SIGNÁLY A SYSTÉMY 2021/2022

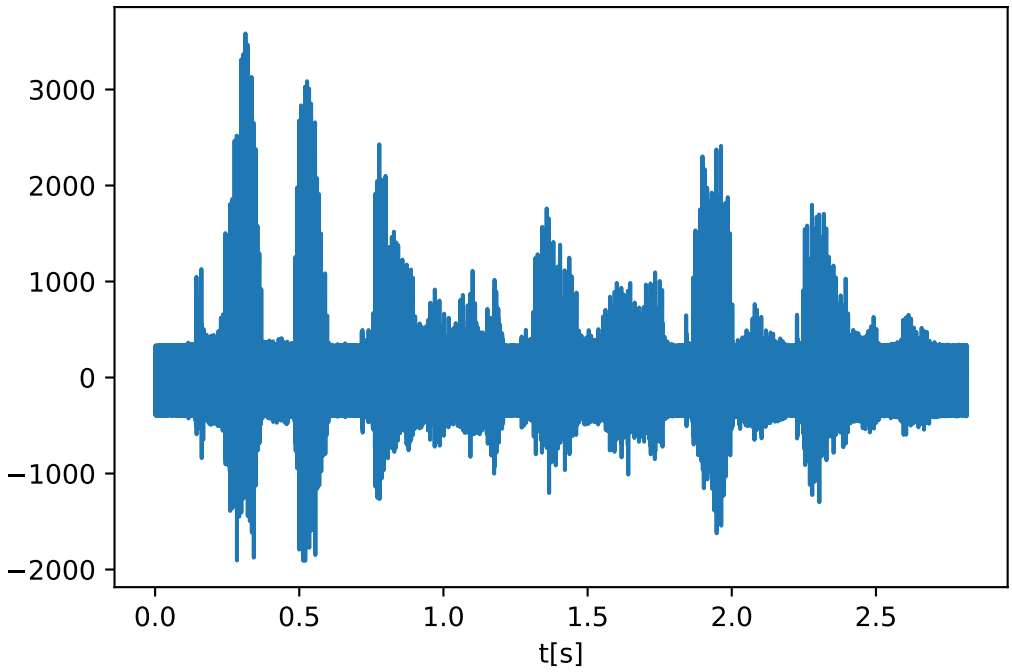
Dokumentace semestrálního projektu

Ondřej Mach (xmacho12)

Obsah

1 Základy

Pro načtení signálu byl použit modul `wavfile` z knihovny `scipy.io`. Načtený signál má **45056 vzorků**. Počet vzorků vydělíme vzorkovací frekvencí a získáme tím délku signálu **2.816 s**. Hodnoty signálu se pohybují v rozmezí od **-1909** do **3584**.



2 Předzpracování a rámce

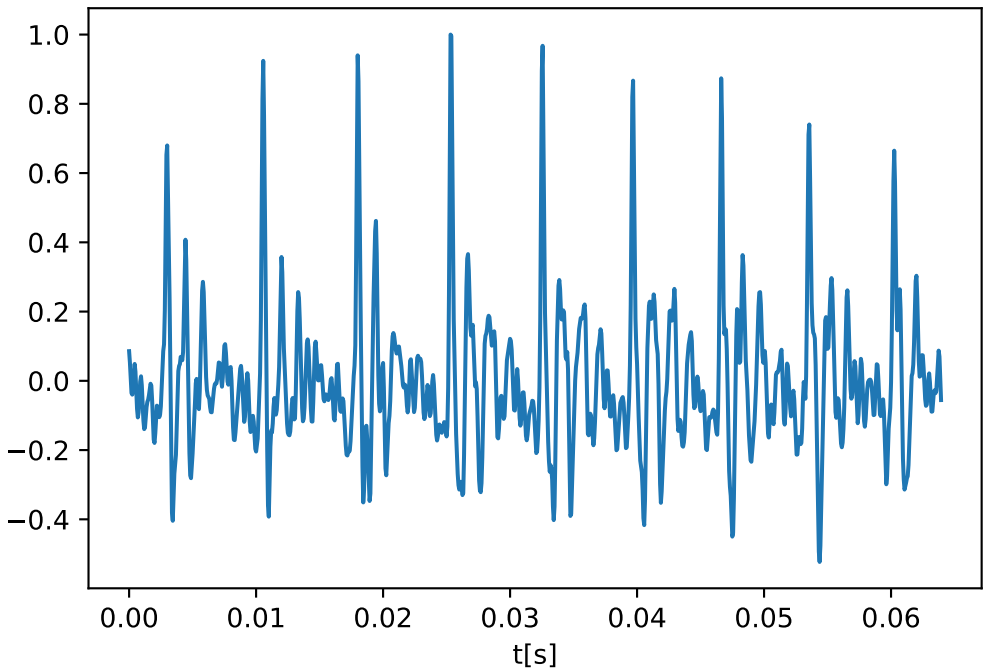
Signál je neprve ustředněn a normalizován.

```
data = data - data.mean()
scale = np.abs(data).max()
data = data / scale
```

Dále je signál rozdělen na kousky o velikosti 512 pomocí metody `np.ndarray.resize`. Je vytvořena prázdná matice `frames`, ve které každý sloupec bude obsahovat jeden rámec. Nakonec for cyklus projde všechny rámce a do každého zapíše konkatenaci dvou odpovídajících kousků.

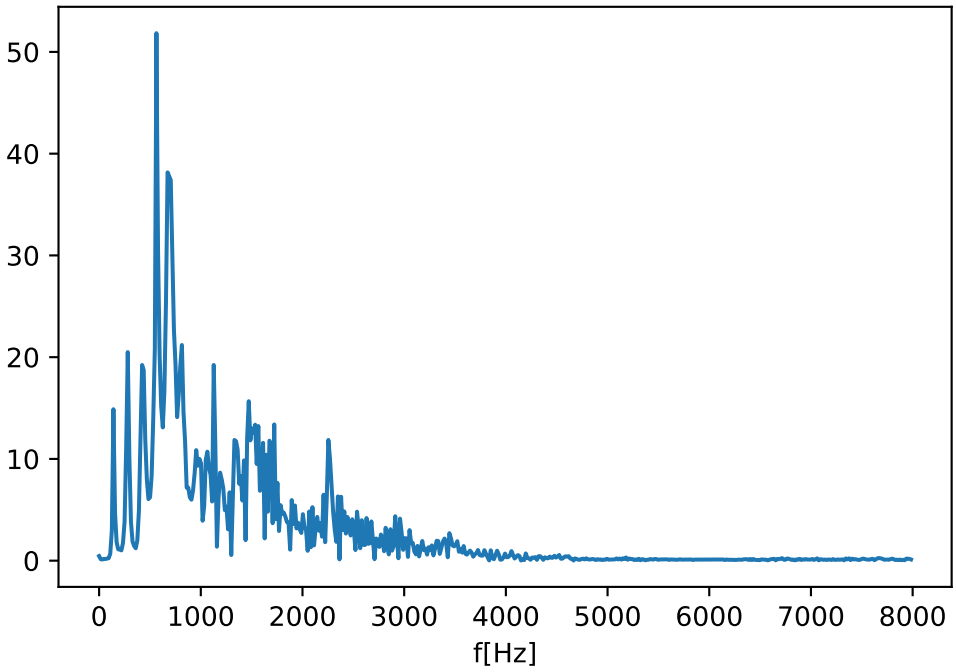
```
chunkedArray = data.reshape(numChunks, chunkSize)
frames = np.empty((numChunks-1, frameSize), dtype=float)

for i in range(numChunks-1):
    frames[i] = np.concatenate((chunkedArray[i], chunkedArray[i+1]))
```



3 DFT

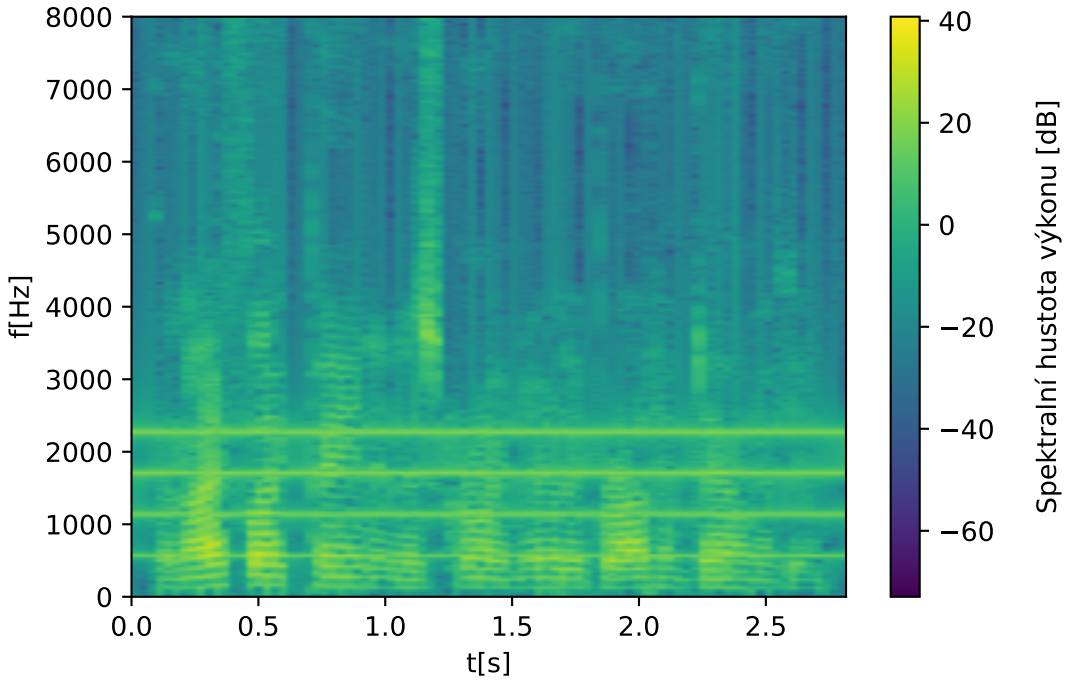
Diskrétní fourierova transformace je implementována násobení maticí `DFT_BASES`. Tato matice má velikost 1024×1024. Díky tomu lze násobit s maticí `frames`, ve které jsou uloženy rámce jako sloupcové vektory.
Je také definována funkce `DFT`, která provede maticové násobení pouze pro jeden rámec a vrátí jeho transformaci.



Výstupní koeficienty funkce DFT byly porovnány s výstupy funkce `np.fft.fft`. Vykreslené grafy jsou na první pohled stejné. Funkce `np.allclose` také potvrdila, že se výstupy shodují. Tím byla implementace prohlášena za funční.

4 Spektrogram

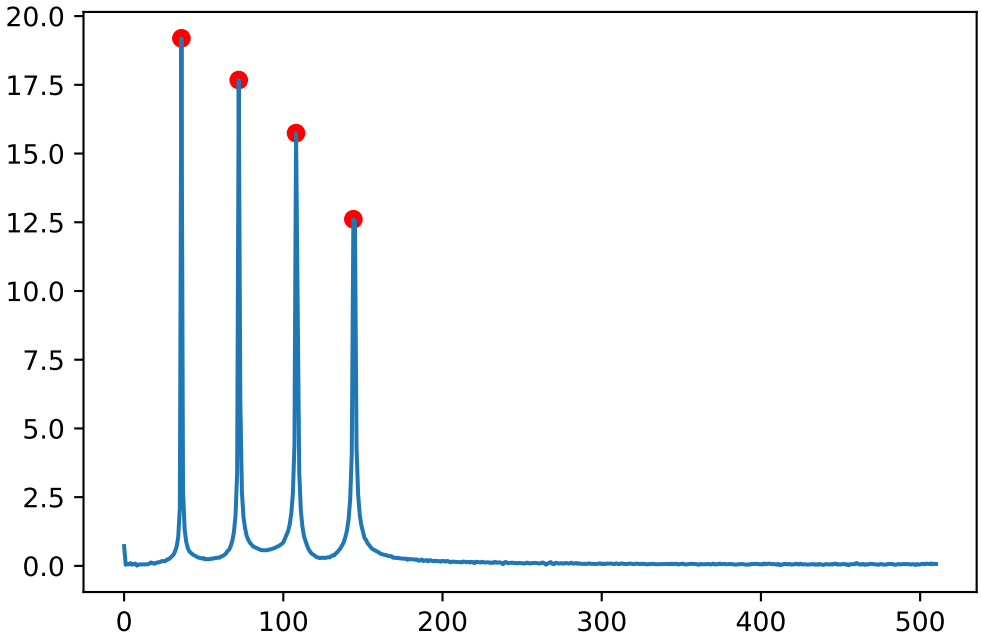
Data pro spektrogram jsou získána fourierovou trandformací všech rámců. Poté je spektrogram oříznut pouze na frekvence menší než polovina vzorkovací frekvence. Z koeficientů DFT je udělána absolutní hodnota. Jejich hodnoty jsou přepočítány na dB pro lepší dynamický rozsah při zobrazení. Poté je vykreslen graf funkcí `matplotlib.pyplot.imshow`.



5 Určení rušivých frekvencí

Pro určení rušivých frekvencí je vhodné vybrat rámec, který obsahuje pouze je. Tento rámec je pomocí DFT přeměněn na koeficienty všech obsažených frekvencí. Tuto transformaci si opět zkrátíme do poloviny vzorkovací frekvence. V našem signálu jsou rušivé frekvence pouze čtyři „ostré špičky“. Díky tomu stačí pro jejich nalezení najít čtyři největší hodnoty v poli. To jde udělat metodou `ndarray.argsort`. Z indexů už můžeme dostat hodnoty rušivých frekvencí.

```
2250.0 Hz
1687.5 Hz
1125.0 Hz
562.5 Hz
```

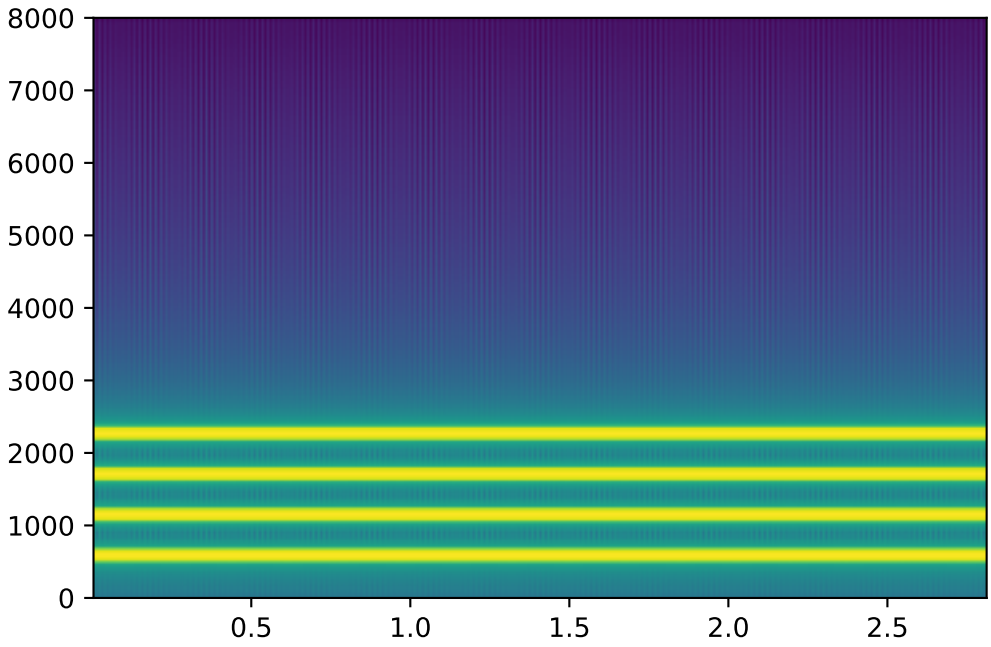


6 Generování signálu

Pro generování signálu bylo přeba inicializovat výstupní pole nulami. Poté bylo spuštěno generování každé frekvence pomocí for cyklu. Pro generovanou frekveci byla spočítána příslušná normovaná úhlová frekvence ω. Samotné generování proběhne na pouhém jednom řádku, kde je index v poli přeměněn na argument cosinusoidy a poté je vypočítána funkční hodnota.

```
signal = np.cos(np.arange(NUM_SAMPLES) * omega)
```

Tato hodnota je vydělěna počtem generovaných frekvencí a přičtena k výslednému signálu. Dělení je zde z důvodu, aby rozsah výstupního signálu zůstal v uzavřeném intervalu od -1 do 1.

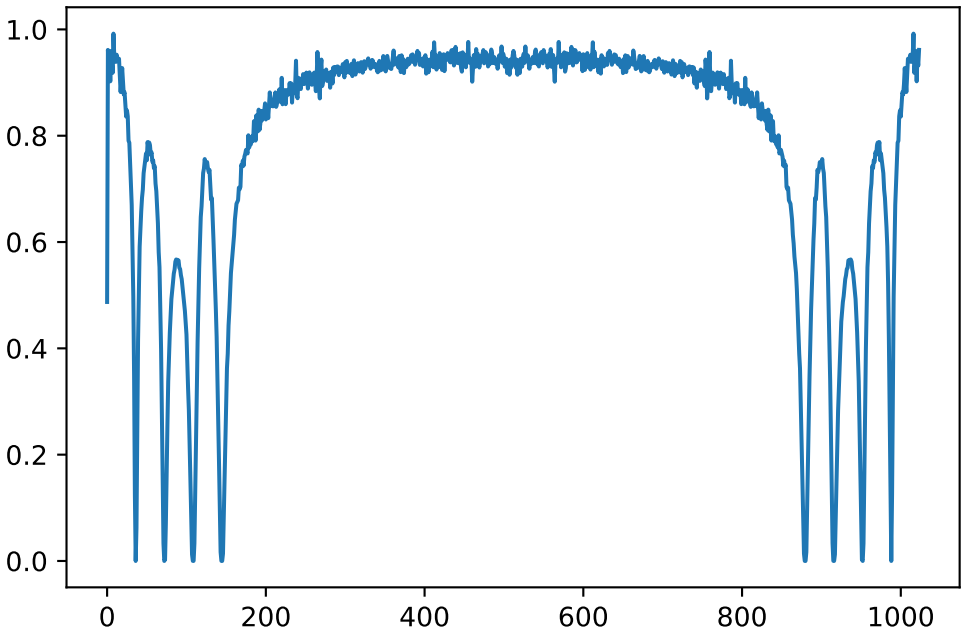


Spektrogram vygenerovaného signálu se shoduje s rušením v zadaném signálu. Na poslech je také signál velmi podobný.

7 Čistící filtr

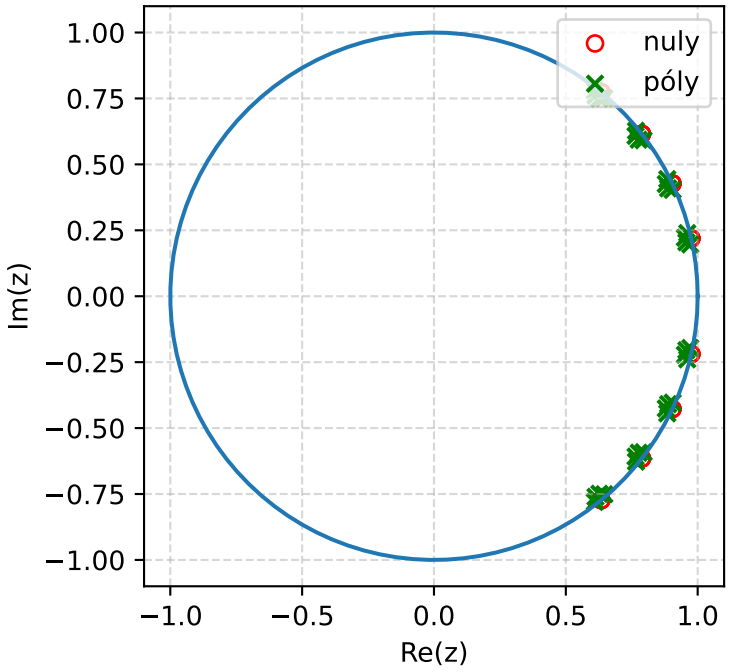
Jako čistící filtr jsem si vybral třetí možnost ze zadání - 4 pásmové zádrže. Ty jsou implementovány jako Butterworthův filtr v modulu `scipy.signal`. Parametry zádrží jsou opět určeny podle zadání. Závěrné pásmo je široké 30 Hz, s přechody širokými 50 Hz po obou stranách. Ripple v propustném pásmu je 3 dB a potlačení šumu -40 dB.
Všechny filtry jsou po spočítání jejich koeficientů `b, a` přidány do seznamu `butter_filters`.
Nad rámec jsem také vytvořil spektrální filtr. Jeho koeficienty byly vypočítány ze spektra rámce, ve kterém je pouze hluk. Transformační funkce byla navržena tak, aby vstupu od 0 do ∞ přiřazovala výstup od 1 do 0.

```
spec_filter_smart = 1 / np.exp(np.abs(np.fft.fft(frames[:, 1])))
```



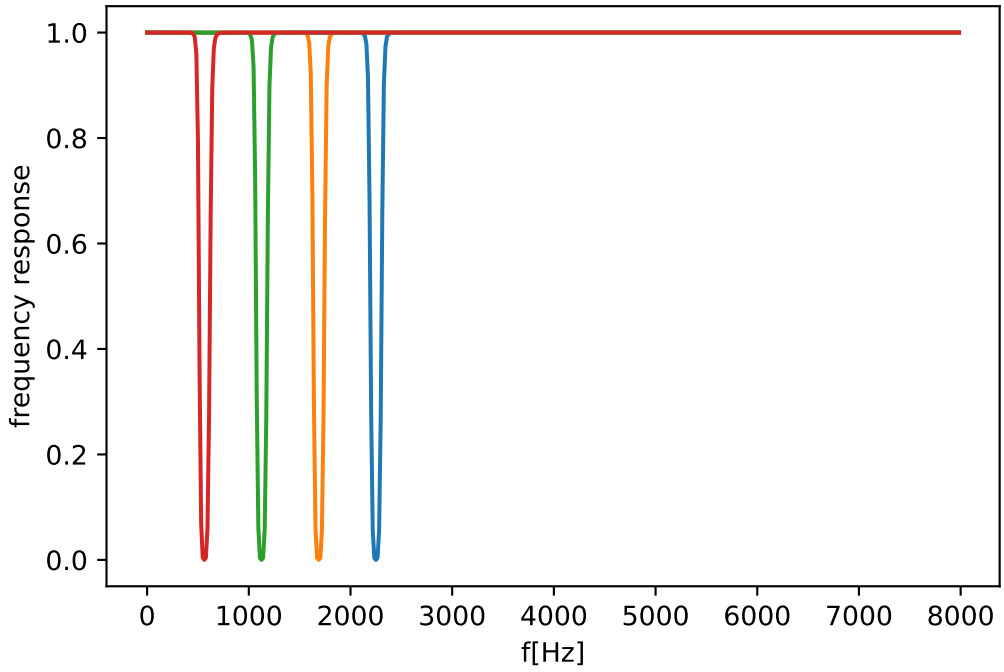
8 Nulové body a póly

Při vytváření Butterworthových filtrů byly také vygenerovány jejich nuly a póly. Ty jsou zobrazeny na jednotkové kružnici.



9 Frekvenční charakteristika

Frekvenční charakteristika Butterworthových filtrů byla získána funkcí `scipy.signal.freqz`. Ta dokáže z koeficientů `b, a` spočítat frekvenční charakteristiku.

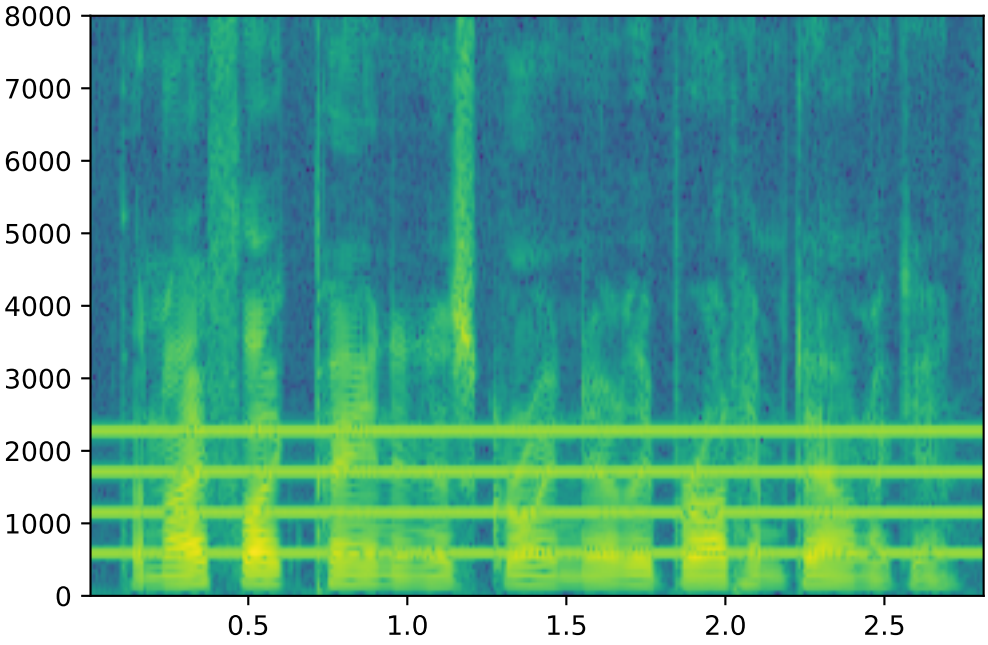


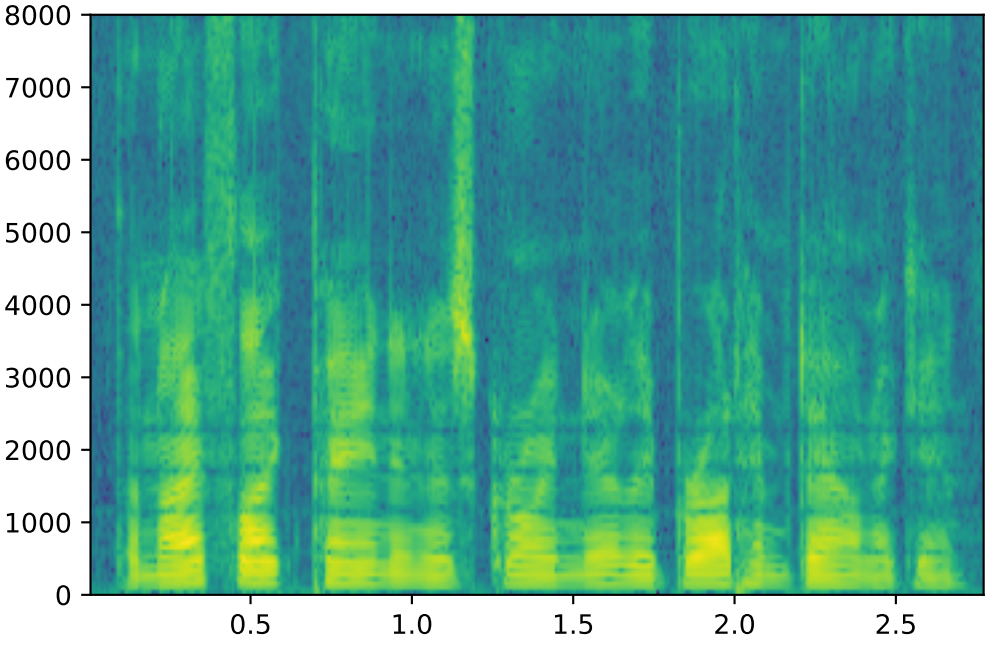
10 Filtrace

Filtrace pomocí Butterworthova filtru se ukázala být zcela triviální. Do pole `filtered_butter` je nejprve zkopírován původní signál. Poté jsou po jednom aplikovány všechny filtry. Výstupní signál je na poslech zcela čistý, nejsou slyšet žádné artefakty.

```
filtered_butter = data
for b, a in butter_filters:
    filtered_butter = scipy.signal.lfilter(b, a, filtered_butter)
```

Aplikace spektrálního filtru nebyla tak jednoduchá. Každý rámec signálu je transformován, stejně jako při tvorbě spektrogramu. Poté je každý rámec vynásoben spektrálním filtrem. Po inverzní fourierově transformaci jsou z rámce zachovány pouze reálné složky. Pokud bychom nyní spojili všechny rámce, náš signál by byl dvakrát delší než původní.
Okraje rámců ale mají nekvalitní signál, který by zanesl artefakty. Proto z každého rámce vyřízneme pouze prostředek. Když tyto části rámců dáme dohromady, dostáváme finální vyfiltrovaný signál. Kvalita je opět velice slušná, nejsou slyšitelné žadné artefakty.
Na poslech ani nebyl mezi Butterworthovým a spektrálním filtrem žádný rozdíl. Je ale velice pravděpodobné, že na „pestřejším signálu“ by se rozdíl objevil.


8000
7000
6000
5000
4000
3000
2000
1000
0
0.5
1.0
1.5
2.0
2.5


8000
7000
6000
5000
4000
3000
2000
1000
0
0.5
1.0
1.5
2.0
2.5