Protokol k projektu z predmetu ISS

Andrej Hýroš - xhyros00

Január 2022

1 Použité moduly/nástroje

Pojekt bol riešený v prostredí Jupyter.

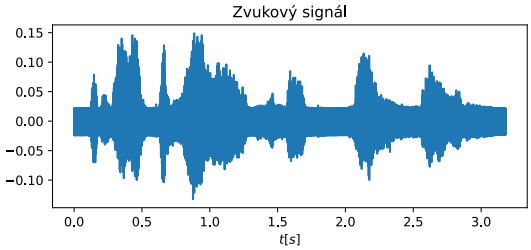
- `SoundFile` Pythonovský modul pre prácu s audio súborom
- `NumPy` Práca s načítanými dátami
- `SciPy` Práca s načítanými dátami
- `matplotlib` Tvorba grafického výstupu

2 Odpovede

2.1 Základy

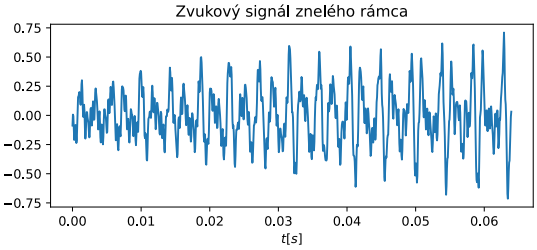
Vstupný súbor bol načítaný funkciou `read()` z modulu `soundfile`.
`s, fs = sf.read('xhyros00.wav')`
Počet vzorkov signálu bol získaný ako dĺžka načítaných dát funkciou `len()`. Pre získanie dĺžky v sekundách je potrebné predeliť počet vzorkov vzorkovacou frekveniou `fs`.
`time = signal_size / fs`
Pre získanie maximálnej a minimálnej hodnoty signálu boly použité funkcie z knižnice `numpy`:
`maximum = np.max(s)`
`minimum = np.min(s)`

Odpovede:
Počet vzorkov signálu je 50893 na vzorkovacej frekvencii 16000 Hz
Dĺžka signálu je 3.1808125 sekúnd
maximum: 0.15
minimum: -0.13



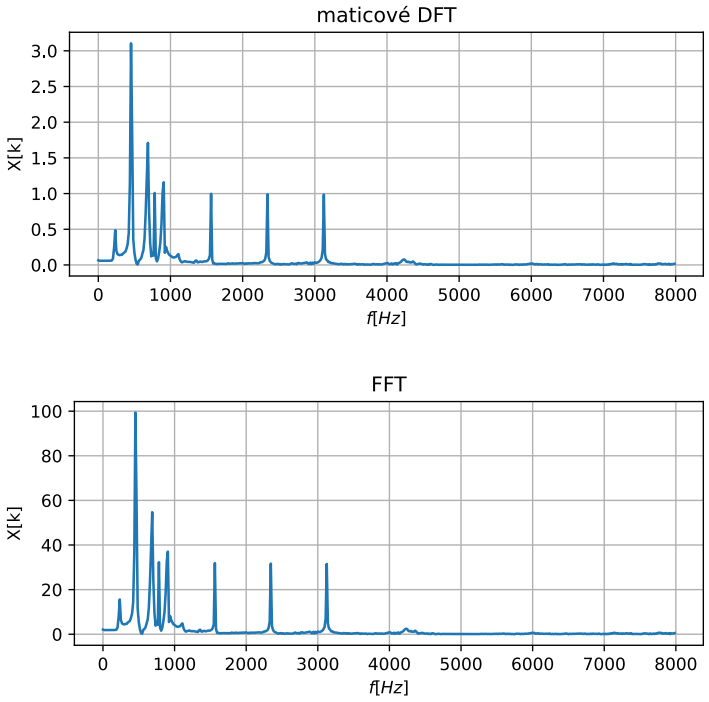
2.2 Predspracovanie a rámce

Signál bol ustrednený a normalizovaný:
`s -= np.mean(s)`
`s /= np.max(np.abs(s))`
Pre rozdelenie funkcie do rámcov som si napísal vlastnú funkciu `strided_frames()`. Znelý rámec som identifikoval na rámci s indexom 25. Je zobrazený na grafe nižšie.



2.3 DFT

Implementoval som vlastnú DFT vo funkciách `create_dft_matrix()` a `matrix_dft()`. Postupoval som podla vzoru DFT matrix, ktorý je napíklad vysvetlený tu: `https://en.wikipedia.org/wiki/DFT_matrix`
Jej výsledky sa ale po teste `np.allclose()` odlišovali od vstavanej `np.fft.fft()`. Zdalo sa, že hodnoty mojej implementácie sú v pomere rovnaké, ale približne 33 krát nižšie než hodnoty vstavanej fft implementácie. Nedokázal som prísť na chybu, ale pokračoval som s výsledkom mojej implementácie. Na grafoch nižšie je možné vidieť rozdiel medzi mojou a numpy implementáciou.



2.4 Spektrogram

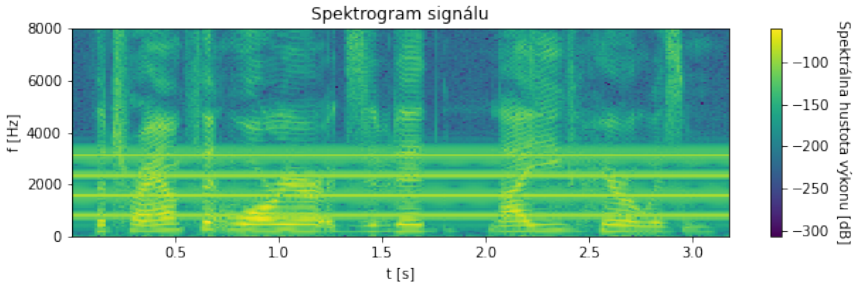
Pre celý signál bol vyrátaný spektogram s využítim funkcie `spectrogram()` z knižnice `scipy`:

```
f, t, sgr = spectrogram(s, fs)
```

Následne podla zadania upravené:

```
sgr_log = 10 * np.log10(np.power(np.abs(sgr), 2))
```

a zobrazené:



Na spektrograme si môžeme všimnúť rušivé frekvencie v podobe štyroch rovných horizontálnych čiar.

2.5 Určenie rušivých frekvencií

Hladanie rušivých frekvencií prebiehalo 'od oka' odčítaním z jedného spektra. Pri troch najvyšších frekvenciách to bolo jednoduché. Odhadol som index vľavo a vpravo od danej frekvencie, a použitím funkcie `np.argmax()` našiel index maxima medzi nimi takto:
`f2_i = np.argmax(X[64:128]) + 64`
Frekvencia sa dá následne z indexu dostať vzťahom:

```
f = index * 15.625
```

Pri hladaní najnižšej rušívej frekvencie som si nechal funkciou `scipy.signal.argrelextrema()` vrátiť všetky indexy lokálnych maxím daného spektra na intervale 0 - 1000 Hz. Z grafu sa potom dá odčítať že táto rušivá frekvencia má v danom intervale štvrtú najvyššiu hodnotu. Zistil som teda index štvrtého najnižšieho lokálneho maxima a opäť prerátal na frekvenciu.
Rušivé funkcie som teda určil ako:

$$f_1 = 781.25Hz$$
$$f_2 = 1546.875Hz$$
$$f_3 = 2343.75Hz$$
$$f_4 = 3109.375Hz$$

2.6 Generovanie signálu

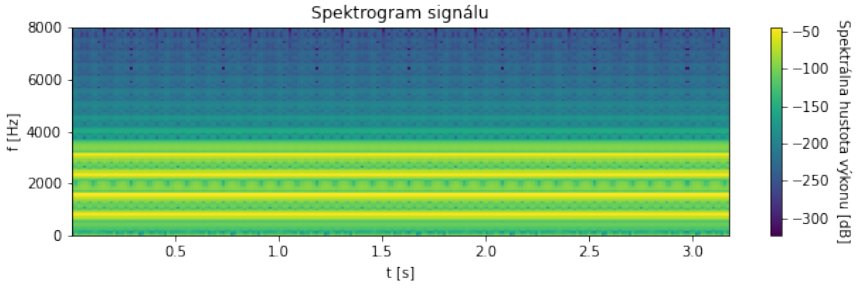
Nájdené frekvencie som si uložil do zoznamu:

```
frequencies_list = np.array([f1, f2, f3, f4])
```

a následne generoval signál zmiešaný z cosinusoviek daných frekvencií:

```
for f in frequencies_list:
    signal += np.cos(2*np.pi*f*time)
```

Spektrogram bol vytvorený rovnakým spôsobom ako v sekcii `4.4 Spektrogram`:



2.7 Čistiaci filter

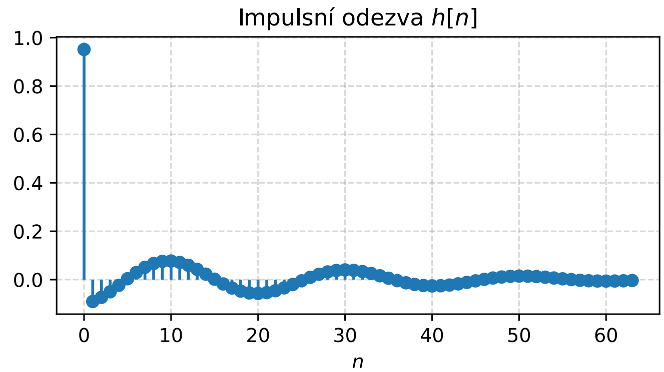
Postupoval som alternatívou č. 3, a to využitím funkcií `scipy.signal.buttord()` a `scipy.signal.butter()`. Potrebné parametre som nastavil podla odporúčania zo zadania. Spolu s koeficientami boli rovno vypočítané nuly a póly filtrov

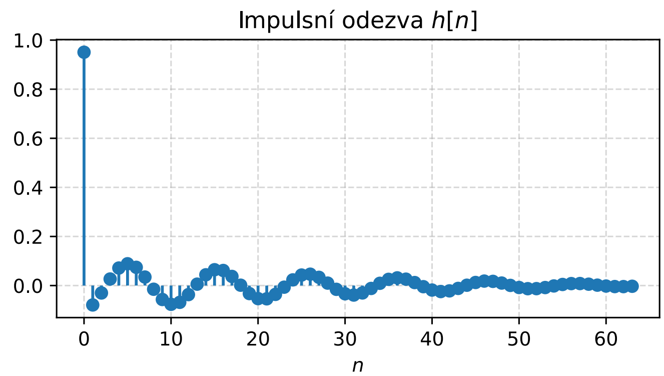
2.7.1 Koeficienty

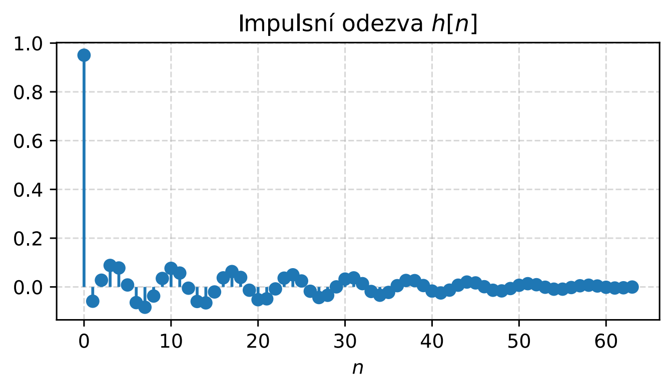
'a' koeficienty filtrov:
1. filter: (1, -7.531, 25.172, -48.738, 59.769, -47.537, 23.947, -6.988, 0.905)
2. filter: (1, -6.486, 19.674, -36.022, 43.375, -35.121, 18.703, -6.011, 0.904)
3. filter: (1, -4.783, 12.476, -20.822, 24.466, -20.298, 11.856, -4.431, 0.903)
4. filter: (1, -2.706, 6.6450, -9.1520, 11.263, -8.9210, 6.3140, -2.507, 0.903)

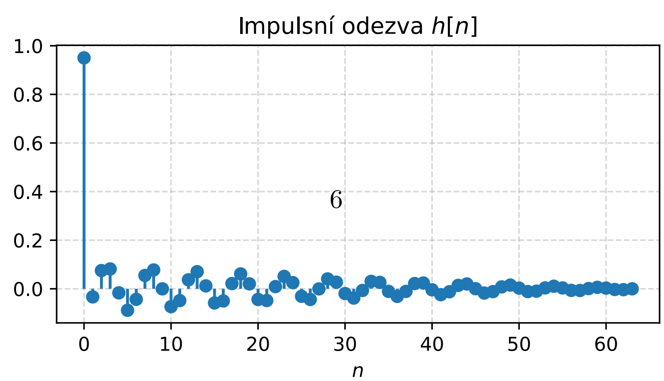
'b' koeficienty filtrov:
1. filter: (0.951, -7.255, 24.555, -48.142, 59.780, -48.142, 24.555, -7.255, 0.951)
2. filter: (0.951, -6.245, 19.185, -35.576, 43.384, -35.576, 19.185, -6.245, 0.951)
3. filter: (0.950, -4.604, 12.164, -20.563, 24.472, -20.563, 12.164, -4.604, 0.950)
4. filter: (0.950, -2.605, 6.4780, -9.0380, 11.266, -9.0380, 6.4780, -2.605, 0.950)

2.7.2 Impulzná odozva filtrov



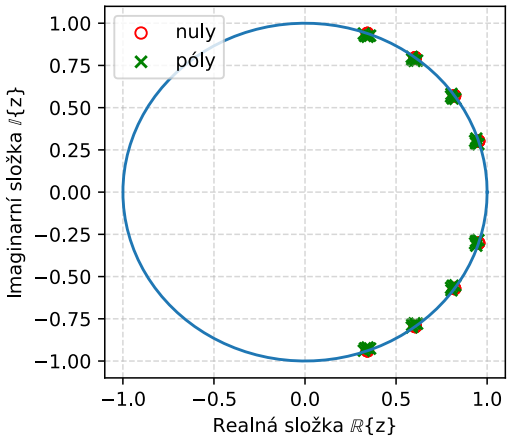






2.8 Nuly a póly

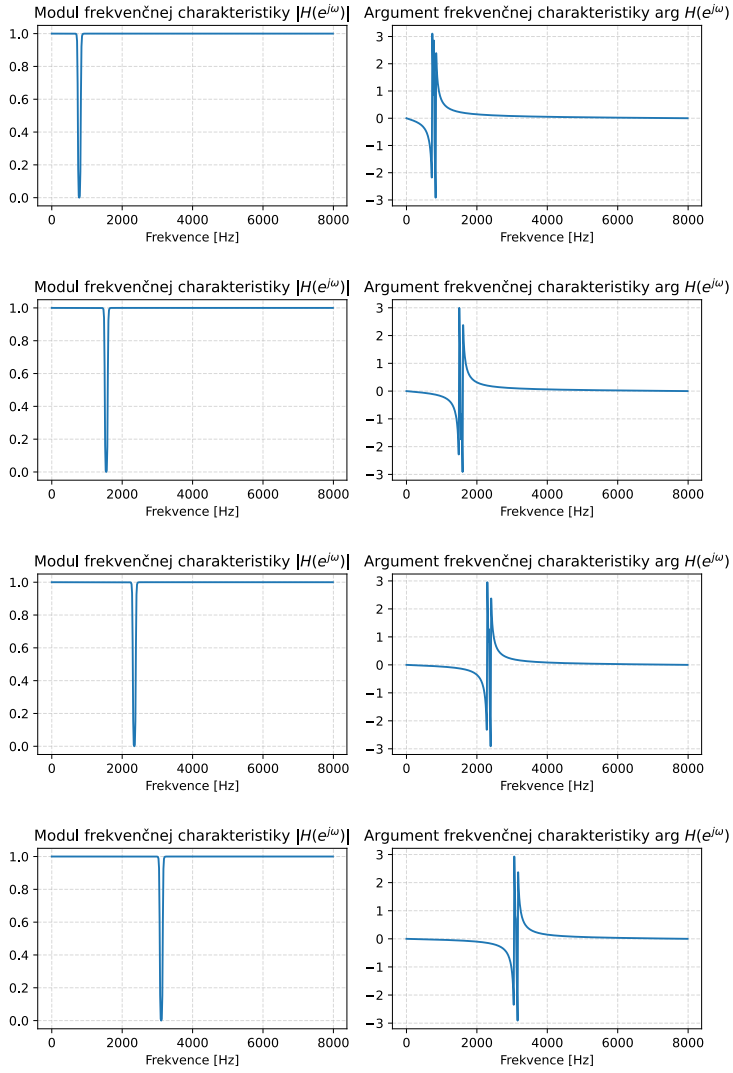
Vyrátané boli pri tvorbe filtrov funkciami `buttord()` a `butter()`.



2.9 Frekvenčná charakteristika

Bola vyrátaná s využitím funkcie `freqz()` z knižnice `scipy.signal` (`https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.freqz.html`) a koeficientov navrhnutých filtrov takto:

```
w, H = freqz(b, a)
```

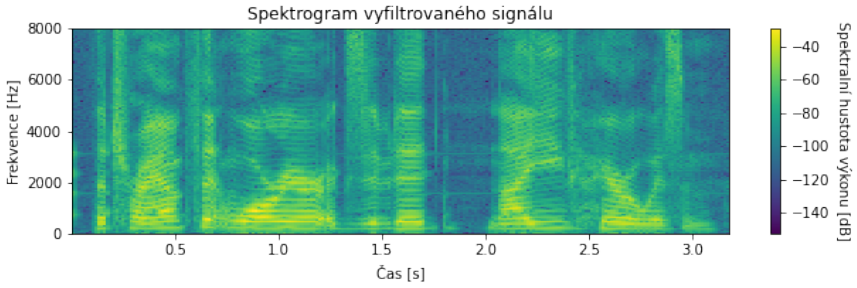

Modul frekvenčnej charakteristiky $|H(e^{j\omega})|$
Argument frekvenčnej charakteristiky arg $H(e^{j\omega})$
Frekvence [Hz]
Frekvence [Hz]
Modul frekvenčnej charakteristiky $|H(e^{j\omega})|$
Argument frekvenčnej charakteristiky arg $H(e^{j\omega})$
Frekvence [Hz]
Frekvence [Hz]
Modul frekvenčnej charakteristiky $|H(e^{j\omega})|$
Argument frekvenčnej charakteristiky arg $H(e^{j\omega})$
Frekvence [Hz]
Frekvence [Hz]
Modul frekvenčnej charakteristiky $|H(e^{j\omega})|$
Argument frekvenčnej charakteristiky arg $H(e^{j\omega})$
Frekvence [Hz]
Frekvence [Hz]

2.10 Filtrácia

Pre filtráciu navrhnutými filtrami bola použitá funkcia `lfilter()` zo `scipy.signal`:

```
sf = s
sf = lfilter(B[0], A[0], sf)
sf = lfilter(B[1], A[1], sf)
sf = lfilter(B[2], A[2], sf)
sf = lfilter(B[3], A[3], sf)
```

kde `s` je pôvodný signál, a `A` a `B` sú polia koeficientov filtrov.
Pre kontrolu bol opäť vygenerovaný spektrogram signálu:



Ako je vidno zo spektrogramu (a taktiež z posluchu), rušivé frekvencie boli z pôvodného signálu úspešne odstránené, čo značí správny návrh filtrov.