# Zápočtový program - FFT ekvalizér

Petr Martišek

## 1 Anotace

Cílem zápočtové práce bylo implementovat jednoduchý pásmový ekvalizér za použití algoritmu FFT. Program umí pracovat pouze s jednoduchými *.wav soubory bez jakékoliv komprese.

## 2 Interface

Program je spouštěn z příkazové řádky s následujícími parametry:

`ffteq` *`INPUT_FILE`* `[-o` *`OUTPUT_FILE`*`]` `[-e` *`LOWER_BOUND UPPER_BOUND QUOTIENT`*`]` `[-e ...] ...`

Argument *`INPUT_FILE`* je povinný a definuje cestu ke vstupnímu souboru. Ten musí být ve formátu RIFF WAV s PCM kódováním, tj. bez jakékoliv komprese. Soubor v jiném formátu program odmítne.

Nepovinný přepínač `-o` s parametrem *`OUTPUT_FILE`* definuje cestu k výstupnímu souboru. Není-li výstupní soubor definován, vytvoří se automaticky se jménem odvozeným od vstupního (konkrétně *`INPUT_FILE`*`_eq.wav`).

Přepínač `-e` se třemi parametry definuje samotnou ekvalizaci. První dva parametry, *`LOWER_BOUND`* a *`UPPER_BOUND`*, udávají dolní a horní hranici frekvenčního pásma v Hertzích, na které se má ekvalizace aplikovat. Třetí parametr *`QUOTIENT`* udává procento změny. Např. kvocient 0.5 způsobí ztlumení daného frekvenčního pásma na 50 %, kvocient 2.0 jej naopak zesílí na dvojnásobek. Ekvalizací lze aplikovat libovolné množství opakovaným použitím argumentu `-e`. Je-li na nějakou oblast spektra aplikováno postupně více ekvalizací, efekty se sčítají.

Přepínač `-h` zobrazí nápovědu a ukončí program (nehledě na případné ostatní argumenty).

Příklad korektního volání programu:

```
ffteq input.wav -o output.wav -e 20 500 0.3 -e 14000 22000 2.3
```

Tato konfigurace způsobí ztlumení frekvencí v pásmu 20 až 500 Hz na 30 % a současně zesílení frekvencí v pásmu 14 kHz až 22 kHz na 230 %.

Dlužno podotknout, že rozsah frekvencí, na něž lze ekvalizaci aplikovat, je shora omezen $\frac{F_S}{2}$, kde $F_S$ je vzorovací frekvence vstupního souboru (typicky 44,1 kHz). Toto plyne z Shannon-Nyquistova teorému a tato hraniční frekvence se nazývá Nyquistova. Vyšší frekvence na vstupu budou automaticky seříznuty na tuto hodnotu.

## 3 Program

### 3.1 Parsování vstupu

Snad netřeba podrobnějšího vysvětlování. Parsování argumentů, uložení ekvalizačních parametrů do vektoru trojic `eqParam`, který později předáme modifikační funkci.

### 3.2 Přečtení a naparsování vstupního souboru

V další části otevřeme vstupní soubor a celý jej překopírujeme po bytech do bufferu – což je pole s prvky typu `char`. Postupně přečteme hlavičku a informace si ukládáme do vlastních proměnných. *(Pro vysvětlení struktury *.wav souboru viz. Příloha.)* Následně přečteme samotná data a uložíme je do vektorů `channels` odpovídajících jednotlivým kanálům. Počet těchto kanálů je odvozen od hodnoty `NumChannels`. Velikost vzorků, které „ukrajujeme“ z bufferu, odpovídá hodnotě `BitsPerSample`. Technicky je toho dosaženo použitím funkce `reinterpret_cast`, kdy přetypujeme ukazatel do bufferu z `char*` na `signed short*` nebo `signed long*` – podle počtu bytů, které chceme uříznout – a následně cíl tohoto ukazatele překopírujeme do vlastního vektoru.

Vzorky jsou v souboru setříděny podle času, nikoli podle kanálu. To znamená, že vzorky pro všechny kanály pro nějaký daný čas jsou u sebe, v jednom bloku.

Program je schopen zpracovat soubory vzorkované až do hloubky 32 bitů, což je pro běžné potřeby dostačující (CD kvalita je 16 bitů). Vyšší hodnoty se používají pouze např. při profesionálním zpracování audia v hudebních studiích.

### 3.3 Fourierova transformace

Samotný algoritmus transformace je implementován ve třídě `Fft`. Jedná se o nerekurzivní in-place algoritmus o složitosti $\Theta(n \log n)$. Každý kanál budeme zpracovávat zvlášť, vytvoříme tedy instatnci třídy `Fft` pro každý z nich. Konstruktoru předáme jako parametr nejnižší mocninu dvojky vyšší než počet vzorků. Při vzniku instance třídy dojde k dvěma výpočtům. Prvním je předpočítání mapy pro bitově reverzní setřídění vzorků, což je nezbytný krok pro implementaci efektivní in-place varianty. Toho je dosaženo vhodným přepínáním bitů pomocí bitové masky. Druhým krokem je výpočet mocnin primitivní n-té odmocniny z jedné, které jsou třeba v chodu algoritmu. Předpočteme hodnoty pro všechny úrovně algoritmu. Pro inverzní transformaci s výhodou použijeme tytéž hodnoty, jen komplexně sdružené.

Pomocí metody `CopyIn` překopírujeme do instance třídy vektor vzorků získaných z *.wav souboru, přičemž je však do in-place bufferu ukládáme již v reverzně-bitově setříděném pořadí, a to pomocí předpočítané mapy.

Následně v každém kanálu proběhne samotná FFT, čímž se prvky in-place bufferu změní na reprezentaci frekvenčního spektra daného kanálu. Implementace klíčového jádra algoritmu je díky předvýpočtům již poměrně jednoduchá. Při pohledu na grafické znázornění průběhu výpočtu FFT pomocí hradlové sítě (tzv. *butterfly* výpočet) by měl být význam jednotlivých kroků a proměnných v kódu zřejmý.

### 3.4 Ekvalizace

Ekvalizace je realizována metodou `Equalize`, které jsou předány jako parametry vzorkovací frekvence a ekvalizační předpisy ve formátu vektoru trojic *(dolní_frekvence, horní_frekvence, kvocient)*.

Pro správnou aplikaci ekvalizace je nejprve třeba správně interpretovat výstup FFT. Pro vstupní vektor $n$ reálných čísel je výstupem vektor $n$ komplexních čísel, kde reálná část reprezentuje koeficienty funkce *cos* odpovídající frekvence a imaginární reprezentuje funkci *sin*. Výpočet frekvence, která je komplexním číslem na daném indexu reprezentována, je následující:

$$frequency = \frac{index \times SampleRate}{n}$$

Např. pro vzorkovací frekvenci 44,1 kHz a počet vzorků 1024 je interpretace následující:

$$\begin{aligned}
&0.\ 0 \times 44100/1024 = 0,0 \text{ Hz} \\
&1.\ 1 \times 44100/1024 = 43,1 \text{ Hz} \\
&2.\ 2 \times 44100/1024 = 86,1 \text{ Hz} \\
&\quad \dots \\
&512.\ 512 \times 44100/1024 = 22050,0 \text{ Hz}
\end{aligned}$$

Pro reálný vstup je druhá polovina výsledku FFT jen komplexně sdruženým obrazem první poloviny a tedy neobsahuje informace o žádných dalších frekvencích. Více viz [1] (kapitola 6). Nejvyšší detekovaná frekvence se nachází na indexu $n/2$ a jedná se o již zmíněnou Nyquistovu frekvenci.

V praxi to znamená dvě věci: Jednak musíme pohlídat, že parametry ekvalizace zadané uživatelem jsou v rozumných mezích (tedy větší než nula a menší nebo rovny Nyquistově frekvenci) a jednak musíme vynulovat hodnoty v druhé polovině výsledku FFT, což současně vykompenzujeme zdvojnásobením intenzity hodnot v první polovině. Teoreticky bychom mohli nechat spektrum nezměněné a výsledek bude stejný, ale v praxi to může vést ke vzniku šumu a různých artefaktů. Takto těmto nepříjemnostem zabráníme a současně nedojde ke ztrátě informace.

### 3.5 Inverzní transformace

Po aplikaci ekvalizace je třeba provést inverzní transformaci. Její implementace se od dopředné liší ve třech detailech. Zaprvé je třeba in-place buffer, který nyní obsahuje frekvenční spektrum, opět bitově reverzně

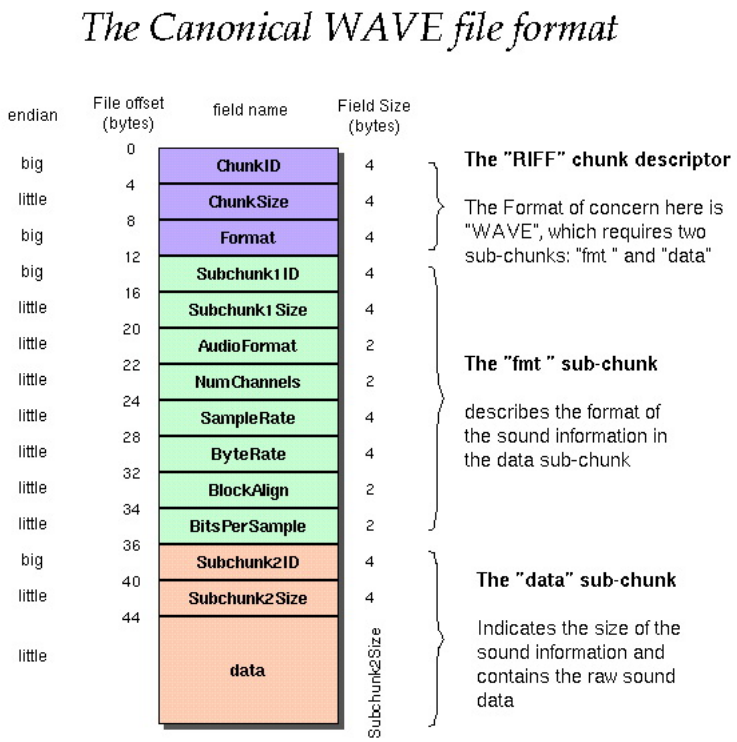
setřídit. To nutně vede k překopírování dat do dalšího vektoru a abychom si ušetřili další kopírování, provedeme samotnou transformaci v tomto novém bufferu. Druhým rozdílem je použití komplexně sdružených obrazů předpočtených exponenciál. Posledním detailem je normalizace výsledku, tj. vydělení všech prvků ve výsledném vektoru velikostí tohoto vektoru. Přitom si rovnou výsledek překopírujeme zpět do původního in-place bufferu.

### 3.6 Výstup

Pro každý kanál získáme výsledek z instance třídy `Fft` pomocí metody `GetResult`. Což je vektor komplexních čísel. Ten přetypujeme na správný typ (podle proměnné `BitsPerSample`) a po překopírování hlavičky jej zapíšeme do výstupního souboru. Opět postupujeme podle času, tedy zapíšeme hodnotu pro daný index pro všechny kanály a až pak se přesuneme na další index. Při zapisování vzorků vetších než 1 byte musíme vzít v úvahu i kódování, konkrétně je třeba převést vzorky z Big Endian na Little Endian. Pomocí bitové masky a bitového posunu to však není žádný větší problém.

## Příloha: Formát *.wav souboru

Pro práci s *.wav soubory je samozřejmě nutné znát jejich strukturu. Ta je ukázána na následující grafice[2]:




| | |
|---|---|
| ChunkID | Obsahuje slovo „RIFF“ v ASCII kódu. |
| ChunkSize | Velikost zbytku souboru po tomto poli, tj. faktická velikost souboru mínus 8. |
| Format | Obsahuje slovo „WAVE“ v ASCII kódu. |
| Subchunk1ID | Obsahuje slovo „fmt “ v ASCII kódu. |
| Subchunk1Size | 16 pro PCM. Velikost zbytku sekce „fmt“. |
| AudioFormat | 1 pro PCM. Jiná hodnota znamená nějakou kompresi. |
| **NumChannels** | **Mono = 1, Stereo = 2, atd.** |
| **SampleRate** | **8000, 44100, atd**. |
| ByteRate | $SampleRate \times NumChannels \times BitsPerSample/8$ |
| BlockAlign | $NumChannels \times BitsPerSample/8$ |
| **BitsPerSample** | **8 bits = 8, 16 bits = 16, atd.** |
| Subchunk2ID | Obsahuje slovo „data“ v ASCII kódu. |
| Subchunk2Size | $NumSamples \times NumChannels \times BitsPerSample/8$. Velikost zbytku této sekce (tj. fakticky velikost vlastních zvukových dat). |
| Data | Samotná data. |

## Reference

[1] Kevin J. McGee, *An Introduction to Signal Processing and Fast Fourier Transform (FFT) (excerpt)*, 2009, `http://fftguru.com/fftguru.com.tutorial.pdf` Vynikající materiál o Fourierově transformaci, cenným zdrojem byla zejména pasáž o interpretaci výsledků a záporných frekvencích.

[2] `https://ccrma.stanford.edu/courses/422/projects/WaveFormat/` Přehledný rozbor struktury standardního *.wav souboru.