Signály a systémy

2021/2022

Projekt ISS

Lucie Svobodová (xsvobo1x) 29. 12. 2021

Obsah

1 Základy

Projekt jsem se rozhodla implementovat v jazyce Python. Zdrojový soubor je uložen v `src/iss.py`.

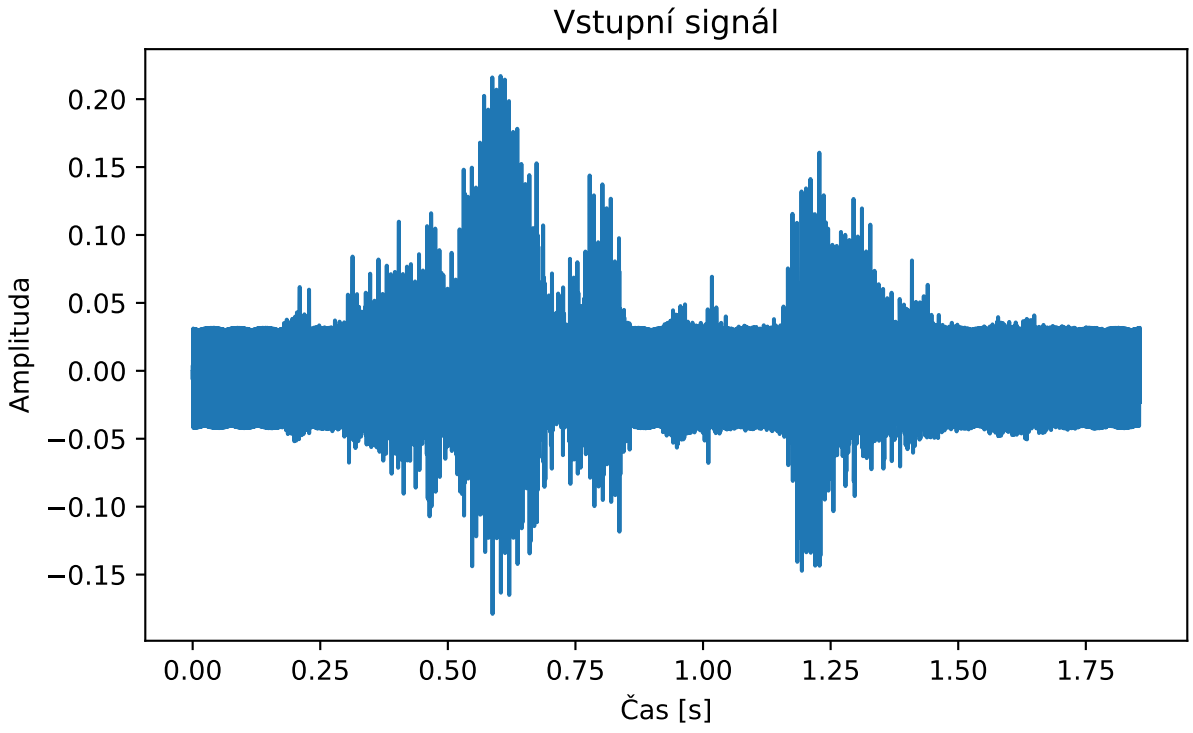
Pro načtení vstupního signálu jsem použila funkci `read()` z knihovny `scipy.io.wavfile()` [5]. Následující dala jsem zjistila z načteného signálu:

Délka [vzorků]	29696
Délka [s]	1.856
Minimální hodnota	-0.178924560546875
Maximální hodnota	0.217010498046875
Vzorkovací frekvence [Hz]	16000

Signál jsem následně zobrazila do grafu pomocí funkce `plot_graph()`, kterou jsem si na zobrazování signálů naprogramovala:

```
1     # funkce ulozi graf
2     def plot_graph(time, title, filename, xlabel, ylabel, data):
3       plt.figure()
4       plt.title(title)
5       plt.plot(time, data)
6       plt.xlabel(xlabel)
7       plt.ylabel(ylabel)
8       plt.savefig(f'{filename}.pdf')
9
```

Zde je vidět výsledný graf:



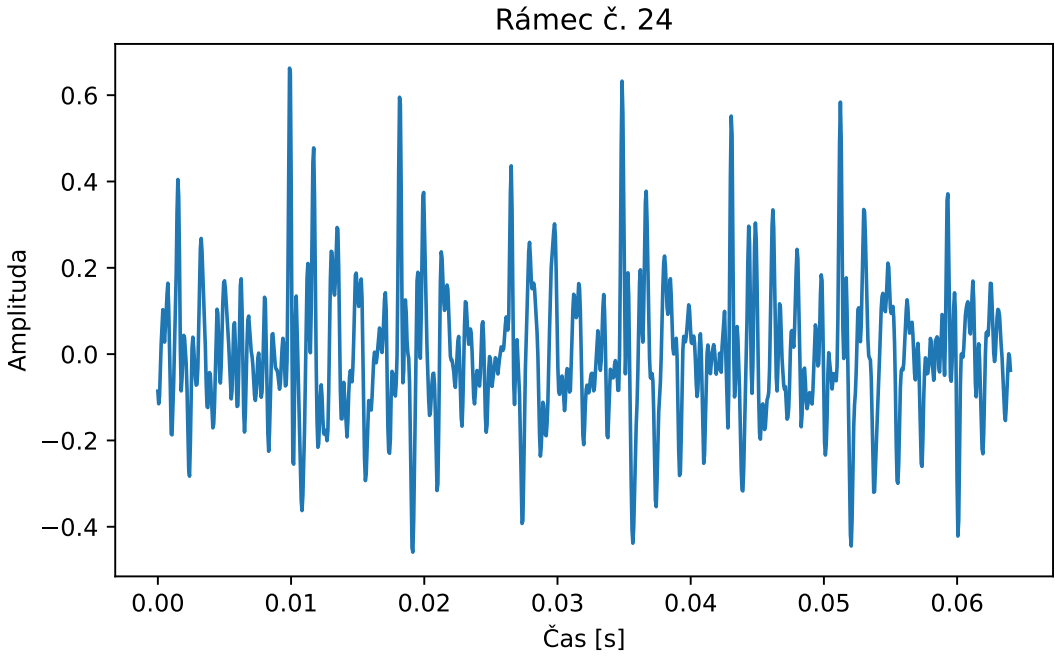
2 Předzpracování a rámce

Vstupní signál jsem s pomocí funkce `numpy.mean()` [7] ustřednila a následně normalizovala dělením maximem absolutní hodnoty, přičemž jsem využila funkce `max()` a `numpy.absolute()` [7].

Dále jsem signál rozdělila na rámce. K tomu jsem si nejdříve vytvořila matici o velikosti odpovídající budoucím vzniklým rámcům. Dále jsem použila `for` cyklus, ve kterém jsem postupně ukládala jednotlivé rámce jako vektory do předpřipravené matice. Nakonec jsem tuto matici transponovala.

```
    # ustredneni signalu
    data = data - np.mean(data)

    # normalizace
    data_abs = np.absolute(data)
    data = data/max(data_abs)

    # rozdeleni signalu na ramce
    frame_len = 1024
    overlap = 512
    frames = [[0] * frame_len for i in range((len(data)//frame_len) * 2 - 1)]
    j = 0
    for i in range(0, len(data) - overlap, overlap):
      frames[j] = data[i : i+frame_len]
      j += 1

    # ulozeni ramcu do matice a jeji transponovani
    frames = np.array(frames)
    frames = frames.transpose()

```

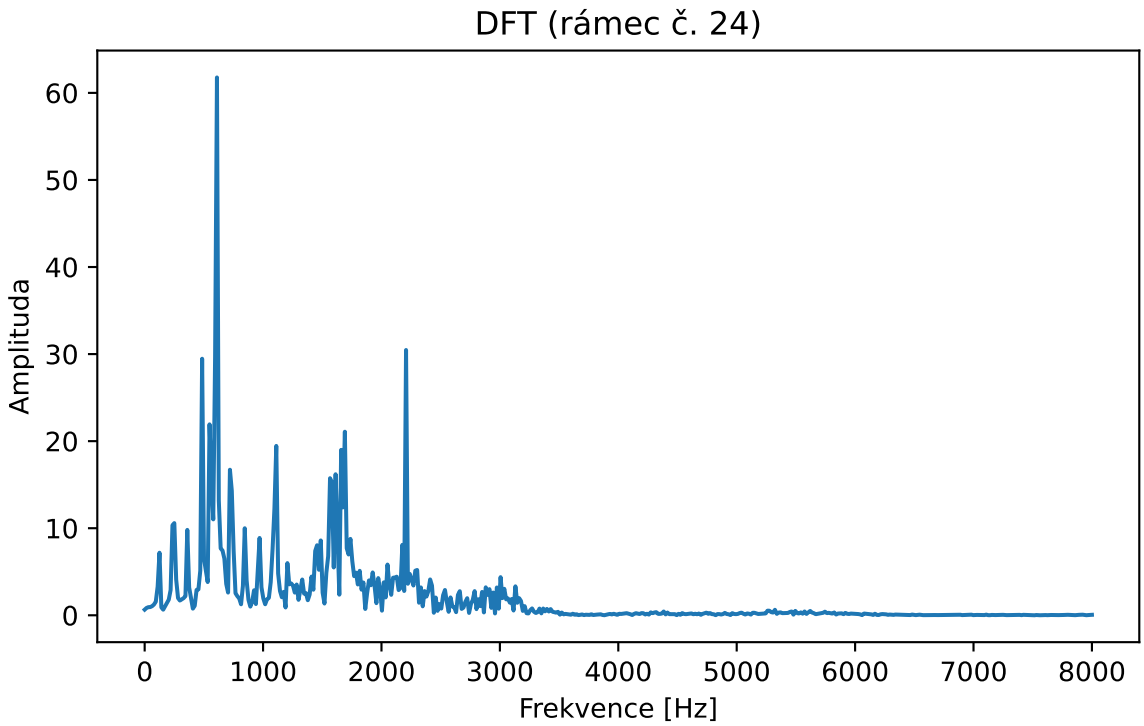
Jako pěkný rámec jsem si vybrala rámec s indexem 24, protože se v něm ve vstupním signálu vyslovuje samohláska. Tento rámec vypadá následovně:



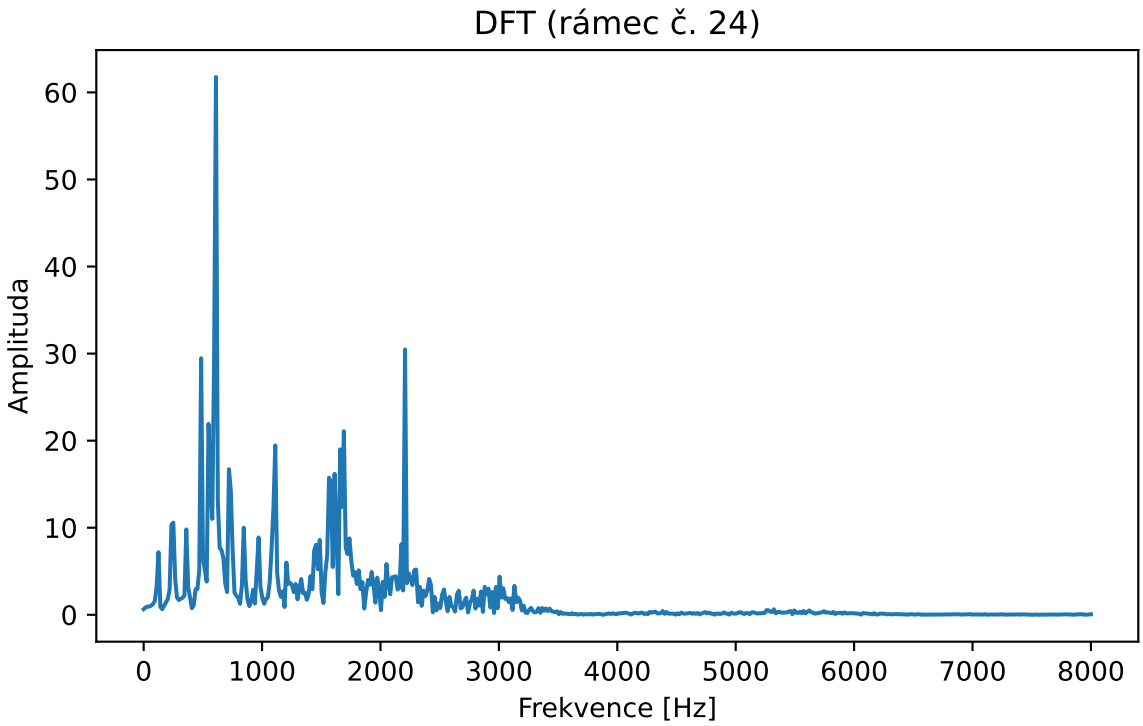
3 DFT

Pro výpočet DFT rámce s indexem 24 jsem si nejdříve naprogramovala vlastní funkci, přičemž jsem se inspirovala funkcí pro výpočet DFT z [8]. Poté jsem si nechala výsledný graf vytisknout. Dále jsem provedla na původní data DFT pomocí funkce `numpy.fft.fft()`. Grafy jsem mezi sebou porovnala, čímž jsem si ověřila, že i první způsob výpočtu je správný (výsledné grafy jsou identické).

```
# funkce vypocita DFT pro zadana data, zobrazi graf
# nasledne vypocita DFT pomoci funkce numpy.fft.fft() a zobrazi graf
def DFT(data, fs):
  n = np.arange(len(data))
  k = n.reshape((len(data), 1))
  M = np.exp(-2j * np.pi * k * n / len(data))
  res = np.dot(M, data)
  # zobrazeni grafu DFT
  time = np.linspace(0,fs/2, num=len(data)//2)
  plot_graph(time, 'DFT (r mec    . 24)', 'task3_1', 'Frekvence [Hz]',
    'Amplituda', np.abs(res[0:len(data)//2]))

  # vypocet DFT pomoci numpy.fft.fft()
  data = np.fft.fft(data)
  time = np.linspace(0,fs/2, num=len(data)//2)
  plot_graph(time, 'DFT (r mec    . 24) - numpy.fft.fft()', 'task3_2',
    'Frekvence [Hz]', 'Amplituda', np.abs(data[0:len(data)//2]))

```



DFT pomocí výpočtu podle vzorce (první způsob)



DFT pomocí numpy.fft.fft() (druhý způsob)

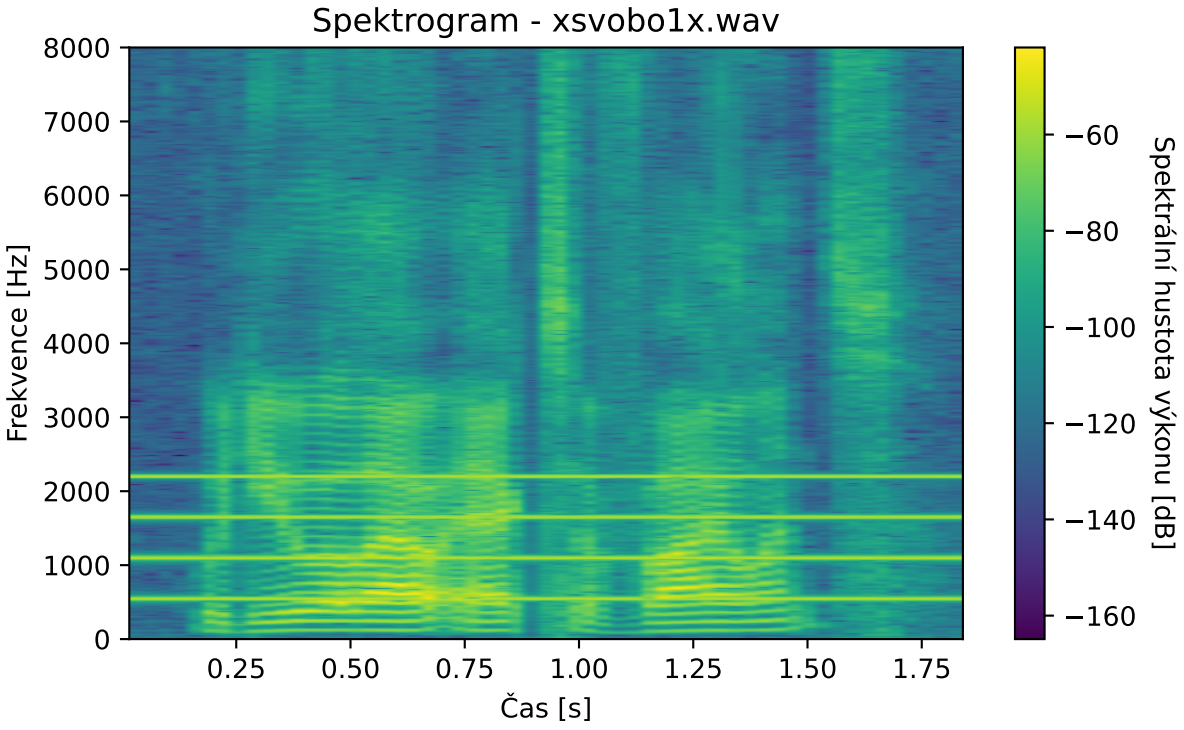
4 Spektrogram

Pro zobrazení spektrogramu jsem využila nejdříve funkci `spectrogram()`, kterou používala i paní inženýrka Žmolíková [2]. Později jsem ale našla funkci `matplotlib.pyplot.specgram()` [6], jejíž výsledky se mi líbily více, a rozhodla jsem se ji pro výsledné zobrazení spektrogramu signálu použít.

Na zobrazení spektrogramu jsem si vytvořila funkci, protože spektrogram v tomto projektu budu zobrazovat ještě vícekrát.

```
    # funkce ulozi spektrogram
    def plot_spectrogram(data, fs, filename, title):
      plt.figure(figsize=(7,4))
      plt.tight_layout()
      plt.specgram(data, Fs=fs, NFFT=1024, noverlap=512, mode='psd', scale='dB')
      plt.title(title)
      plt.gca().set_xlabel('Cas [s]')
      plt.gca().set_ylabel('Frekvence [Hz]')
      cbar = plt.colorbar()
      cbar.set_label('Spektr. hustota vykonu [dB]', rotation=270, labelpad=15)
      plt.savefig(f'{filename}.pdf')

```

Na následujícím snímku je zobrazen spektrogram vstupního signálu.



5 Určení rušivých frekvencí

Na spektrogramu jsou jasně viditelné čtyři žluté přímky - rušivé komponenty. Z grafu jsem vyčetla jejich frekvence:

f1	555 Hz
f2	1110 Hz
f3	1665 Hz
f4	2220 Hz

Z hodnot je patrné, že jsou to 4 harmonicky vztažené cosinusovky, tedy platí pro ně následující vztahy:

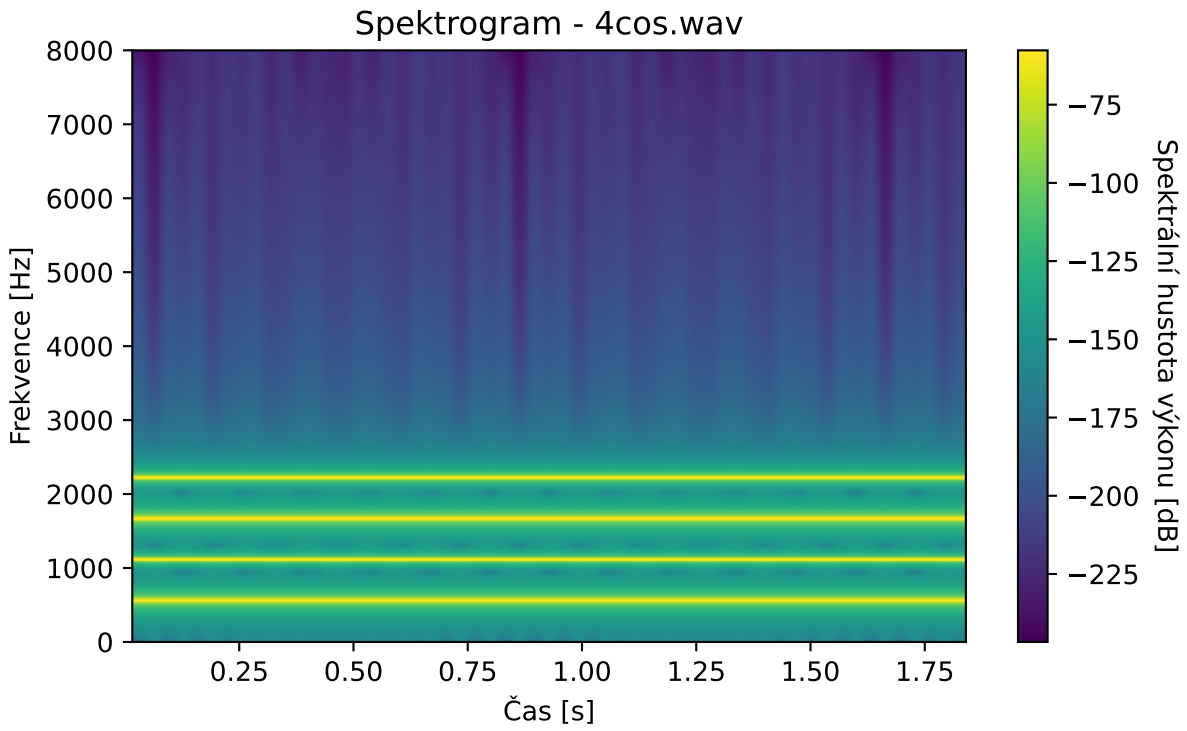
$$f1 = f1 * 1$$
$$f2 = f1 * 2$$
$$f3 = f1 * 3$$
$$f4 = f1 * 4$$

6 Generování signálu

Nový signál složený ze směsi 4 cosinusovek jsem vygenerovala pomocí funkce, ve které jsem si postupně vždy vytvořila cosinusovku s danou frekvencí a následně ji přičetla k již vzniklému signálu. Jako amplitudu jsem zvolila 0.09, protože se podle mne takto vygenerovaný signál nejvíce přiblížil původnímu signálu.

```
1     # funkce vygeneruje novy signal - smes 4 cosinusovek
2     def generate(signal_length, fs, f1):
3       time =  np.arange(signal_length)
4       signal = 0
5
6       for i in range(1, 5):
7         signal += 0.009 * np.cos((2 * np.pi) * (f1 * i)/fs * time)
8
9       plot_spectrogram(signal, fs, 'task6', 'Spektrogram - 4cos.wav')
10
11      wavfile.write('../audio/4cos.wav', fs, signal)
12
```

Spektrogram vygenerovaného signálu je zobrazen na následujícím obrázku.



Výsledný signál jsem uložila jako `audio/4cos.wav`. Na spektrogramu je vidět, že rušivé frekvence jsou určeny správně.

7 Čistící filtr

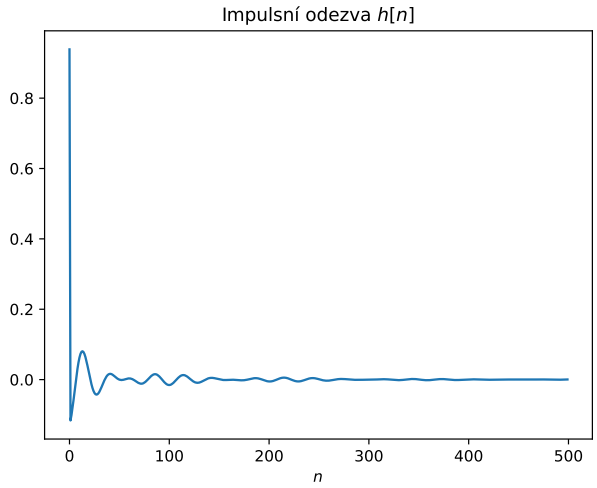
Pro výrobu filtru jsem si vybrala postup pomocí návrhu čtyř pásmových zádrží. Pro jejich vytvoření jsem využila funkce `scipy.signal.buttord()` a `scipy.signal.butter()` [5]. Dosazovanými hodnotami jsem se inspirovala v zadání projektu. Následující funkce vytvoří vždy jednu pásmovou zádrž, zobrazí její impulsní odezvu, zobrazí nuly a póly a frekvenční charakteristiku. Následně pomocí právě vytvořené pásmové zádrže vyfiltruje vstupní data. Tuto funkci používám v cyklu, díky čemuž jsou vyfiltrována vstupní data pomocí čtyř těchto pásmových zádrží. Více informací o filtrování celého signálu bude uvedeno v následujících kapitolách.

```
1     # funkce vyfiltruje data pomoci jedne pasmove zadrze
2     def bandstop_filter(data, fs, f, i):
3       nyq = 0.5 * fs
4       wp = [(f - 65)/nyq, (f + 65)/nyq]
5       ws = [(f - 15)/nyq, (f + 15)/nyq]
6       N, Wn = signal.buttord(wp, ws, 3, 40)
7       b, a = signal.butter(N, Wn, 'bandstop', output='ba')
8
9       impulse_response(b, a, i)
10      zeros_and_poles(b, a, i)
11      frequency_response(fs, b, a, i)
12
13      y = signal.lfilter(b, a, data)
14
15      return y
16
```

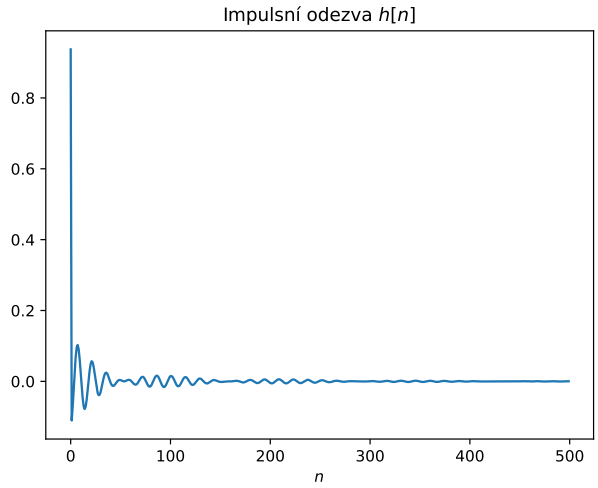
Pro označení jednotlivých filtrů bude v následujících kapitolách používáno označení:
Pásmová zádrž 1 - filtr, který nepropouští frekvence kolem frekvence f1 (555 Hz).
Pásmová zádrž 2 - filtr, který nepropouští frekvence kolem frekvence f2 (1110 Hz).
Pásmová zádrž 3 - filtr, který nepropouští frekvence kolem frekvence f3 (1665 Hz).
Pásmová zádrž 4 - filtr, který nepropouští frekvence kolem frekvence f4 (2220 Hz).

Přesné hodnoty koeficientů těchto filtrů jsou uloženy v souboru `src/coefficients.txt`. Koeficienty první pásmové zádrže (první pásmová zádrž nepropouští frekvenci f1) jsou v souboru uloženy jako `a1` a `b1`, koeficienty druhé pásmové zádrže (nepropouštějící frekvenci f2) jsou uloženy jako `a2` a `b2` atd.

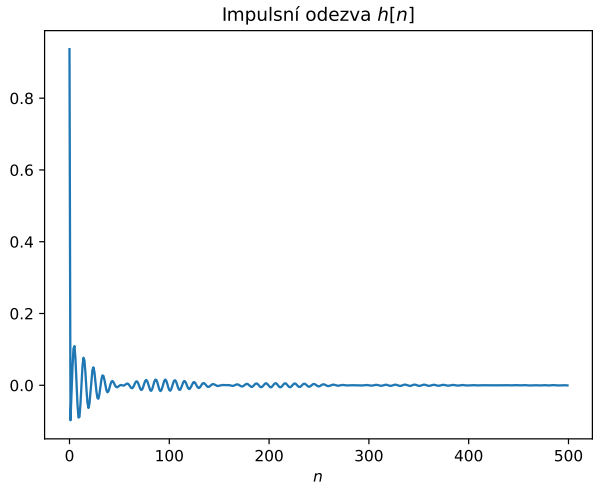
Na následujících obrázcích jsou zobrazeny impulsní odezvy jednotlivých filtrů. U programování funkce `impulse_response()` jsem se inspirovala v materiálech paní inženýrky Žmolíkové [2].



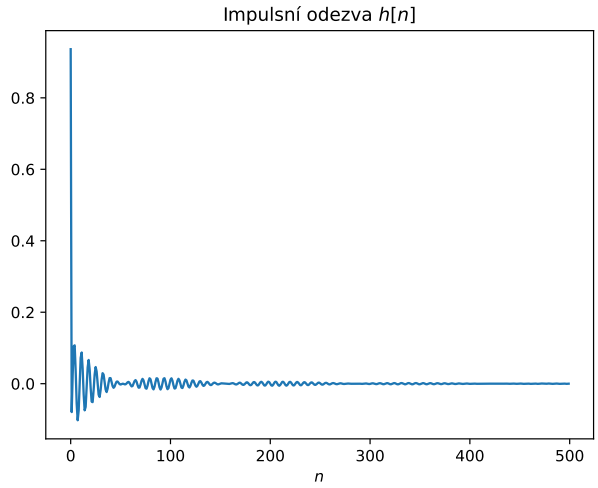
Pásmová zádrž 1



Pásmová zádrž 2



Pásmová zádrž 3



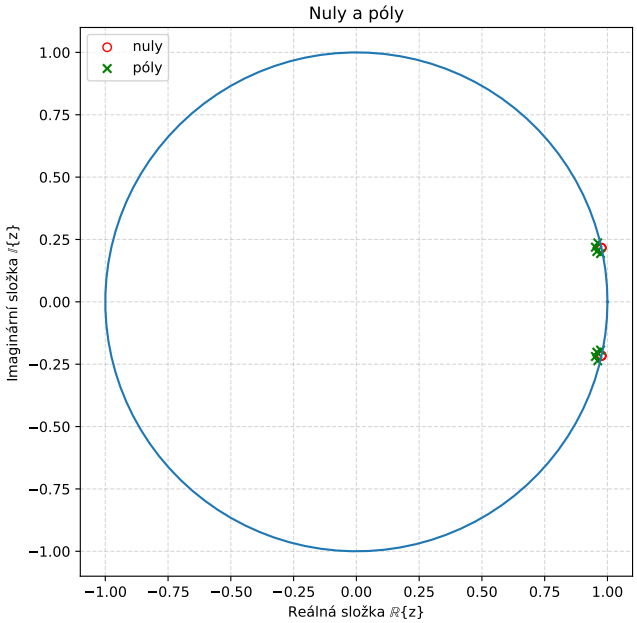
Pásmová zádrž 4

8 Nulové body a póly

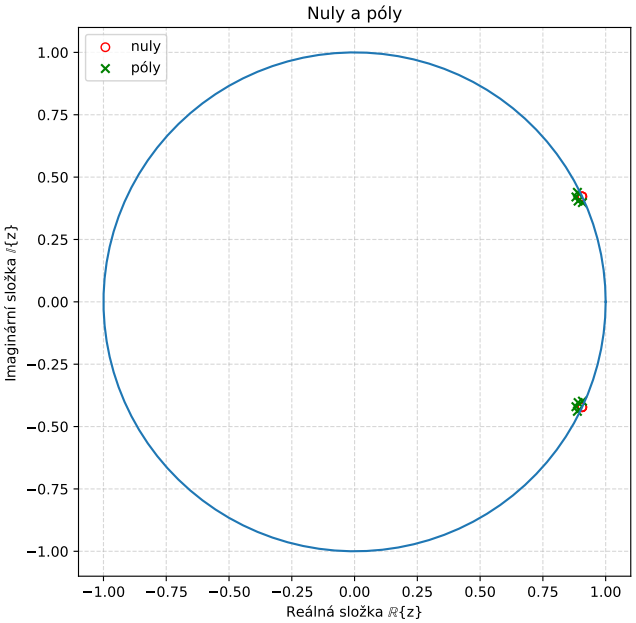
Vzhledem k tomu, že jsem navrhla filtr pomocí 4 pásmových zádrží, rozhodla jsem se nuly a póly zobrazit pro každý filtr zvlášť. Funkci pro výpočet nul a pólů jsem převzala z materiálů paní inženýrky Žmolíkové [2]. Využila jsem funkci `scipy.signal.tf2zpk()` [5]. Funkce zároveň ověřuje stabilitu filtru.

```
    # funkce zobrazi nuly a poly jednoho filtru
    def zeros_and_poles(b, a, i):
        z, p, k = signal.tf2zpk(b, a)
        plt.figure(figsize=(7,7))

        # stabilita filtru
        is_stable = (p.size == 0) or np.all(np.abs(p) < 1)
        print('Filtr {} stabilni.'.format('je' if is_stable else 'neni'))

        # jednotkova kruznice
        ang = np.linspace(0, 2*np.pi,100)
        plt.plot(np.cos(ang), np.sin(ang))

        # nuly, poly - zobrazeni
        plt.scatter(np.real(z),np.imag(z),marker='o',facecolors='none',
            edgecolors='r',label='nuly')
        plt.scatter(np.real(p),np.imag(p), marker='x', color='g', label='poly')
        plt.gca().set_title('Nuly a poly')
        plt.gca().set_xlabel('Realna slozka $\mathbb{R}\{$z$\}$')
        plt.gca().set_ylabel('Imaginarni slozka $\mathbb{I}\{$z$\}$')
        plt.grid(alpha=0.5, linestyle='--')
        plt.legend(loc='upper left')
        plt.savefig(f'task8_{i}.pdf')

```

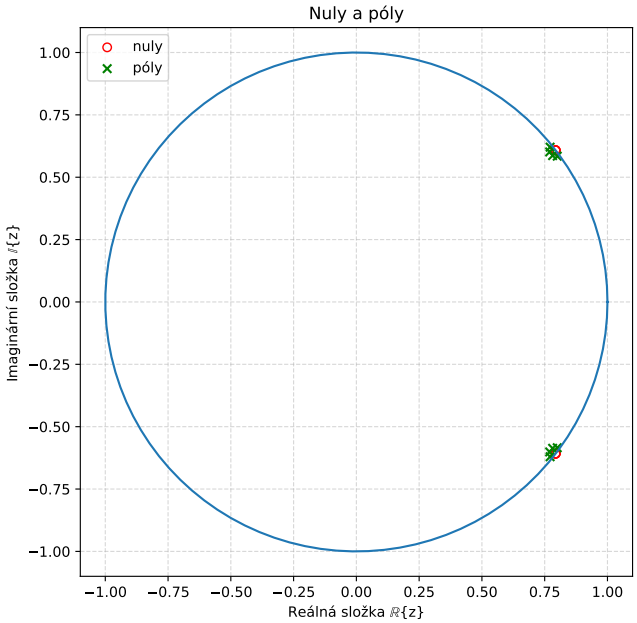
Výsledné nuly a póly pro jednotlivé filtry jsou následující:



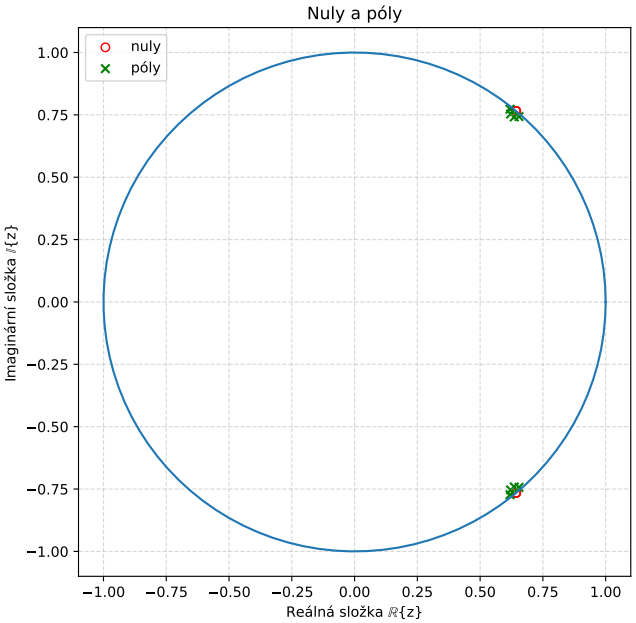
Pásmová zádrž 1



Pásmová zádrž 2



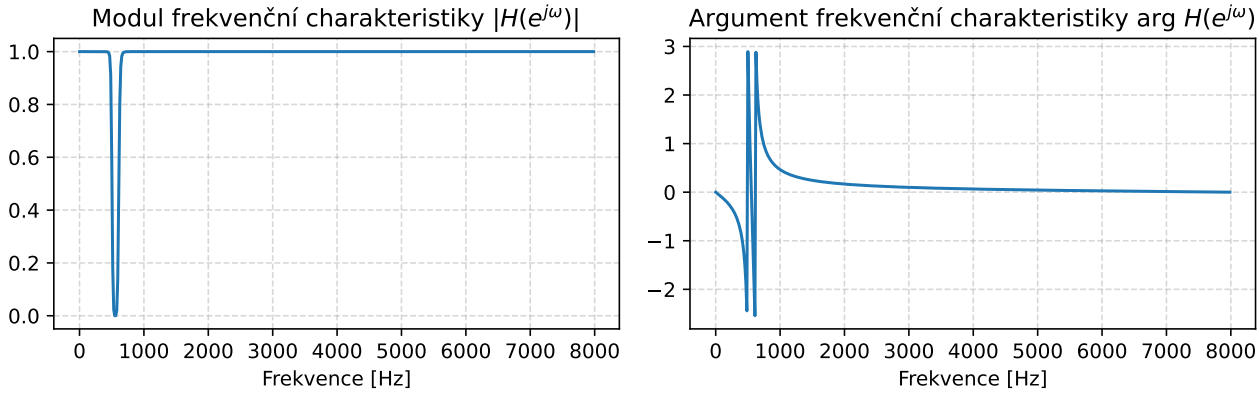
Pásmová zádrž 3



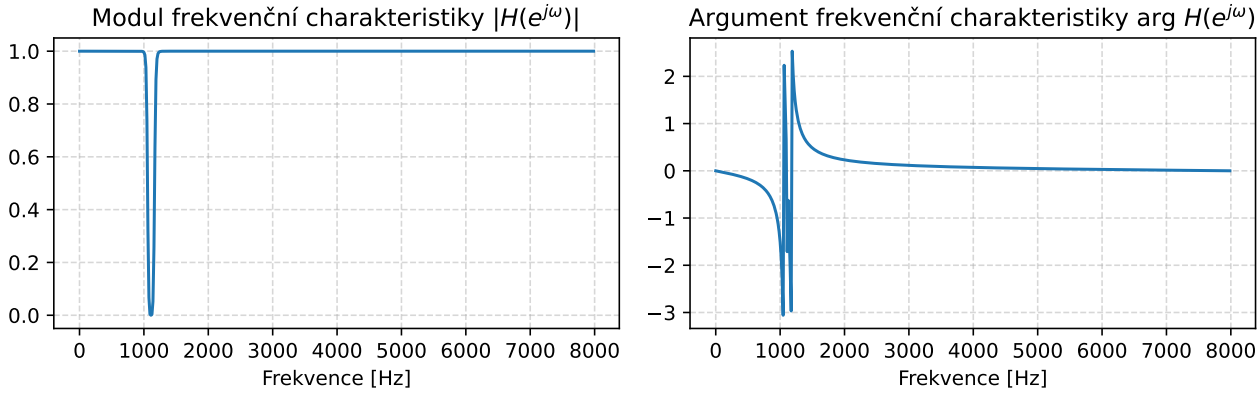
Pásmová zádrž 4

9 Frekvenční charakteristika

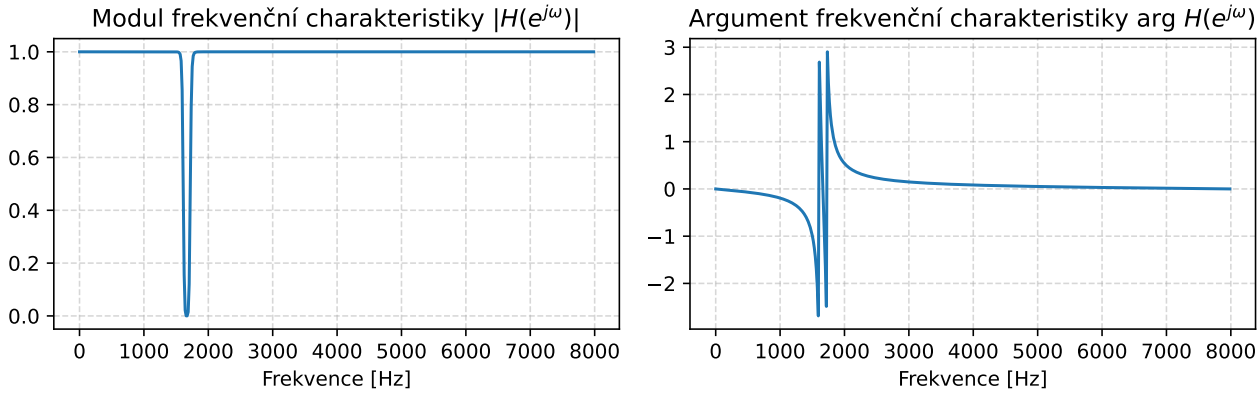
Funkci pro výpočet frekvenční charakteristiky jsem taktéž převzala z materiálů paní inženýrky Žmolíkové [2]. Stejně jako u nul a pólů mi přišlo přehlednější frekvenční charakteristiku zobrazit pro každou pásmovou zádrž zvlášť. Každou frekvenční charakteristiku zobrazuji jako modul (vlevo) a argument (vpravo).



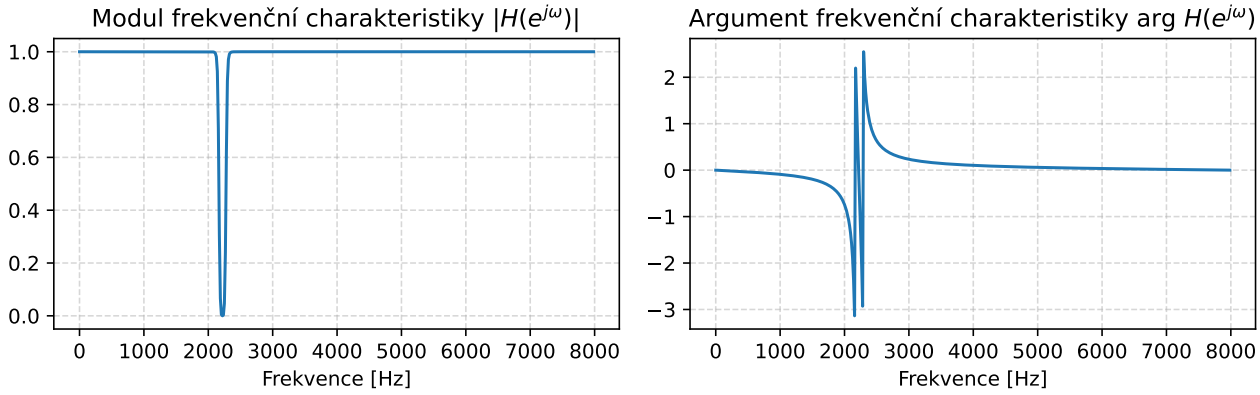
Pásmová zádrž 1



Pásmová zádrž 2



Pásmová zádrž 3



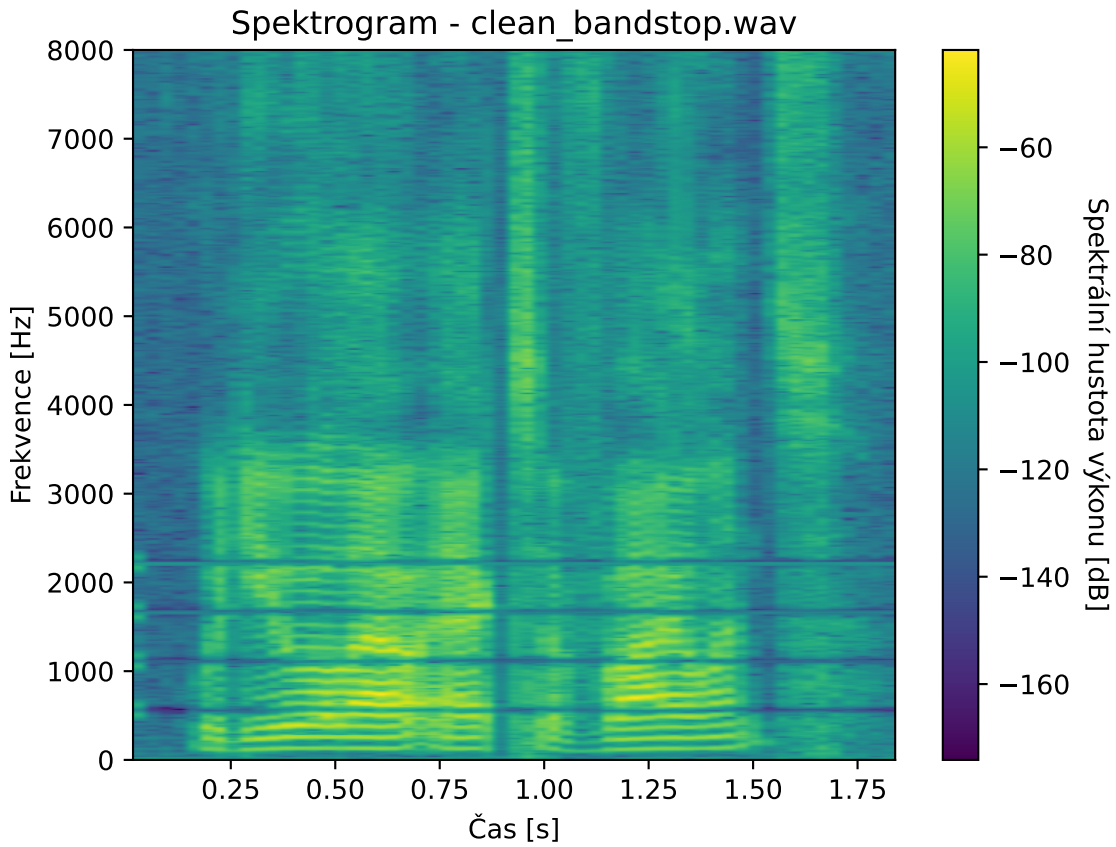
Pásmová zádrž 4

10 Filtrace

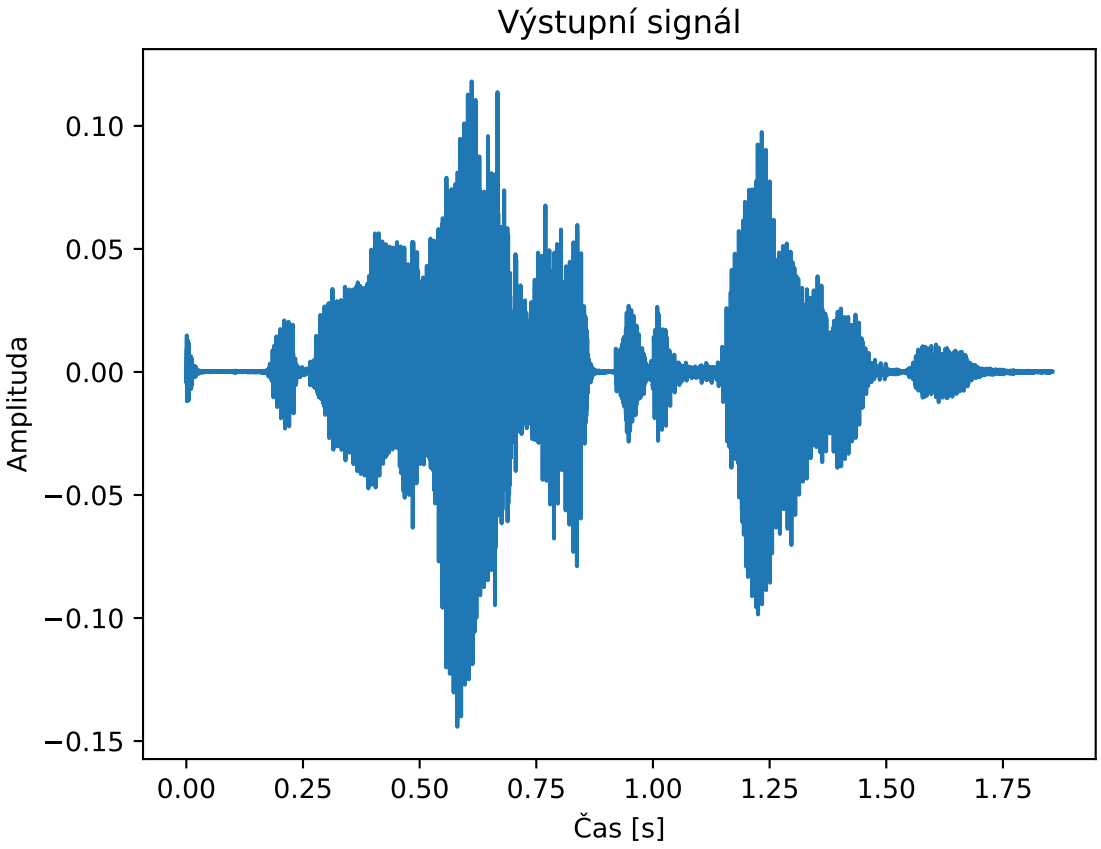
Výsledný signál jsem vyfiltrovala pomocí dříve popsaného navrženého filtru.

```
    # funkce pouzije 4 pasmove zadrze na vstupni signal
    def filtering(data, fs, f1):
        y = data

        for i in range(1, 5):
            y = bandstop_filter(y, fs, f1 * i, i)

        plot_spectrogram(y, fs, 'task10_2', 'Spektrogram - clean_bandstop.wav')
        wavfile.write('../audio/clean_bandstop.wav', fs, y)

        return y

```

Zobrazila jsem si spektrogram výstupního signálu, na kterém je krásně vidět, že byly rušivé frekvence odfiltrovány.



Dále jsem zobrazila výstupní signál stejně, jako jsem to udělala na začátku se vstupním signálem. Je vidět, že signál je nyní mnohem čistší.



Tento rozdíl je znatelný i při poslechu vstupního a výstupního signálu. Ve vstupním signálu je možné slyšet již zmiňované rušivé frekvence, které bránily jednoduchému rozpoznání textu. Tyto rušivé frekvence po filtraci úplně zmizely a výstupní signál je čistý a je jednoduché rozpoznat text v nahrávce. Výstupní signál je uložen v `audio/clean_bandstop.wav`.

11 Použitá literatura

[1] https://nbviewer.org/github/zmolikova/ISS_project_study_phase/blob/master/Obecne_Python_tipy.ipynb

[2] https://nbviewer.org/github/zmolikova/ISS_project_study_phase/blob/master/Zvuk_spektra_filtrace.ipynb

[3] https://www.fit.vutbr.cz/study/courses/ISS/public/opora/iss.pdf

[4] https://www.fit.vutbr.cz/study/courses/ISS/public/proj_studijni_etapa/3_sound/3_sound.pdf

[5] https://docs.scipy.org/doc/

[6] https://matplotlib.org/

[7] https://numpy.org/

[8] https://books.google.cz/books?id=GWzqDwAAQBAJ&pg=PA425&lpg=PA425&dq=dft+np,arange+np.transform+np.dot&source=bl&ots=H4bQSF1f16&sig=ACfU3U3HBaVXVFjr9wwkENQxJQmBdoK1Vw&hl=cs&sa=X&ved=2ahUKEwiHgfeTyof1AhWsRPEDHcy1BaYQ6AF6BAgSEAM#v=onepage&q=dft%20np%2Carange%20np.transform%20np.dot&f=false