8000 [Hz]. Tak bych mohl porovnat graficky, zda jdu správným směrem.

[3] https://numpy.org/doc/stable/reference/generated/numpy.fft.fft.html

Z obrazků 1 a 2 je pěkné vidět, že moje vlastní DFT je shodná s FFT z Pythonu. Toto potvrzuje i funkce `numpy.allclose()`[4].



Obrázek 1: Vlastní DFT

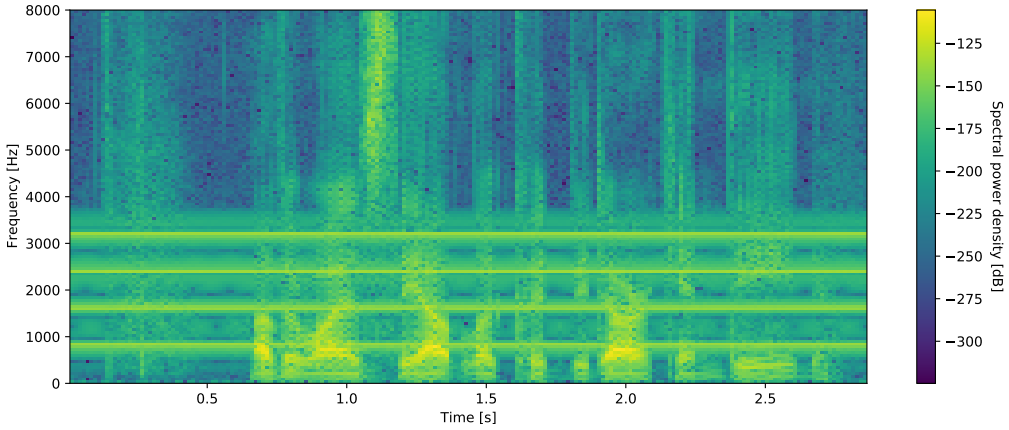


Obrázek 2: FFT z knihovny NumPy

[4]https://numpy.org/doc/stable/reference/generated/numpy.allclose.html

4 Spektrogram

Až jsem měl správnou DFT, posunul jsem se ke kreslení spektrogramu. Pro tyto účely bylo možné využít funkci `matplotlib.pyplot.pcolormesh()`[5] s malými úpravami pro lepší zobrazení.



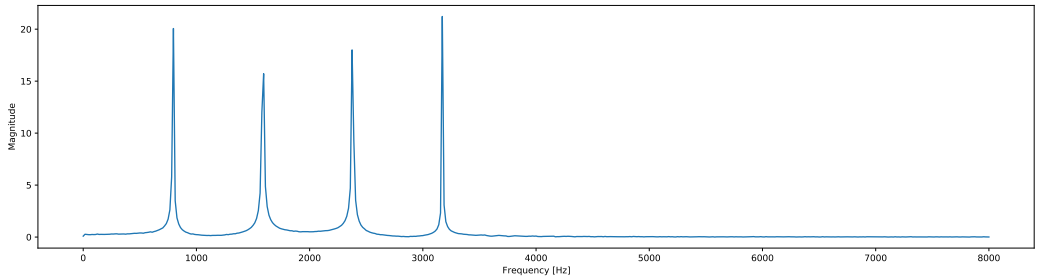
Obrázek 3: Spektrogram celého signálu

Pro tento způsob bylo nutné udělat následující kroky:
1. Aplikovat DFT na každý překrývající rámec.
2. Upravit pomocí vzorce: $20 \cdot \log_{10} |X[k]|$.
Ale byl i lepší způsob použít pěknou funkci `scipy.signal.spectrogram()`[6]. Zvolil jsem právě 2. způsob. Výsledný spektrogram lze vidět na obrázku 3.

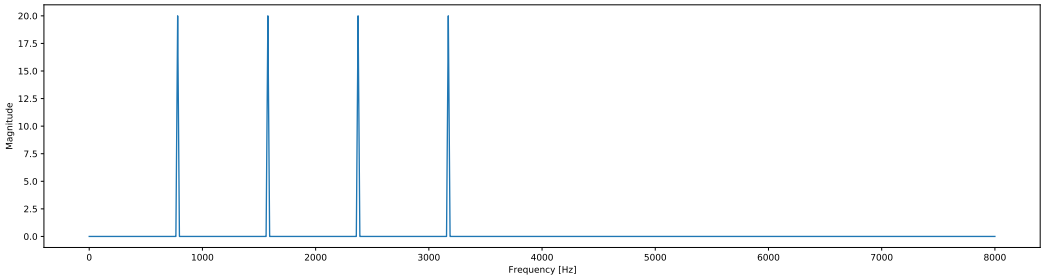
5 Určení rušivých frekvencí

Na obrázku 3 jsou viditelné 4 rušivé komponenty, frekvence kterých jsem musel určit. Vybral jsem následující metodu:
1. Ze spektogramu je vidět, že 1. rušivá komponenta se nachází na frekvenci od 0 do 1000 Hz.
2. Vzál jsem 1. rámec, udělal jsem nad ním DFT a našel jsem pozici s největsím modulem mezi prvními 128 položkami.
3. Poslední, co jsem udělal, tak vypočítal frekvenci na základě pozice v poli.
Po těchto manipulacích jsem dostal frekvenci 1. cosinusovky.

[5] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.pcolormesh.html
[6] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.spectrogram.html



Obrázek 4: Ručně naleznuté frekvence cosinusovek



Obrázek 5: Násobky 1. frekvence cosinusovky

Z obrazků 4 a 5 je vidět, že jsem určil přiblízně správné frekvence cosinusovek. Chyba je kvůli tomu, že běžná FFT v našem případě má přesnost 16000/1024 = 15.6 [Hz]. Takže můžu počítat s tím, ze metoda pro určení rušivé frekvence byla zvolena správně. Ale pro přesnou filtraci by bylo vhodné najít přesnou frekvenci. Díky tomu, že v projektu jsou bonusové úkoly, našel jsem tam způsob, který jsem zkusil naimplementovat:

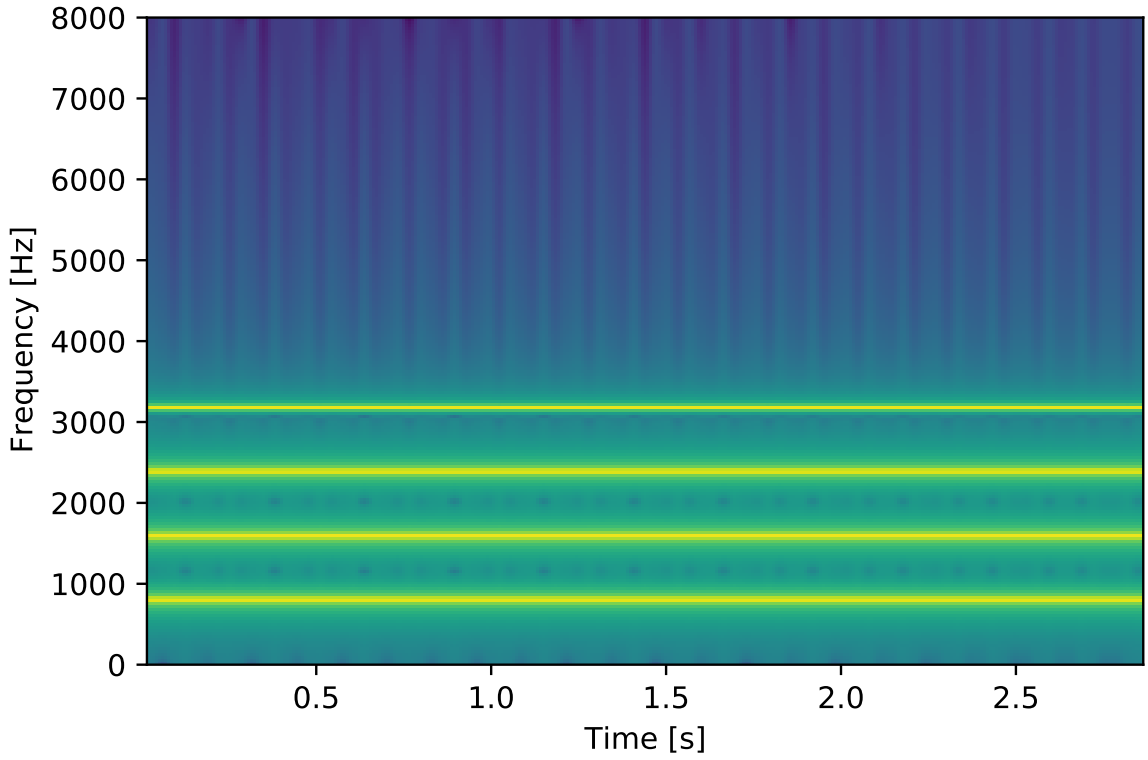
```
# frq = 16000/1024.0
# signal = original_signal[0:16000]
# from_fs = (index_of_potential_freq - 1) * frq
# to_fs = (index_of_potential_freq + 1) * frq
def find_freq_1_cos(signal, from_fs, to_fs):
    N = 16000
    freq_step = N/1024.0
    n = np.arange(0, N)
    k = np.linspace(from_fs, to_fs, to_fs - from_fs)
    k = np.reshape(k, (len(k), 1))
    e = np.exp(-2j * np.pi * k * n / N)
    X = np.dot(e, signal)
    X_abs = np.abs(X)
    return from_fs + np.argmax(np.abs(X_abs))

```

Stručně popíšu, jaká je tady myšlenka založena. Určil jsem přiblíznou frekvenci 1. cosinusovky. Pak jsem vygeneroval matici s komplexními exponenciálami s krokem 1 Hz okolo hrubo určené. Určil jsem podobnost s prvními 16000 vzorky původního signálu a dostal jsem, že největší podobnost je na frekvenci **793** Hz. To je dost pravděpodobné.

6 Generování signálu

V tomto úkolu jsem musel vygenerovat signál z cosinusovek na frekvencích, které jsem zjistil v předchozím úkolu, a zobrazit jeho spektrogram.



Výše je spektrogram signálu, který jsem dostal pomocí součtu 4 cosinusovek. Je si lehce všimnout, že tento signál obsahuje pouze cosinusovky, nic jiného. O tom nám říká zelená bárva.

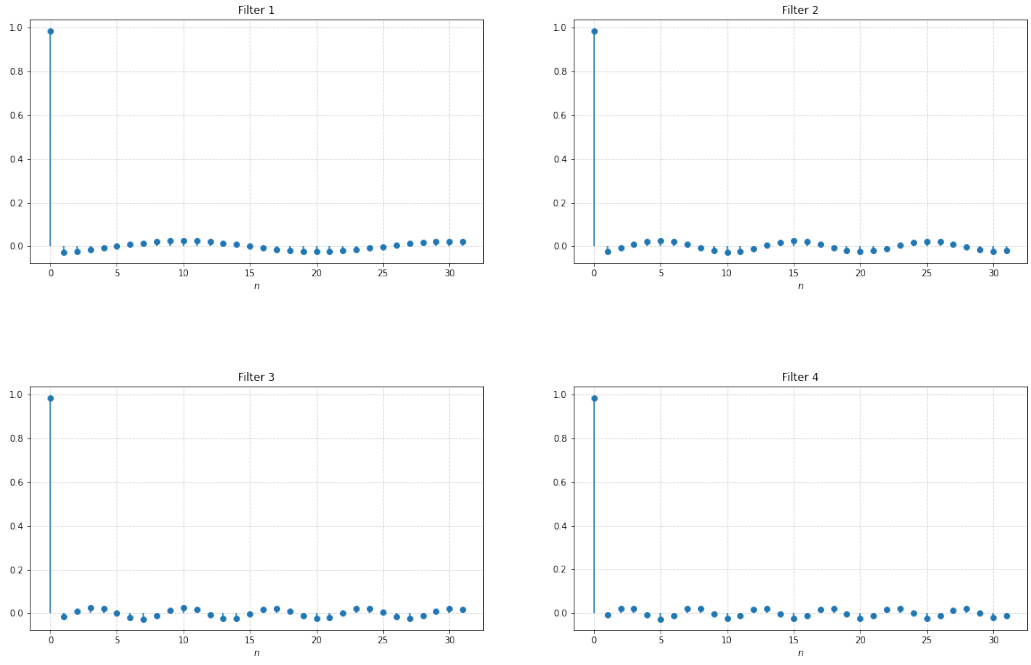
7 Čisticí filtr

Určil jsem frekvence rušivých komponent. Teď nastal čas filtrování. Jsou různé způsoby filtrace signálů. Zvolil jsem implementovat pro každou komponentu vlastní filtr - pásmová zádrž (band-stop filter). Python obsahuje pěkné pomocné funkce `scipy.signal.buttord`[7], `scipy.signal.butter`[8] , které na základě jistých dat vygenerují koeficienty **a**, **b** pro filtr.

Níže v tabulce je uveden stručný popis filtrů.

	Characteristic of filters			
	Filter 1	**Filter 2**	**Filter 3**	**Filter 4**
Frequency [Hz]	793	1586	2379	3172
Passband [Hz]	778 - 808	1571 - 1601	2364 - 2394	3157 - 3187
Stopband [Hz]	768 - 818	1561 - 1611	2354 - 2406	3147 - 3197
Ripple [dB]	3	3	3	3
Stop-band attenuation [dB]	-40	-40	-40	-40
Stability	True	True	True	True

Dolů je zobrazena impulsní odezva každého filtru zvlášť a je uvedena tabulka s koeficienty filtrů.

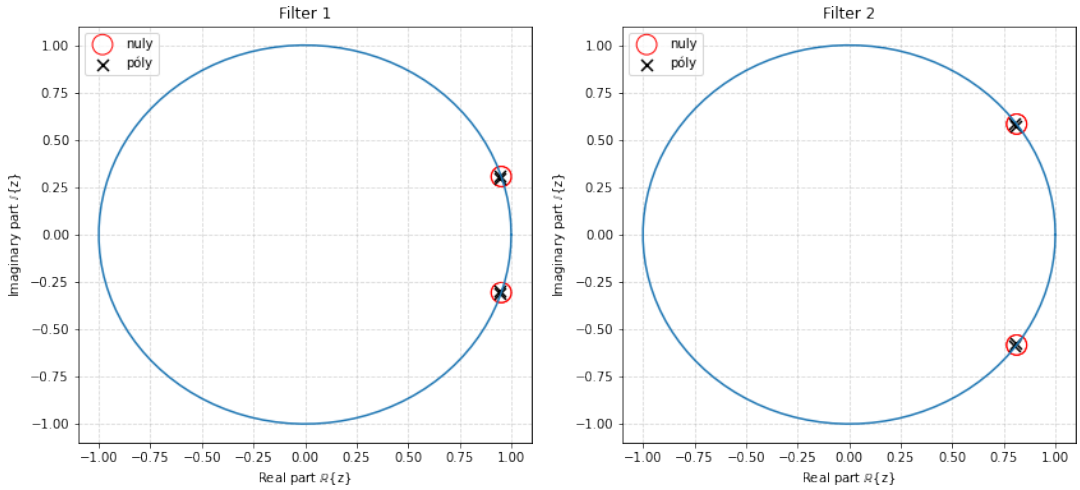


[7] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.buttord.html
[8] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html

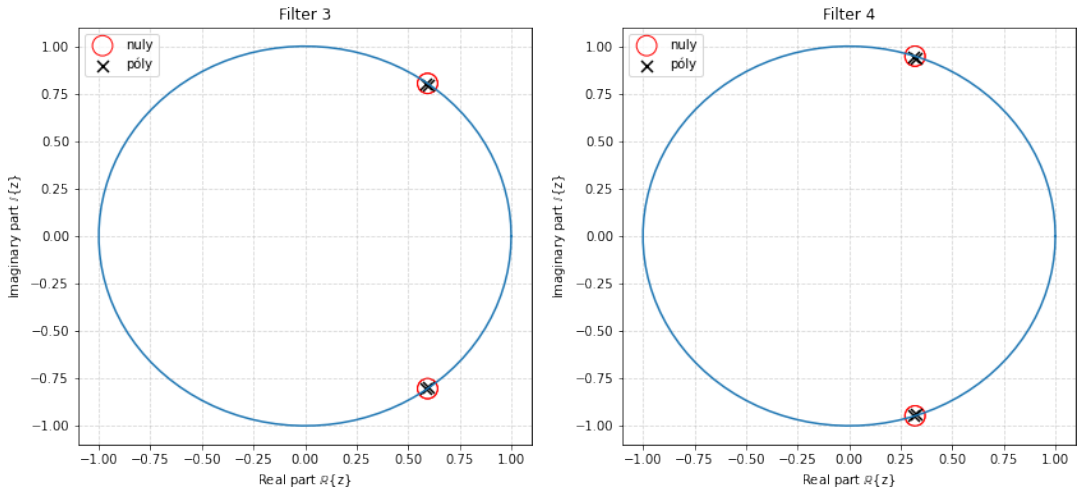
	Coefficients of filters							
	Filter 1		Filter 2		Filter 3		Filter 4	
N	**a**	**b**	**a**	**b**	**a**	**b**	**a**	**b**
0	1.0	0.986	1.0	0.986	1.0	0.986	1.0	0.986
1	-3.781	-3.755	-3.227	-3.204	-2.361	-2.345	-1.268	-1.260
2	5.548	5.548	4.575	4.575	3.366	3.366	2.375	2.375
3	-3.730	-3.755	-3.182	-3.204	-2.329	-2.345	-1.251	-1.260
4	0.973	0.986	0.973	0.986	0.973	0.986	0.973	0.986

8 Nulové body a poly

V tomto úkolu bylo nutné zobrazit nuly a poly navržených filtrů. Na pomoc zase přijde rozsahlá knihovná Pythonu. Pro tyto účely jsem vybral funkci `sp.signal.tf2zpk()`[9], která z koeficientů **a**, **b** dostane nuly a poly. Nuly a poly na jednotkové kružnici můžete vidět dolů.



[9] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html



Přesnějsi informace o nulách a polech jsou uvedené v tabulce dolů.

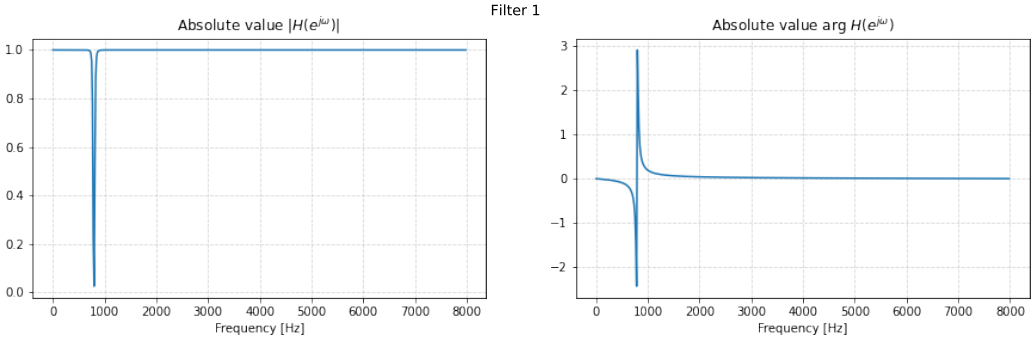
	Zeros and pols of filters			
	Filter 1	**Filter 2**	**Filter 3**	**Filter 4**
1. pol	0.943 + 0.311j	0.802 + 0.586j	0.585 + 0.803j	0.310 + 0.943j
2. pol	0.943 - 0.311j	0.802 - 0.586j	0.585 - 0.803j	0.310 - 0.943j
3. pol	0.947 + 0.298j	0.811 + 0.574j	0.596 + 0.795j	0.323 + 0.939j
4. pol	0.947 - 0.298j	0.811 - 0.574j	0.596 - 0.795j	0.323 - 0.939j
1. zero	0.952 + 0.306j	0.812 + 0.583j	0.594 + 0.804j	0.319 + 0.947j
2. zero	0.952 - 0.306j	0.812 - 0.583j	0.594 - 0.804j	0.319 - 0.947j
3. zero	0.952 + 0.306j	0.812 + 0.583j	0.594 + 0.804j	0.319 + 0.947j
4. zero	0.952 - 0.306j	0.812 - 0.583j	0.594 - 0.804j	0.319 - 0.947j

9 Frekvenční charakteristika

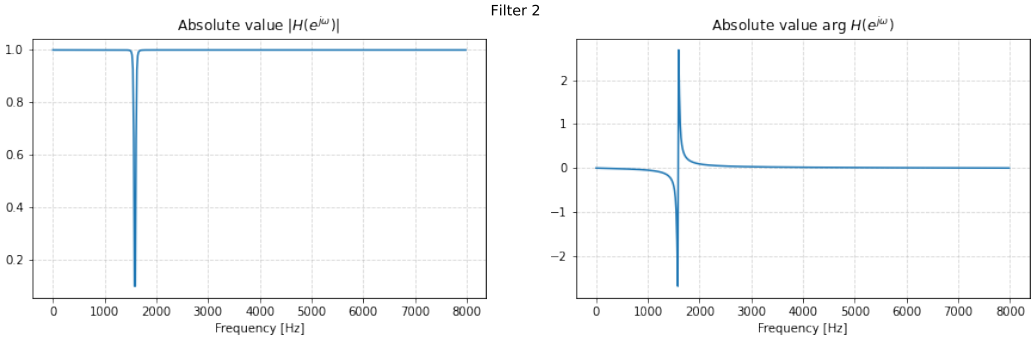
Před filtrací je vhodné udělat frekvenční charakteristiku a ověřit, že filtr potlačuje rušivý signál na správných frekvencích. Dostal jsem frekvenční charakteristiku pomocí funkce `scipy.signal.freqz()`[10], která na vstup bere koeficienty filtru.

[10] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.freqz.html

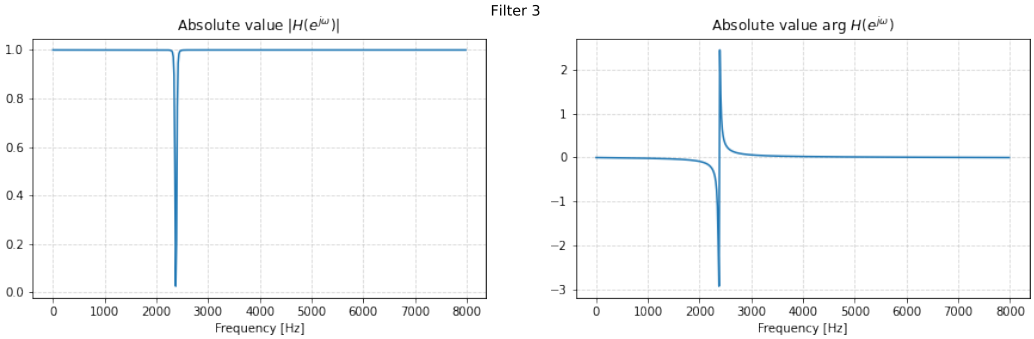
Filter 1

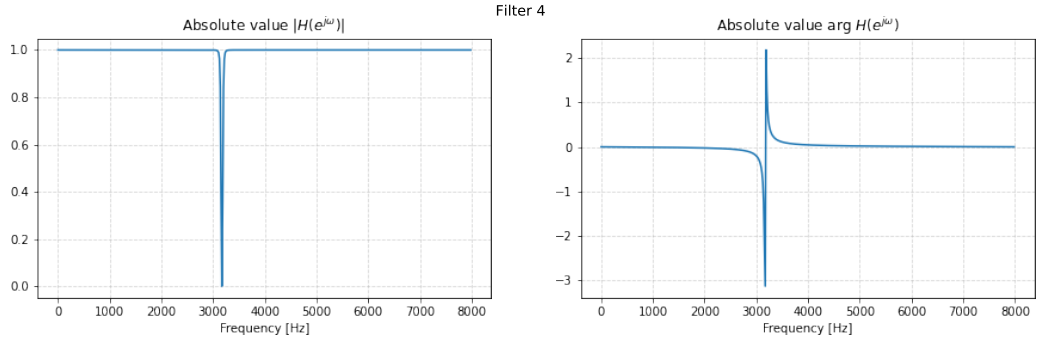


Filter 2



Filter 3

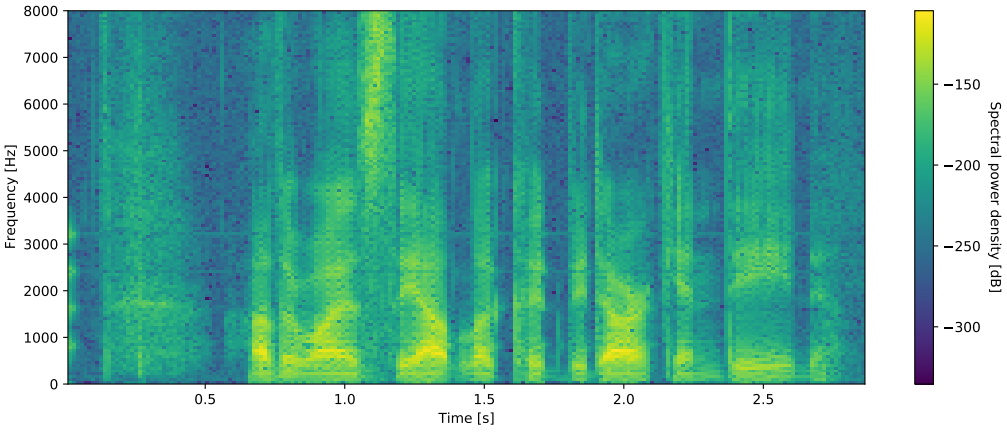




Když se podíváme na osu x s frekvencemi, tak se přesvědčíme, že navržené filtry jsou korektní. Můžeme se posunout k filtraci signálu.

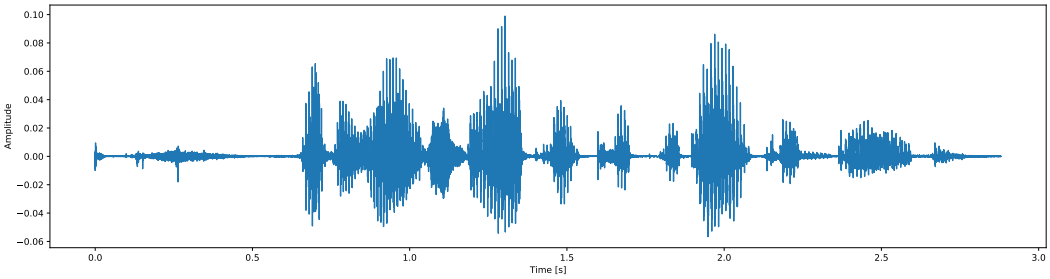
10 Filtrace

Posledním úkolem v tomto projektu je provést filtraci signálu. Graficky ověřit správnou filtraci je možne pomocí obrázků 6 a 7, kde je vidět, že cosinusovky zmizly.
Filtrace byla provedena pomocí funkce `scipy.signal.lfilter()`[11].



Obrázek 6: PSD finálního odfiltrovaného signálu

[11] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.lfilter.html



Obrázek 7: Finální odfiltrovaný signál

11 Závěr

Signál `xshcha00.wav` byl úspěšně odfiltrován pomocí 4 band-stop filtrů a cosinusovky na rušivých frekvencích byly potlačené.