Dominik Horký xhorky32

Signály a systémy

Protokol s řešením projektu

FIT VUT v Brně 2021/2022

Úvod

Projekt byl řešen na Windows 10 – WSL (Ubuntu 20.04) v programovacím jazyce Python (verze 3). Po extrakci souboru s řešením projektu lze spustit projekt tak, že (v terminálu) navigujete do složky `src` a následně spustíte skript `project.py` například takto (WSL - bash):

```
python3 project.py
```

Pro správný chod skriptu jsou vyžadovány následující Python moduly:

- `NumPy`
- `MatPlotLib`
- `Scipy`

Projekt využívá i další moduly (`math`, `wave`), ty jsou ale součástí samotného Pythonu. Pokud jeden z požadovaných modulů nebudete mít nainstalovaný na svém stroji, skript vás na to upozorní a skončí předčasně. Modul nainstalujete například takto:

```
pip install [nazev]
```

Projekt není vypracován úplně celý. Vypracoval jsem pouze prvních šest úkolů, tedy po zobrazení 4 kosinusovek a vytvoření audia z nich.

Otevírání obrázků z WSL

Zobrazené grafy z projektu jsou vždy uloženy jako PNG obrázky v adresáři `out` (která je v kořenovém adresáři – archivu projektu). Zároveň jsou i zde, v tomto protokolu.

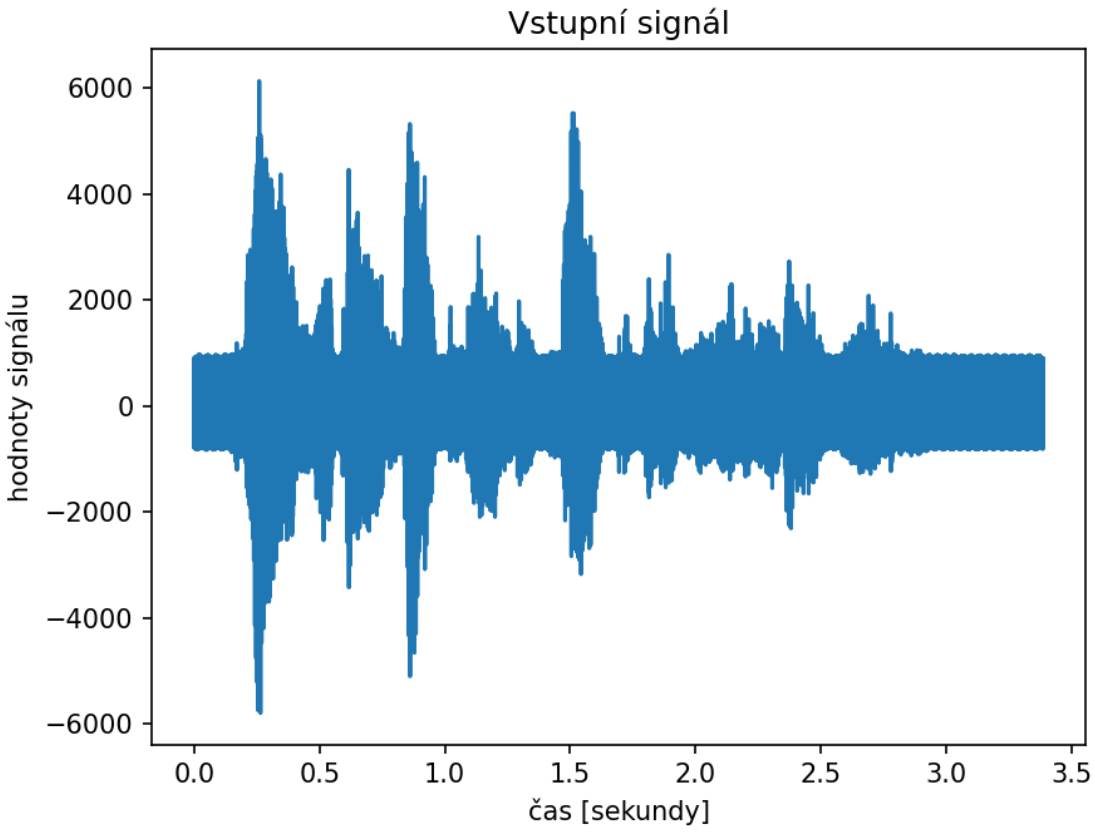
Pokud používáte Windows 10 WSL a nemáte nastavený Xserver (nebo cokoliv co vám umožní otevírat Linuxové GUI aplikace) a máte například nastavenou síťovou jednotku (disk) pro přístup k souborům z WSL pomocí Prohlížeče souborů (Windows 10), výsledné grafy či obrázky se vám nemusí zobrazit, pokud je budete otevírat z této síťové jednotky, pokud k tomuto účelu budete používat vestavěnou aplikaci Fotky (stalo se tak v mém případě).

Pokud se vám tak stane, řešení je jednoduché – stačí pro otevření obrázků použít jinou aplikaci a nebo obrázky překopírovat mimo WSL (třeba do Dokumentů).

Úkol 1: Vstupní signál

Tento signál má 54 170 vzorků a má délku zhruba 3.39 sekund. Jeho maximální hodnota je 6127, minimální pak -5792. Tyto extrémy jak je vidět z grafu signálu níže. se vyskytují někde kolem 0.25 sekundy.

Pro zjištění počtu vzorků jsem použil vestavěnou funkci knihovny *wave*.Počet sekund vypočten: *počet vzorků / vzorkovací frekvence*. Vzorkovací frekvence činí 16 kHz, což odpovídá zadání. Jelikož jsem signál uložil do numpy pole, maximální a minimální hodnotu jsem získal jednoduše pomocí funkcí *min()* a *max()*.

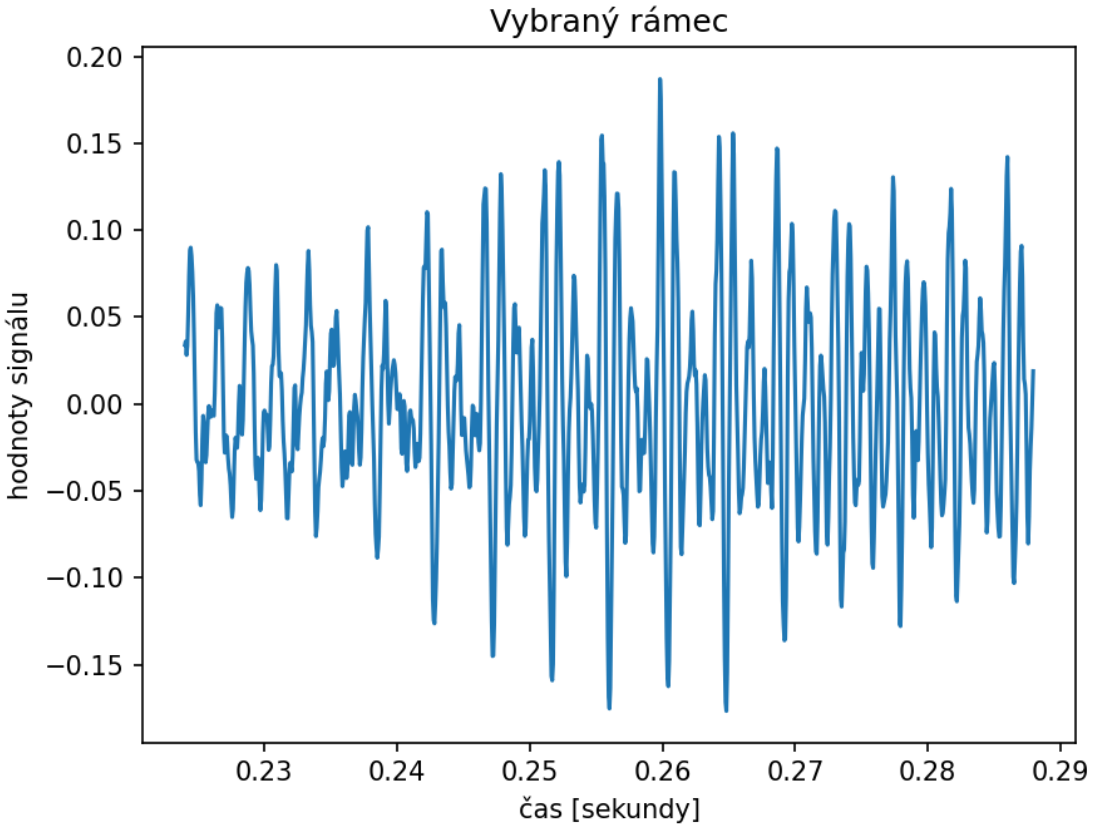


Úkol 2: Vybraný rámec s periodickým charakterem

Rámec, který jsem si vybral se nachází na začátku signálu.

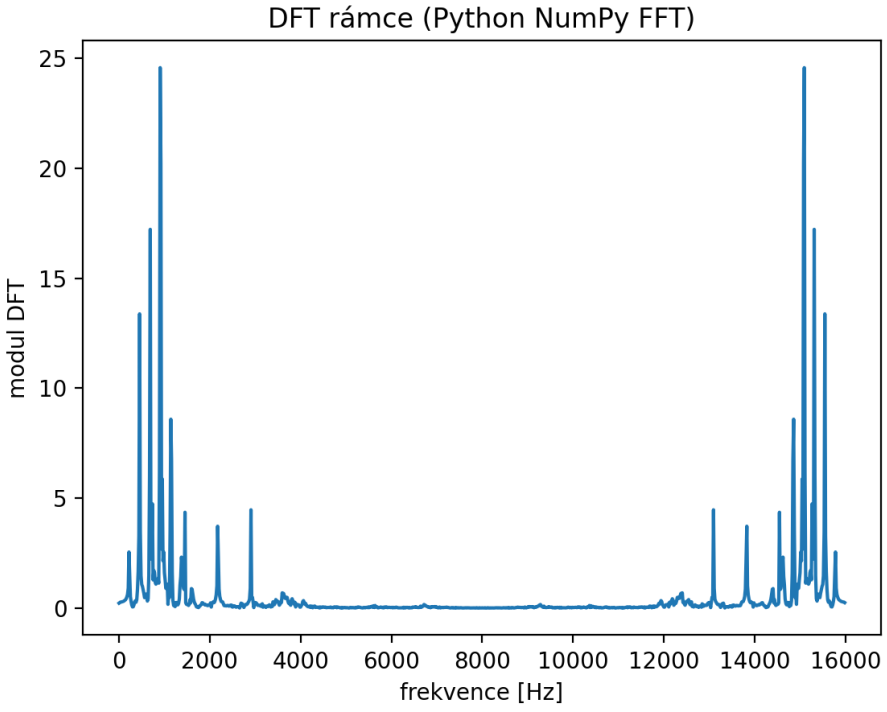
Pro vypracování této části jsem musel zjistit, kolik rámců budu vlastně potřebovat (105.8), tuto hodnotu zaokrouhlit nahoru. Následně vytvořit matici o 106 řádcích, kde každý řádek obsahuje 1024 vzorků, s tím, že pro všechny řádky (kromě prvního) platí, že prvních 512 hodnot v aktuálním je shodných s posledními 512 hodnotami předchozího řádku.

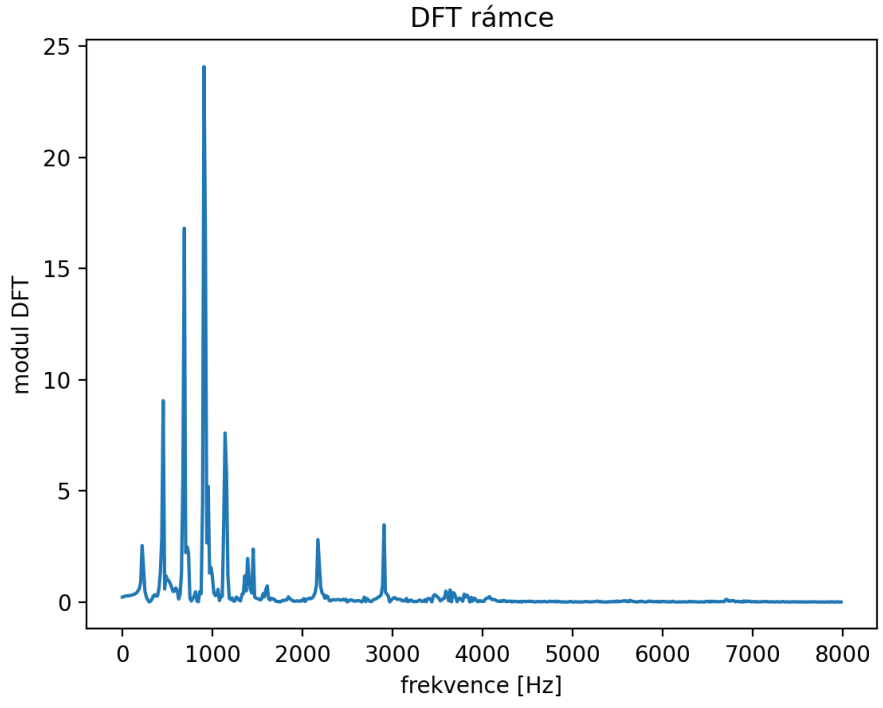
Bylo nutné ošetřit případ, kdy rámec měl zpracovávat vzorky, které byly již mimo původní signál, v takovém případě jsem nastavil hodnotu signálu v daném rámci na 0 (týká se posledních dvou rámců). Přesněji každý rámec zobrazuje vzorky z původního signálu v intervalu *[a, b]*, kde *a = i*512*, *b = a+1024*, *i = kolikátý rámec (řádek v matici) zobrazuji*. Poslední (dva) rámce však zobrazují vzorky ze signálu v intervalu, kde *a < počet vzorků signálu* a zároveň *b >= počet vzorků signálu*, tím pádem v daném rámci do určitého bodu *X* zobrazuji daný signál (dokud platí, že zobrazuji *n*-tý vzorek signálu, kde *n < počet vzorků signálu*), ale poté už ne.



Úkol 3: Diskrétní Fourierova transformace

Dle zobrazených grafů se výsledky mojí implementace neliší od té z pythonu (FFT). FFT je však několikrát rychlejší. Čas skriptu s pouze FFT je 2.1s, s mojí implementací 3.7s. Pokud pustím oboje, skript pracuje 3.8s. Mohu tedy říct, že moje implementace je o 1.6s pomalejší, než ta od Pythonu, která zřejmě zabere jen několik ms. (měřeno pomocí příkazu `time` na WSL, AMD Ryzen 5 5600H)

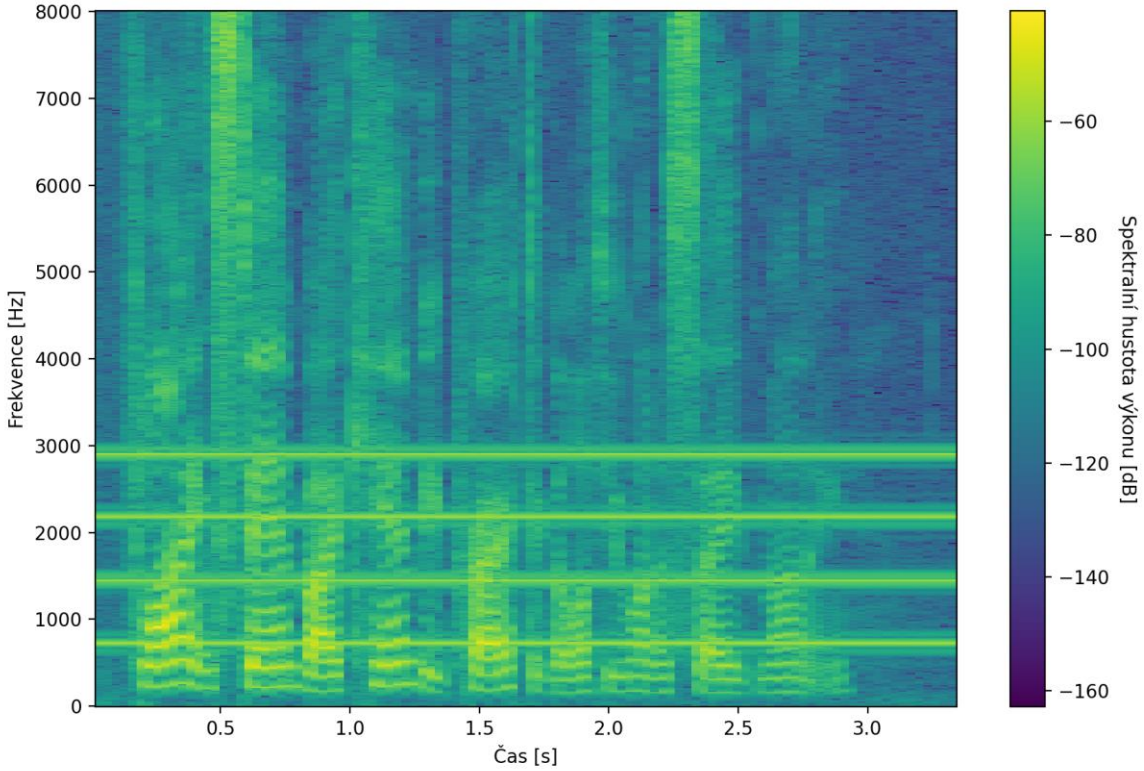




Úkol 4: Logaritmický výkonový spektrogram

Na grafu spektrogramu je jasně vidět, že kolem frekvencí [Hz] 3000, 2100, 1500 a 800 se nacházejí jakési rušivé signály.

Pro výpočet a zobrazení spektrogramu jsem se inspiroval kódem z Jupyter notebooku Ing. Kateřiny Žmolíkové a použil funkci spectrogram z modulu scipy.



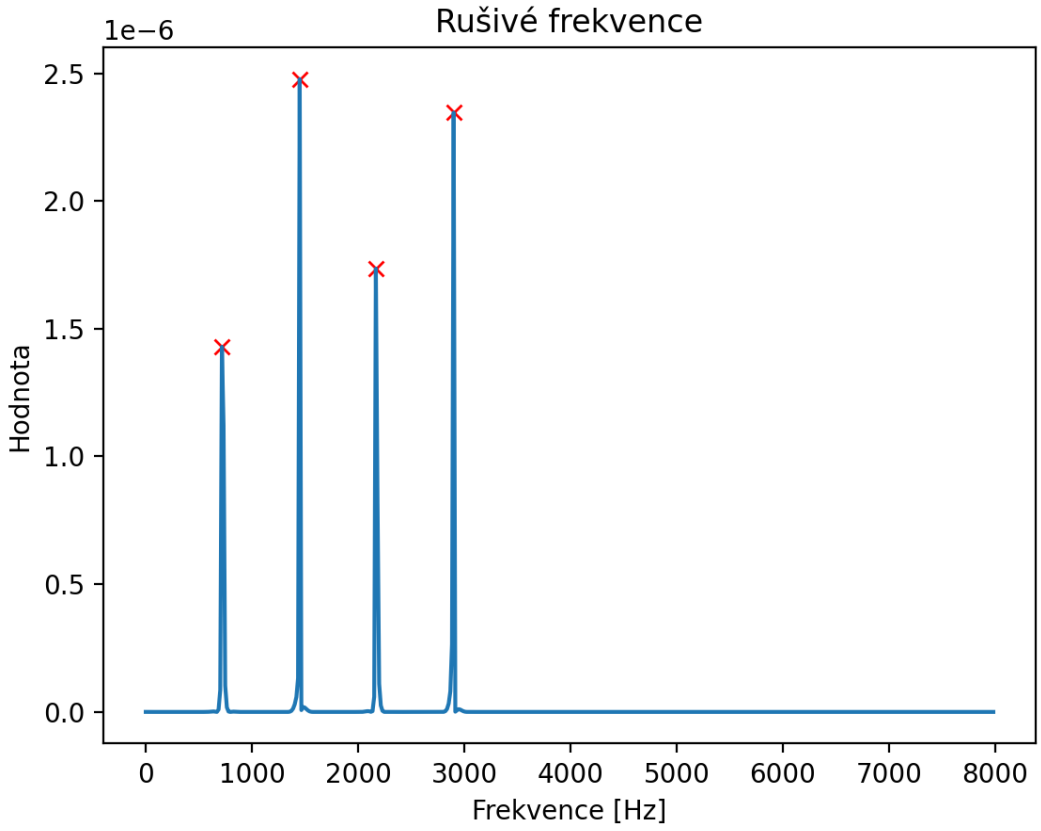
Úkol 5: Rušivé frekvence

Rušivé frekvence jsem našel pomocí funkce `scipy.signal.find_peaks`. To hledá extrémy z předaného pole, kterým zde byl (téměř) poslední sloupec z spektrogramu. Pokud se totiž podíváte na předchozí úlohu – spektrogram, tak na konci signálu je nejméně "okolního šumu" (řeči), ale rušivá frekvence je tam stále patrná, tak jako v celém signálu. Proto stačilo zvolit (téměř) poslední sloupec z spektrogramu, zde najít extrémy, a to jsou rušivé frekvence, tj.:

- 744.19 Hz
- 1426.36 Hz
- 2170.42 Hz
- 2852.71 Hz

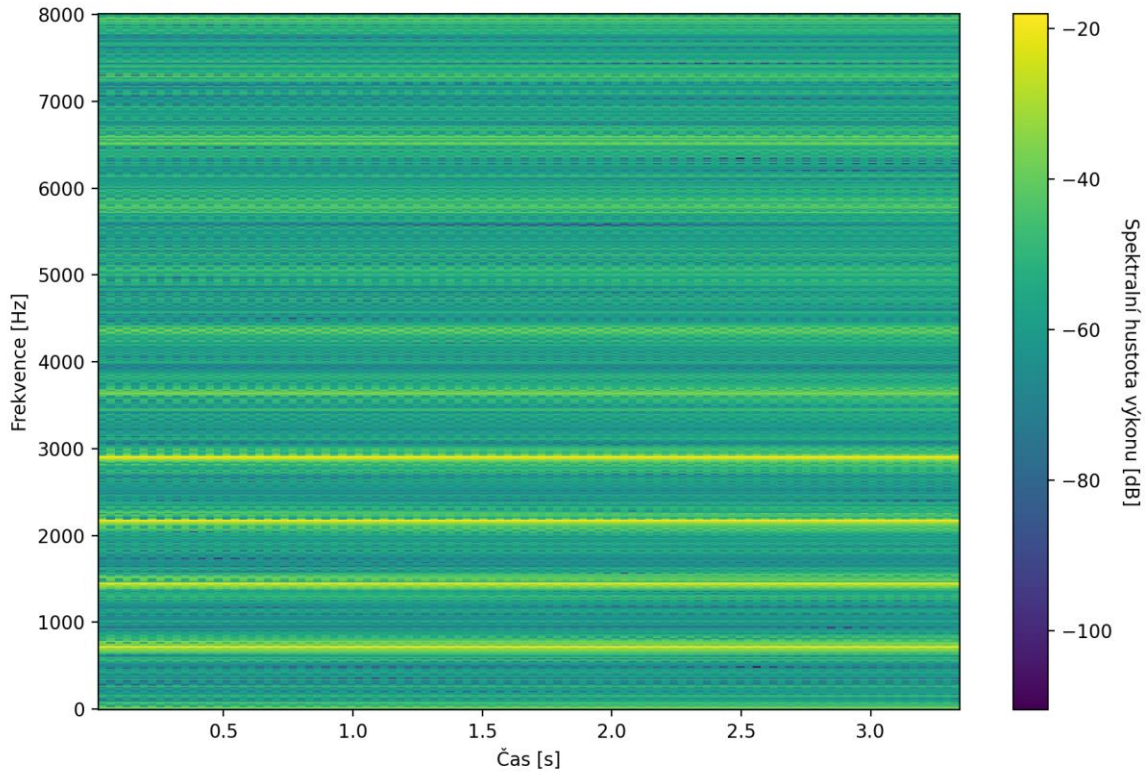
Pokud vezmeme první frekvenci a vynásobíme ji 2 – 4 krát, dostáváme přibližně hodnoty dalších rušívých frekvencí, s mírnými odchylkami:

- 744.19 Hz
- 1488.37 Hz
- 2232.56 Hz
- 2976.74 Hz



Úkol 6: Vytvoření kosinusovek a audia z nich

Audio je umístěno ve složce `audio` pod názvem `cosines.wav`. Spektrogram tohoto nového signálu není nejčistčí, nicméně jsou zde vidět opět 4 rušivé signály.



Použité zdroje

Projekt jsem nevypracoval zcela sám, bez jakékoliv pomoci. Proto zde uvádím přehledně všechny zdroje, ze kterých jsem čerpal při implementaci, případně z čeho jsem se inspiroval.

[1]: https://newbedev.com/python-write-a-wav-file-into-numpy-float-array

[2]: https://www.fit.vutbr.cz/~izmolikova/ISS/project/