ISS Projekt

Leden 2022

Augustin Machyňák
xmachy02

1 Úkoly

Pro řešení projektu byl použit programovací jazyk Python a knihovny *numpy*, *scipy* a *matplotlib*. Kromě materiálů zmíněných v zadání bylo pro realizaci hojně využíváno dokumentace již zmíněných knihoven, jelikož se autor s těmito knihovny setkal poprvé v životě.

```
import numpy as np
from scipy.io import wavfile
from scipy import signal
import matplotlib.pyplot as plt
```

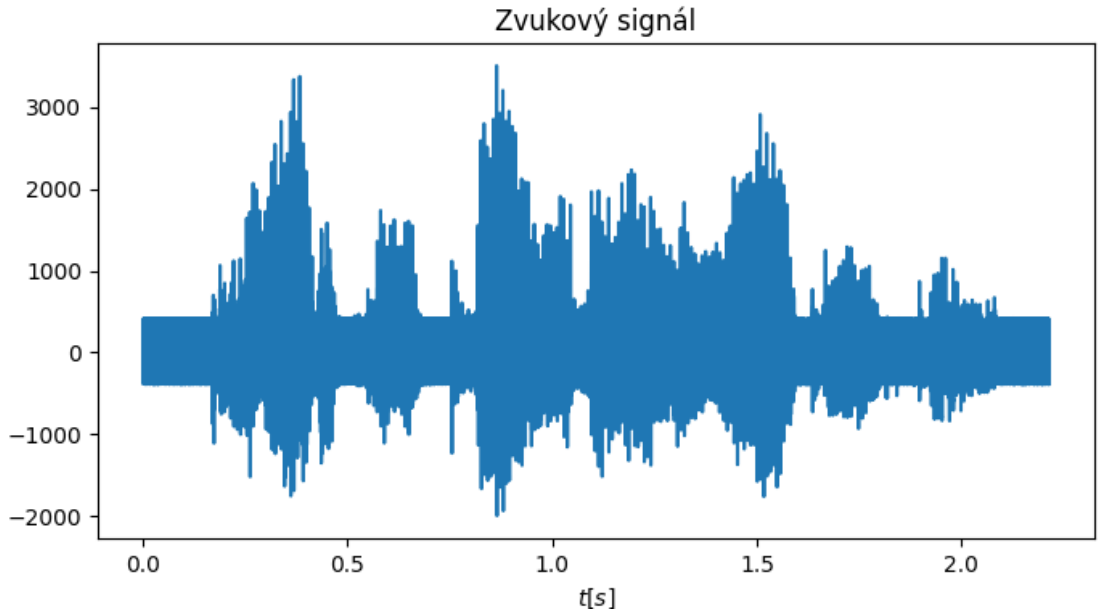
Kód 1: Použité knihovny

1.1 Základy

```
def read_file(file_name):
    fs, data = wavfile.read(file_name)
    return fs, data
```

Kód 2: Načtení vstupního signálu

Délka signálu ve vzorcích (*data.size*): 35431
Délka signálu v sekundách (*data.size/fs*): 2.214 [s]
Minimální hodnota (*data.min()*): -1996
Maximální hodnota (*data.max()*): 3506



Obrázek 1: Zobrazení vstupního signálu

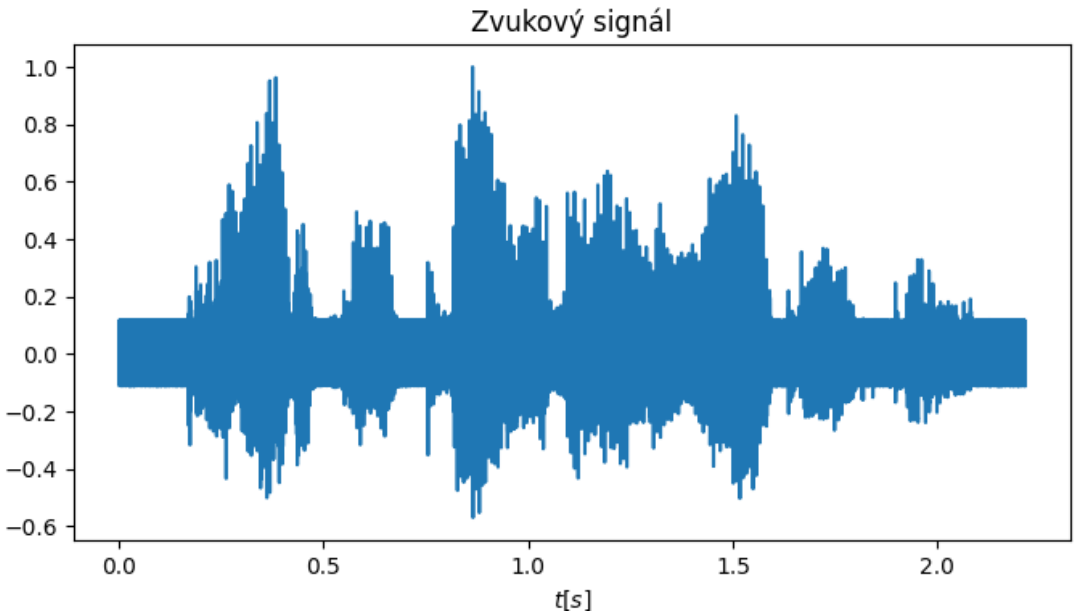
1.2 Předzpracování a rámce

```
def normalize(data):
    new_data = data.copy()
    m = np.mean(new_data, axis=0)
    new_data -= int(m)
    abs_max = np.maximum(np.abs(data.min()), data.max())
    new_data = new_data / abs_max
    return new_data
```

Kód 3: Ustřednění signálu a jeho normalizování do dynamického rozsahu

```
def split_512(data):
    new_data = []
    i = 0
    for i in range(0, data.size - 1024, 512):
        new_data.append(data[i:i + 1024])
    i += 512
    if i < data.size:
        new_data.append(data[i:data.size])
    return new_data
```

Kód 4: Rozdělení na rámce o velikosti 1024 vzorků s překrytím 512 vzorků



Obrázek 2: Ustředněný a normalizovaný vstupní signál

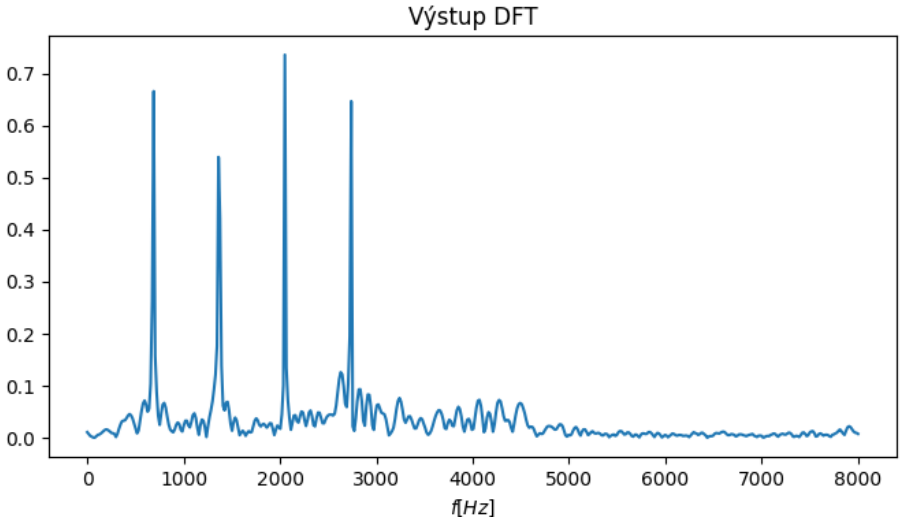
1.3 DFT

Pro implementaci DFT byla využita DFT Matice. DFT je realizována jako násobení této matice s vektorem signálu.

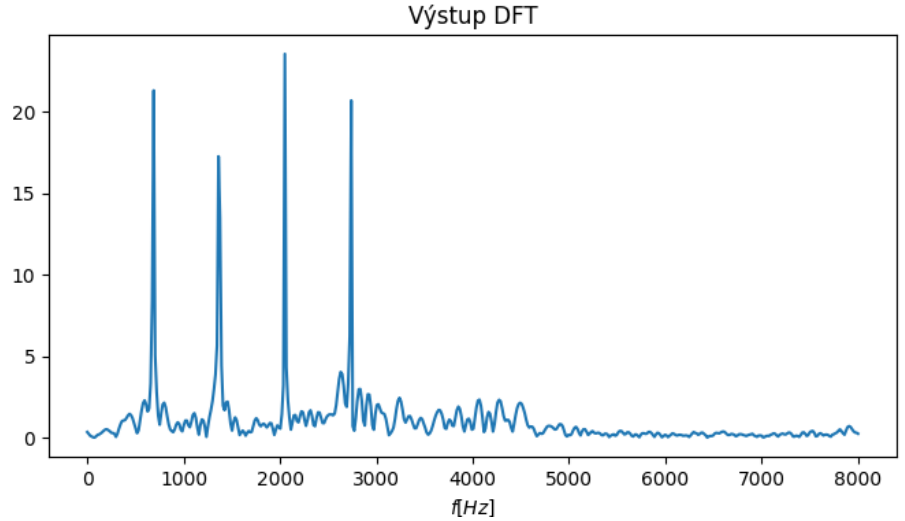
```
def my_dft(data):
    N = data.size
    # dft matrix
    a, b = np.meshgrid(np.arange(0, N), np.arange(0, N))

    # e^ -2*PI*J/N
    w = np.exp(-2 * np.pi * 1J / N)
    matrix = np.power(w, a * b) / np.sqrt(N)

    result = data.dot(matrix)
    return result
```

Kód 5: Implementace DFT

Bohužel však výsledné hodnoty neodpovídají hodnotám z numpy knihovní implementace FFT. Vizuálně výsledek vypadá stejně, ale hodnoty jsou několikanásobně menší (viz následující obrázky). Příčina se nepodařila odhalit.

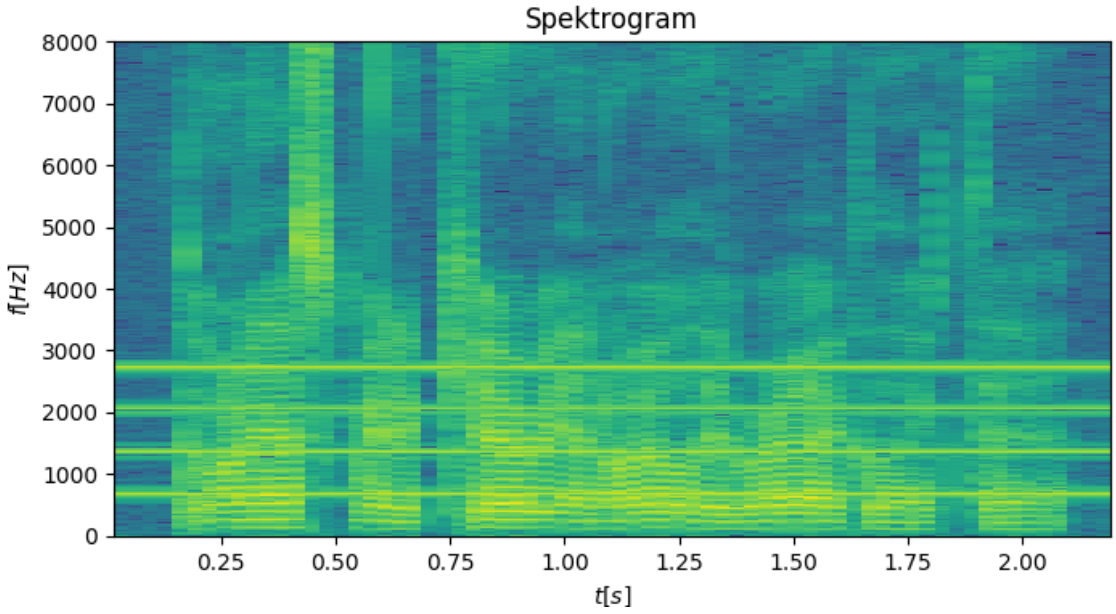


Obrázek 3: Výsledek vlastní implementace DFT



Obrázek 4: Výsledek knihovní FFT

1.4 Spektrogram



Obrázek 5: Spektrogram vstupního signálu

1.5 Určení rušivých frekvencí

Pomocí funkce níže (Kód 6) a spektrogramu bylo zjištěno, že rušivé frekvence jsou:

687.5 Hz, 1359.375 Hz, 2046.875 Hz a 2734.375 Hz

Tyto frekvence byly následně zprůměrovány pro získání harmonicky vztažených frekvencí následovně:

$$f_{avg} = \frac{f_1 + f_2 + f_3 + f_4}{10}$$

$$f'_1 = f_{avg} \quad f'_2 = f_{avg} \cdot 2 \quad f'_3 = f_{avg} \cdot 3 \quad f'_4 = f_{avg} \cdot 4$$

$$f'_1 = 682.8125 \quad f'_2 = 1365.625 \quad f'_3 = 2048.4375 \quad f'_4 = 2731.25Hz$$

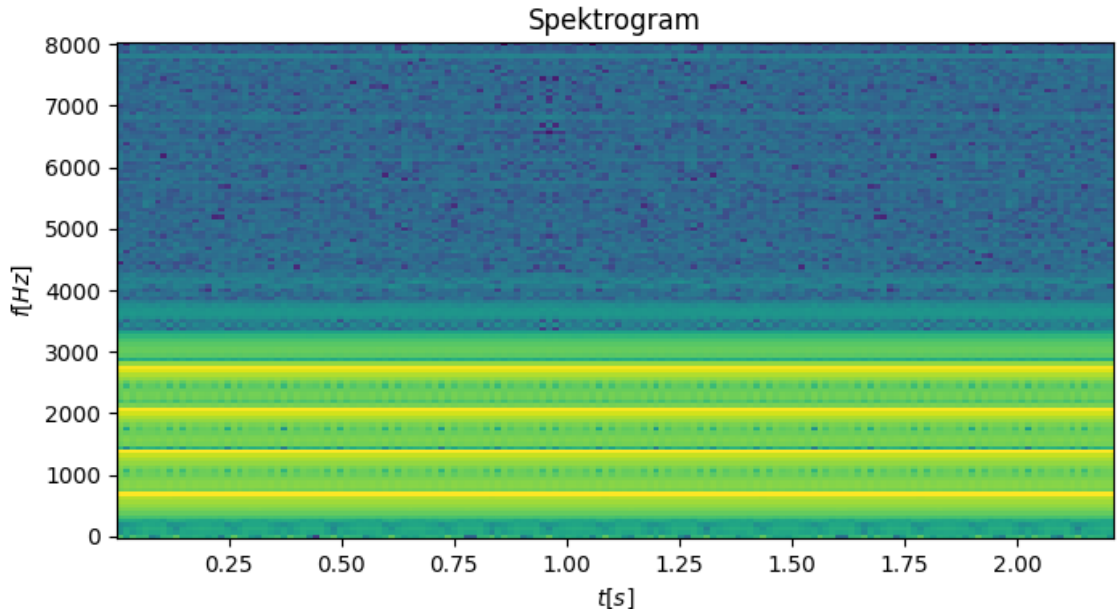
```
def get_n_max(data, n):
    arr_max = []
    for a in range(0, n):
        arr_max.append([0.0, 0.0])
    for a in range(0, data.size):
        calc_val = np.sqrt(data[a].real**2 + data[a].imag**2)
        for i in range(0, n):
            if calc_val > arr_max[i][0]:
                arr_max[i + 1:n] = arr_max[i:n - 1]
                arr_max[i] = [calc_val, a]
                break
    return arr_max
```

Kód 6: Funkce pro nalezení 'n' největších hodnot z výsledku DFT/FFT

1.6 Generování signálu

```
def gen_cosine_waves(fs, data, F):
    AMPL = 500
    t = data.size / fs
    samples = np.arange(t * fs) / fs
    cos = [
        np.int16(np.cos(2 * np.pi * F[0] * samples) * AMPL),
        np.int16(np.cos(2 * np.pi * F[1] * samples) * AMPL),
        np.int16(np.cos(2 * np.pi * F[2] * samples) * AMPL),
        np.int16(np.cos(2 * np.pi * F[3] * samples) * AMPL)
    ]
    return cos

_coss = gen_cosine_waves(_fs, _data, _F)
_4cos = _coss[0] + _coss[1] + _coss[2] + _coss[3]
wavfile.write('audio/4cos.wav', _fs, _4cos)
```

Kód 7: Funkce pro generování 4 cosinusovek



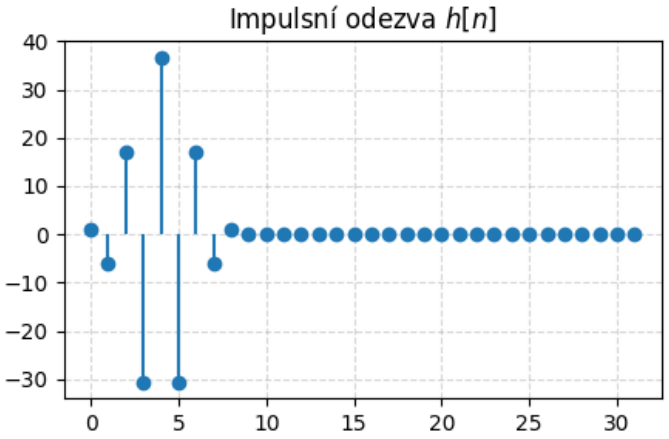
Obrázek 6: Spektrogram signálu 4 cosinusovek

1.7 Čistící filtr

Byl zvolen filtr podle první alternativy - filtr v z-rovině a byl dodržen postup, který byl v zadání navržen.

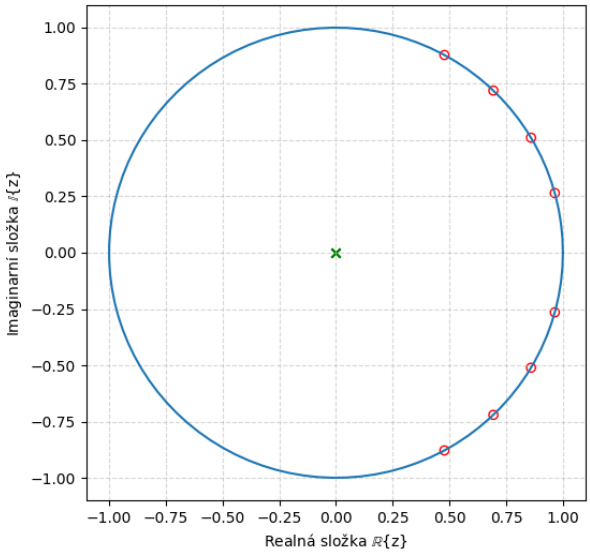
```
def z_filter_zeros(fs, F):
    omegak = []
    nk = []
    for i in range(0, len(F)):
        omegak.append(2 * np.pi * (F[i] / fs))
        nk.append(np.exp(1J * omegak[i]))
    nk = [*nk, *np.conj(nk)]
    return nk

def z_filter(nk):
    coeff = np.poly(nk)
    return coeff

_filter_zeros = z_filter_zeros(_fs, _F)
_filter_b = z_filter(_filter_zeros)
```

Kód 8: Výroba filtru v z-rovině



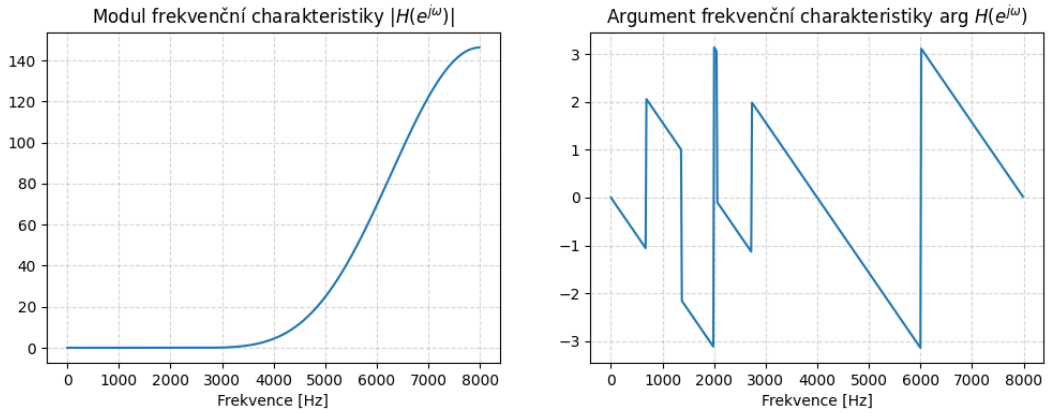
Obrázek 7: Impulzní odezva filtru v z-rovině

1.8 Nulové body a póly



Obrázek 8: Nulové body a póly filtru (kolečka - nuly, křížky - póly)

1.9 Frekvenční charakteristika



Obrázek 9: Frekvenční charakteristika filtru

1.10 Filtrace

Vstupní signál je vyfiltrován pomocí konvoluce vstupního normalizovaného signálu a koeficientů filtru v z-rovině. Následně je signál vynásoben maximální absolutní hodnotou vstupního signálu (pro zpětný přechod z dynamického rozsahu -1 až 1).

```
1 _out = np.int16(np.convolve(_data, _filter_b) * _abs_max)
```

Kód 9: Filtrování

Výsledek filtrování je však zklamáním. Výsledný signál je totiž poměrně dost zkreslený. Malé zkreslení bylo očekávané, ale zkreslení vypadá, že je až příliš velké. Jsem v domnění, že pravděpodobně došlo někde k chybě při mém postupu, jelikož i frekvenční charakteristika vypadá pochybně.