Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Detekce a klasifikace základní frekvence zvukového záznamu za účelem nalezení hraného tónu

Plzeň, 2014

Zdeněk Janeček

Děkuji své rodině za obrovskou podporu a také Marku Špelinovi za cenné připomínky.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 9. května 2014

Zdeněk Janeček

Abstract

This thesis presents time-domain methods for pitch period detection. In the first part, the basic sound theory is described, including tone frequencies, from the point of view of physical and human perception. The second part is dedicated to the methods, such as the autocorrelation, SDF, their combination called SNAC, and AMDF.

The last part mentions actual implementation for Android devices and shows the results of the described methods and my own application architecture. The final application is called Muassist.

Abstrakt

Tato práce představuje metody v časovém pásmu pro hledání znějícího tónu. V první části popisuje obecnou teorii vzniku a zaznamenávání tónů z fyzikálního i počítačového pohledu. Mezi popisované metody patří autokorelace, SDF, jejich kombinace SNAC a AMDF.

Další část se věnuje reálné implementaci pro zařízení s operačním systémem Android a ukazuje výsledky popisovaných metod i s jejich chybami a výhodami. Výsledná aplikace Muassist je určena všem hudebníkům jako užitečný pomocník. Poslední část se věnuje programovému návrhu ladičky.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Lidé se vyjadřují hudbou už od pradávna. Najdeme ji v každé kultuře. Sloužila k obřadům, oslavám a vyprávěla své příběhy lidí od nejchudších až po nejvyšší vrstvy.

Hudba prochází evolucí stejně jako lidstvo a je jejím obrazem. Naučili jsme se rozeznít kus dřeva, ale i píšťalu z plechu. Umíme dokonce dělat zvuky pomocí počítačových programů a napodobovat ty reálné.

Sluchové vjemy také prochází svým vývojem stejně jako celý mozek. Učíme se jak tóny mají znít a naše neuronové spoje se přizpůsobují okolnímu prostředí. Dlouhodobé využívání jedné části mozku může upravit mozkové funkce natrvalo. Tento jev lze pozorovat v mnoha oblastech lidského života.

Schopnost rozlišování tónů u muzikantů a nemuzikantů zjišťovala studie (Tervaniemi et al., 2004). Obecně se považuje, že profesionální houslista je schopen rozpoznat malé změny tónu, které ostatní nepoznají. Vybrali tedy 13 muzikantů a 13 nemuzikantů. Metoda zkoumala vnímání sekvencí mnoha čistých a minima falešných tónů při soustředění a začtení do knihy. Muzikanti rozpoznávali přesněji a rychleji, nicméně nemuzikanti byli také schopni rozpoznat i malé změny. Tento výzkum ukazuje, že muzikanti nejsou nejlepší v rozpoznání falešného tónu za každé situace.

U všech skupin bylo zjištěno, že čím větší byla odchylka, tím rychlejší byla doba reakce. Rozpoznávání falešného akordu bylo lepší za použití stupnice při temperovaném ladění než u exotického ladění. To nahrává teorii, že jsme se jej naučili rozpoznávat, a to muzikanti i nemuzikanti.

Každý člověk, který slyší rozladěný tón většinou ví, že je něco špatně, ale neví, jak je tón posunutý. Historicky byla ladička kovový předmět vydávající tón pevné výšky a naopak chromatické ladičky umí najít jakýkoli jiný tón. Čím déle slyšíme samostatný lehce rozladěný tón, tím nám přijde lepší. Lidský mozek se na rozdíl od ladičky pořád učí a přizpůsobuje.

1.1 Zvuk

Zvuk je mechanické vlnění. Prostředím kde k vlnění dochází, může být vzduch, voda, ale i pevná látka. Toto vlnění se přenáší z jednoho prostředí do druhého. Vnímatelné frekvence se liší u každého jedince a pohybují se v rozsahu 16 Hz až 20000 Hz.

Zdrojem zvuku může být každé kmitající těleso. O výsledném vlnění rozhoduje nejen zdroj samotný, ale všechna prostředí přes která prochází. Takovým příkladem jsou samohlásky, které odlišujeme nikoli hlasivkami, ale rozevřením či uzavřením úst. Jejich rozpoznávání pak spočívá v hledání *formantů*, které vykazují ve frekvenčním spektru nejvyšší amplitudu[1]. Spektrum zvuku bývá mimo požadovaného tónu také bohaté na bílý šum[2].

Hudební teorie dala pojmenování mnoha jevům, V dodatku A.1 jsem uvedl pár termínů, které souvisí s touto tématikou.

Získávání zvuku stejně jako jeho tvorba vychází z pohybu. Rozpohybováním vnitřních částí ucha budíme citlivé buňky, které vytváří vzruchy do mozku. Na podobném principu pracuje také mikrofon. Magnetická indukce na cívce vytváří elektrický proud a tím získáváme analogovou reprezentaci zvuku.

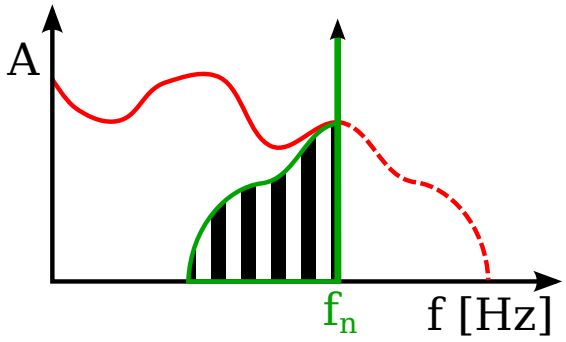
Při počítačovém zpracování pracuji s konečnou posloupností vzorků. Analogový signál převádím na digitální konečný signál *vzorkováním*. Vzorkovací frekvence (sampling rate) f_r udává, jak často zaznamenávám aktuální výchylku analogového signálu. Převod mezi digitálním a analogovým signálem je bezpečný, pokud je dodržen Nyquistův vzorkovací teorém:

> Přesná rekonstrukce spojitého, frekvenčně omezeného, signálu z jeho vzorků je možná tehdy, pokud byla vzorkovací frekvence vyšší než dvojnásobek nejvyšší harmonické složky vzorkovaného signálu.

Na obrázku 1.1 je vidět zlom ve frekvenčním pásmu. Všechny frekvence vyšší než nyquistovo frekvence f_n se projeví v nižším pásmu jako šum. Frekvence f_n je tedy polovina vzorkovací frekvence. Při dodržení tohoto kritéria, lze bez ztráty převádět mezi digitálním a analogovým signálem. Analogový signál je pro nás přirozený, ale jeho zpracování a uchovávání přináší potíže ve formě šumu, ceny a případných fyzických úprav zařízení. Výsledný signál se vždy převede na analogový, ale až při jeho reprodukci a i tady záleží na kvalitě převodníku. Digitální filtr lze napsat třeba pro mobilní telefon a aktualizovat třeba z internetu. (Montgomery, 2010)

[1] více na http://is.muni.cz/elportal/estud/ff/js07/fonetika/materialy/ch06s02.html

[2] Bílý šum je nazýván jako analogie s bílým světlem, které obsahuje všechny frekvence.



Obrázek 1.1: Nyquistova frekvence a vzniklý šum

1.2 Ladění

Tóny jsou tvořeny základní frekvencí f_0 a jejími násobky, které nazýváme harmonické. Dvojnásobná frekvence je tzv. *oktáva*, která je důležitá v dalším povídání.

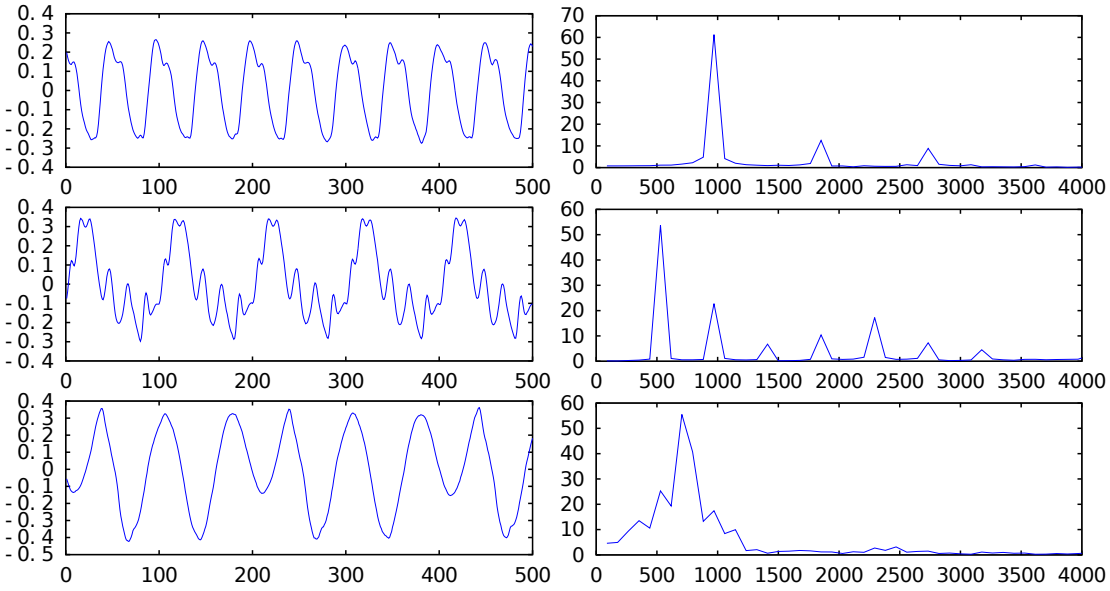
Ladění každého hudebního nástroje znamená nastavení znějící struny, plátku nebo objemu vzduchového sloupce v rámci dovolených mezí. Neexistuje nástroj, který by byl „z fabriky naladěn“, protože je vystavován změnám teploty, tlaku a vlhkosti.

Strunné nástroje se ladí napínáním struny. Čím je struna napnutější, tím je výška tónu vyšší. Protože kovy snížením teploty povolují, jde tak výška tónu také dolů. Celý nástroj je ze dřeva, tudíž mění objem jinak než kov. Housle mají dva nezávislé ladiče a to velké dřevěné kolíky a přesné doladovače. Hráč na strunný nástroj tvoří další tóny stisknutím struny na tzv. hmatníku. Samotné kmitání struny se vytváří pomocí smyčce nebo drnkáním. Typickým strunným nástrojem jsou například housle, nebo kytara. Piano se sice nepovažuje za strunný nástroj, ale tón také vzniká na napnutých strunách.

Dechové nástroje jsou buď dřevěné, nebo kovové. Výška tónu je dána objemem celého nástroje. Nejvýše zní pikola a jedním z nejnižších je tuba. Při změně teploty se chovají stejně jako strunné a jejich ochlazení způsobí snížení tónu. Další tóny vznikají uzavíráním otvorů a to zvětšuje celkový objem. Při hře se nástroj také ohřívá dechem, proto je třeba stále hlídat výšku tónu. Některé dechové nástroje tvoří zvuk rozkmitáním plátku a některé nárazem vzduchu na hranu. Plátkové nástroje tvoří zvuk rozkmitáním plátku, který přerušuje proud vzduchu.

Na obrázku 1.2 jsou ukázány krátké vzorky zvuků flétny, houslí, hlasu a jejich spektra. Všechny zvukové vzorky byly nahrány mobilním telefonem v normální místnosti bez komprese. Můžete si všimnout že nástroj vytváří spektrum harmonické řady. Hlas je obecně mnohem složitější a není předmětem této práce.

Na flétně zněl přibližně tón A^2 který odpovídá $F_0 = 880\,Hz$. Housle zněly tónem A^1 tj. $F_0 = 440\,Hz$.



Obrázek 1.2: Porovnání spekter flétny, houslí a hlasu. Vlevo okna o délce 500 vzorků a vpravo příslušná spektra v závislosti na frekvenci.

Výsledné spektrum zřetelně ukazuje významné frekvence. Transformace signálu do frekvenčního pásma je popsána v kapitole 2.6. Frekvenční rozlišení spektra je závislé na délce okna. Při vzorkovací frekvenci 44100 a délce 500 vzorků to odpovídá 88, 2 Hz. Při ladění je třeba přesnosti alespoň na 1 Hz. Velikost okna způsobila, že vrchol nevyšel 440 Hz, ale spíše 520 Hz.

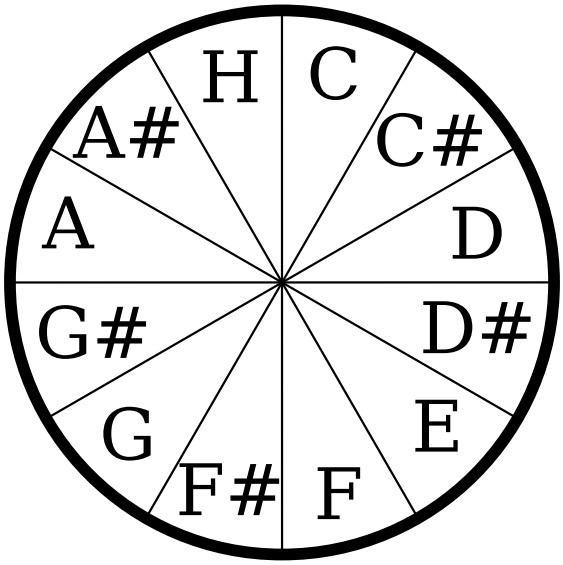
Velikost intervalu n se měří v centech. Vychází z poměru frekvencí f a libovolného vztažného tónu f_0 (často komorní A[1]).

$$\log_2 \frac{f}{f_0} = \frac{n}{1200} \Rightarrow n = 1200 \cdot \log_2 \frac{f}{f_0} \text{ [cent]} \tag{1.1}$$

Dvojkový logaritmus často neumíme spočítat přímo, ale počítá se přes dekadický:

$$\log_2 x = \frac{\log x}{\log 2} \tag{1.2}$$

Jaké tóny jsou ale ty správné? Existuje mnoho způsobů, jak definovat množinu tónů. Za celou historii vzniklo bezpočet systémů. Obecně v hudbě jde jen málo o fyziku a matematiku, ale o estetické cítění. Slyšíme-li nějakou melodii vnímáme především intervaly mezi tóny a je úplně jedno v jaké výšce. Můžeme jí tedy libovolně posunout nahoru a dolů a pořád to bude stejná melodie. Problém nastává má-li spolu hrát více nástrojů. Hudba totiž není o jednom hlasu a běžně slyšíme dva a více hlasů, které jsou od sebe vzájemně posunuty. Posun může být oktáva, ale



Obrázek 1.3: Tónový kruh

může být teoreticky libovolný. Notový zápis by byl neúplný bez tónového systému. Pro přesné určení výšky tónu je potřeba znát jeho označení a v jakém tónovém systému se nachází. Více o tématu pojednává učebnice Zenkl (1982).

Nejstarší stupnice je *pentatonika*, kde je pouhých 5 tónů. Stupnice vznikla z tónů vzdálených o kvintu (7 půltónů). Pohledem na obrázek 1.3 snadno dopočítáme tóny ve stupnici. Začíná se od tónu C a vyjde posloupnost C-G-D-A-E. Tento rozsah nemohl stačit, a proto se stupnice rozšířila na 12 půltónů, tak jak je uvedeno na obrázku.

Nejmenší společný násobek 7 (kvinty) a 12 (oktávy) půltónů je 84. Poskládáním 12 kvint a 7 oktáv za sebe by mělo dát stejný tón, protože odpovídají stejnému počtu půltónů. Ve skutečnosti tomu tak není, a zavadí se tzv. Pythagorejské Koma které odpovídá:

$$\frac{(3:2)^{12}}{(2:1)^{7}} = \frac{3^{12}}{2^{12+7}} \approx 1,01364 \approx 23,46\,[\text{cent}]$$

Tato chyba odpovídá téměř čtvrt půltónu (využitím rovnice 1.1). Poměr frekvencí v čistém ladění musí být možné složit z celých čísel, takže kvinta odpovídá poměru 3:2 a oktáva vždy 2:1.

V době Renesance a Baroka vzniklo kolem 40 různých druhů ladění. Každé bylo zkonstruováno s nějakým cílem jako dosáhnout určitých čistých intervalů, to například přinášelo falešné souzvuky při modulaci do jiné tóniny. Proto se všeobecně přijalo *rovnoměrně temperované dvanáctistupňové ladění* které navrhl v r. 1691 Andreas Werckmeister. Na tónech lze pak provádět tzv. *enharmonickou záměnu*,

#	Čisté	Pythagorejské	Werckmeister I	Temperované
1	16/15 → 111,73	90,22	90	100
2	9/8 → 203,91	203,91	192	200
3	6/5 → 315,64	294,13	294	300
4	5/4 → 386,31	407,82	390	400
5	4/3 → 498,04	498,08	498	500
6	7/5 → 582,51	$7 \downarrow 588,27$ $5 \uparrow 611,73$	588	600
7	3/2 → 701,96	701,96	696	700
8	8/5 → 813,69	792,18	792	800
9	5/3 → 884,36	905,87	888	900
10	16/9 → 996,09	996,09	996	1000
11	15/8 → 1088,27	1109,78	1092	1100

Tabulka 1.1: Nejznámější druhy ladění v centech

to znamená že snížení vyššího je stejné jako zvýšení nižšího tónu. Toto ladění řeší také Pythagorejskou Komu. Máme-li daný základní tón f_0 pak jakýkoli n-tý tón spočítáme:

$$f_0 \cdot \left(\sqrt[12]{2} \right)^n = f \,[\mathrm{Hz}] \tag{1.3}$$

Nárůst frekvencí je tedy geometrická řada s kvocientem $\sqrt[12]{2}$. Máme-li tedy rozumně zobrazit tóny v závislosti na frekvenci, volíme logaritmické měřítko. Notová osnova je takovým příkladem.

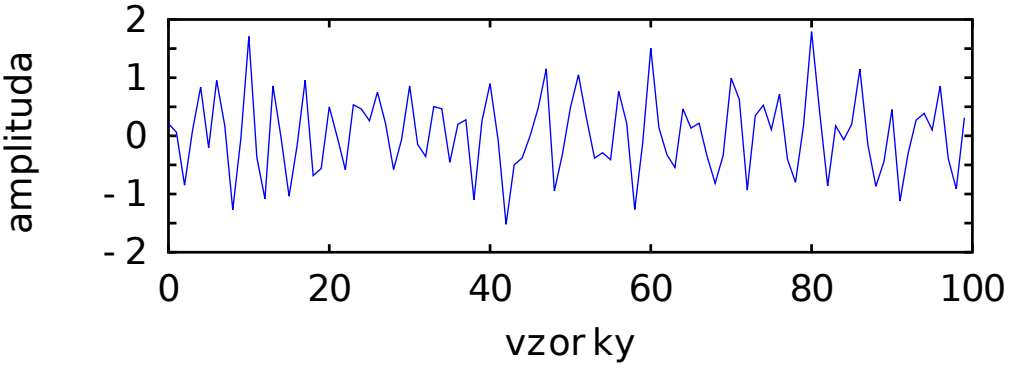
Půltón byl definován jako vzdálenost 100 centů. Lidské ucho je schopno rozlišit přibližně rozdíl 8 centů (McLeod, 2008)(Jacob et al., 2008)[3]. Záleží na frekvenci, hlasitosti a délce tónu. Jakmile zní více tónů najednou, je tento rozdíl mnohem znatelnější. Výsledná vlna složená z kmitů blízkých frekvencí totiž není harmonická a vznikají rázy. Vymizení rázů je tedy indikátorem shody dvou frekvencí (Rusňák, 2008).

Jako důkaz stačí když máme dvě vlny s úhlovou rychlostí ω_i a platí:

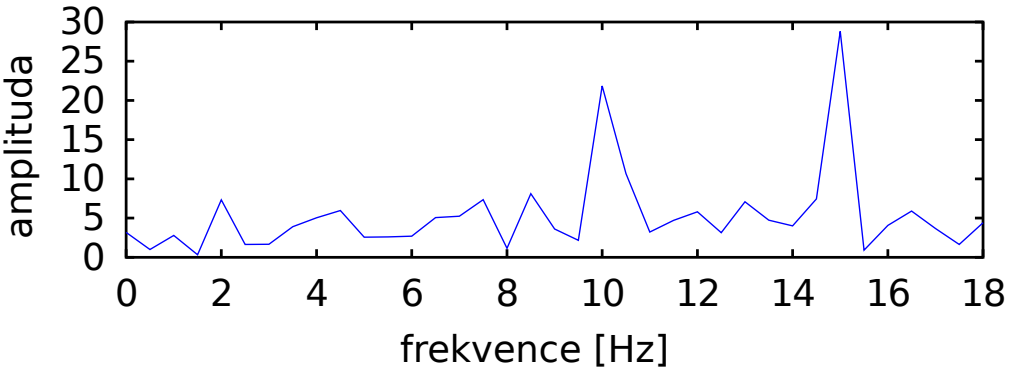
$$\frac{\omega_1 - \omega_2}{2} = \omega_0 \to 0 \qquad T_0 = \frac{1}{f_0} = \frac{2\pi}{\omega_0} \to \infty$$

Jen pro informaci uvádím různé druhy ladění v tabulce 1.1.

[3]při nižších frekvencích kolem 500 Hz je to 8 centů a nad 1000 Hz je to 10 centů.



Obrázek 1.4: Ukázkový signál vzorkovaný 50 Hz o délce 2 s.



Obrázek 1.5: Spektrum signálu na obrázku 1.4

1.3 Dominantní a základní frekvence

Zvuk nástroje může nést mnoho zvuků, které vytváří barvu tónu. Významných frekvencí nebývá mnoho na rozdíl od četného šumu pozadí. Nejvyšší vrchol spektra odpovídá dominantní frekvenci.

Základní frekvence F_0 je definována jako nejnižší tón harmonické řady. Typické harmonické řady nástrojů jsem uvedl na obrázku 1.2, kde nejsilnější frekvence byla současně dominantní a základní. Obecně se základní od dominantní liší. Na obrázku 1.4 je příklad zašuměného zvuku, který je složen z 10 Hz, 15 Hz a bílého šumu. Základní frekvence je v tomto případě 5 Hz, protože rozdíl frekvencí je $15 - 10 = 5$, ale v signálu se nenachází.

Tento tón se nazývá virtuální tón. Náš mozek je schopen ho rozpoznat. Každý harmonický tón vzniká násobkem F_0, takže vzdálenosti mezi harmoniemi ve frekvenčním pásmu jsou stále stejné.

Pokud není zvuk harmonický, nemá význam hledat v něm hraný tón. Příkladem zvuku, který není harmonický jsou bubny, činely nebo ruchy.

Kapitola 2

Dostupné metody

Dosud popisovaná reprezentace zvuku je takzvaně *časově-amplitudová* a odpovídá reálnému světu. Ze zvukové vlny jako byla na obrázku 1.4 nelze jen pohledem zjistit jaké složky obsahuje. Proto jsem uváděl takzvané spektrum (vzniklé Fourierovo transformací), které zobrazuje signál ve *frekvenčně-amplitudové* reprezentaci. Toto zobrazení je nevhodné na časově proměnné signály, protože nevíme kdy konkrétní frekvence začíná a končí. Proto se zavádí *časově-frekvenční* reprezentace, která nabývá prostoru o jednu dimenzi větší a je tedy vhodná na nestacionární signály. Nejznámější je Waveletová transformace, nebo STFT (Short Time Fourier Transform). (Polikar, 1996)

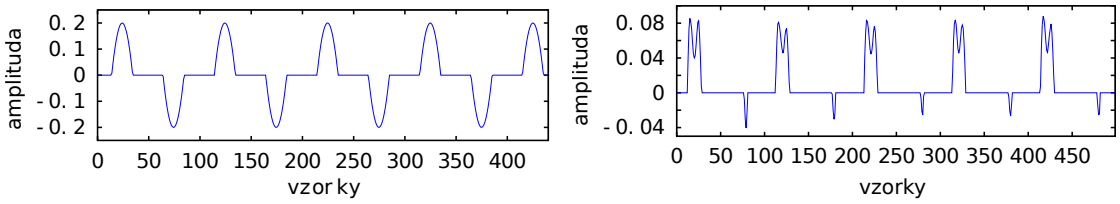
V této práci je vzhledem k vlastnostem zkoumaného signálu vhodné analyzování časově amplitudové, protože se zvuk při ladění typicky nemění a není tak výpočetně náročné. Protože se stala Fourierova Transformace standardem signální analýzy, uvádím na konci této kapitoly krátké seznámení.

2.1 Nejjednodušší přístup

Základní postup vychází z časové periodicity. Pohledem na zvukové vzorky na obrázku 1.2 strany 4 nás rychle napadne najít maxima a pomocí vztahu 2.1 spočítat frekvenci. Vzdálenost maxim udává „čas“ T ve vzorcích a f_r je vzorkovací frekvence.

$$f = \frac{f_r}{T} \tag{2.1}$$

Protože signál obsahuje obvykle mnoho šumu odstraňuje se neurčitost kolem nuly prahováním. Velikost prahu lze volit v závislosti na rozsahu signálu $\Theta = \alpha(V_{max}+V_{min})$. Osvědčil se ořezávající faktor $\alpha = 0,4$, V_{max} najde nejvyšší hodnotu amplitudy v okně a obdobně V_{min} minimum.



Obrázek 2.1: Ořezávání nízkých amplitud. Vlevo sinusoida, vpravo housle.

Nalezené vrcholy se nalézají ve vzdálenostech harmonických frekvencí. Výslednou vzdálenost jsem hledal průměrem vzdáleností od předchozího vrcholku. To v Octave znamenalo:

```
dist = tops(2:end) - tops(1:end-1);
freq = rate / mean(dist(1 : end - 1));
```

Výsledek byl až překvapivě dobrý. Pro čistou sinusoidu jsem dosahoval chyby nejvíce 4 centy. To bych mohl říci, že stačí. Faktem je, že tato metoda nezaručuje stabilitu. S přibývajícím šumem a množstvím obsažených frekvencí se tato naivní metoda stává nevypovídající. Výsledek je na obrázku 2.1. Například u houslí už nejsou jednoznačně určené vrcholky.

2.2 Autokorelace

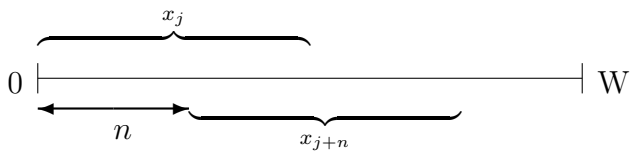
Ještě před Autokorelací uvedu definici obecné korelace dvou signálů. Jedná se o operátor ukazující podobnost signálů f a g a odpovídá skalárnímu součinu. V rovnici 2.2 uvádím diskrétní podobu.

$$(f \star g)[n] = \sum_m f(m)g(m+n) \qquad 0 \leq n < M \tag{2.2}$$

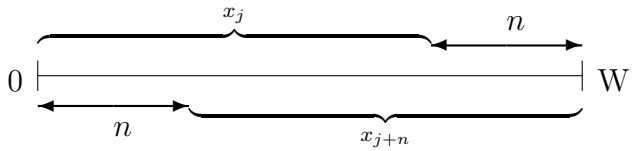
V definici není nijak určeno, jak mají být signály dlouhé ani jakých hodnot nabývají indexy n a m. Když se jejich délky liší, doplňuje se kratší signál nulami. Délka signálu je M, takže m bývá v rozsahu $0 \ldots M-1$. Posun n nabývá stejného rozsahu, ale je třeba pamatovat, že g bude potřebovat $2M-1$ vzorků. Hledáme takové n, kde vyjde největší číslo, protože tam došlo k nejsilnější shodě.

Při autokorelaci (ACF) je g a f stejná funkce, jedná se tedy o speciální případ korelace, kde výsledná délka signálu je stejná jako vstupní signál. Výstupem je opět funkce závislá na čase. Největší shoda bude samozřejmě při posunu $n = 0$. Definují se dva typy. Typ 1:

$$r(n) = \sum_{j=0}^{W/2-1} x_j x_{j+n} \qquad 0 \leq n \leq W/2 \tag{2.3}$$



Obrázek 2.2: ACF typu I



Obrázek 2.3: ACF typu II

Autokorelační funkce značím r. W je velikost okna. V tomto případě obsahuje suma stále stejný počet sčítanců a to $W/2$. Tato metoda může být použita na stacionární signály, které se nemění v čase a opakují se v okně.

Typ II značím apostrofem:

$$r'(n) = \sum_{j=0}^{W-1-n} x_j x_{j+n} \qquad 0 \leq n < W \tag{2.4}$$

Při použití této definice se mění počet sčítanců. Získaná energie $r'(n)$ klesá až k nule při $n = W$. Tento efekt běžně odstraňujeme škálováním, kterým vzniká nezkreslená forma (unbiased) 2.5, kterou označuji dolním indexem U. Musíme si ale uvědomit, že kvůli numerické stabilitě nese riziko vysokých nestabilit při $n \to W$. Proto se často bere jen polovina zpracovaného signálu.

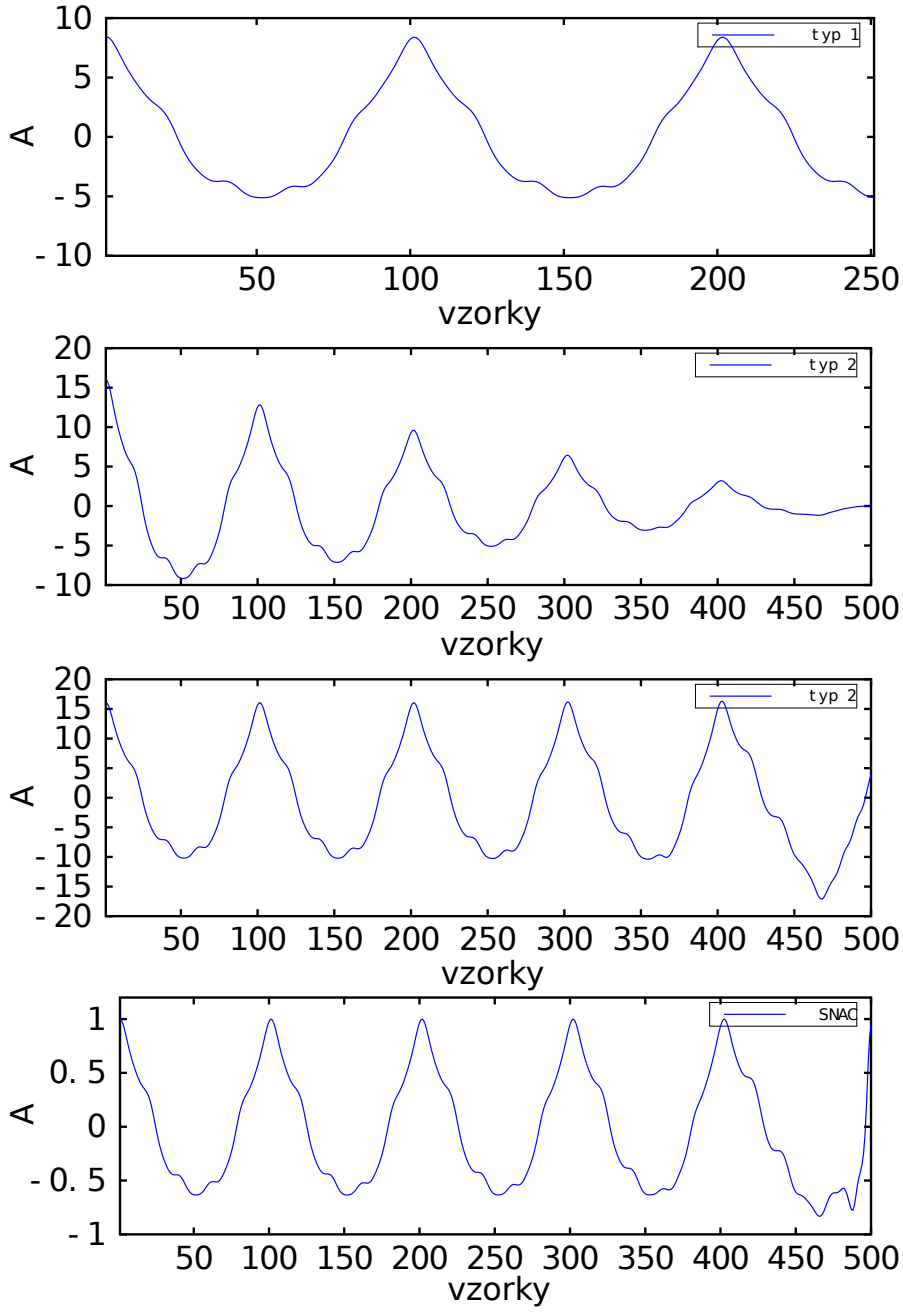
$$r'_U(n) = \frac{W}{W-n} \sum_{j=0}^{W-1-n} x_j x_{j+n} \qquad 0 \leq n < W \tag{2.5}$$

Indexování je zřejmé z obrázků 2.2 a 2.3.

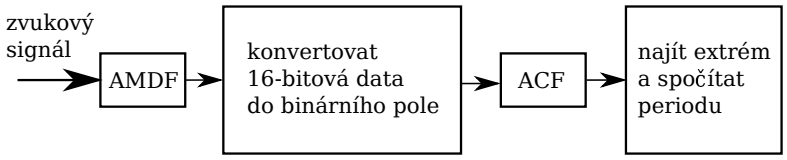
Srovnání jednotlivých ACF variant je na obrázku 2.4. Všimněte si rozkmitání na konci signálu nezkreslené formy $r'_U(n)$.

Obdobně se definuje SDF (Square Difference Function), také v I a II. formě. Vychází z toho, že periodické části jsou tvarově podobné. Pokud tedy uděláme rozdíl fázově posunutých částí signálu, shodné části budou dávat v součtu malé hodnoty.

$$d'(n) = \sum_{j=0}^{W-1-n} (x_j - x_{j+n})^2 \qquad 0 \leq n < W \tag{2.6}$$



Obrázek 2.4: Výsledek ACF typu I, II, její nezkreslené formy a SNAC signálu houslí.



Obrázek 2.5: Schéma AMDF+ACF algoritmu

Autokorelace je jednoduchá, ale dává nestabilní maxima v závislosti na počtu period v okně a obsažených signálů s rozdílnou fází. Existuje modifikace nazvaná SNAC, která dává dohromady výhody ACF a SDF. Blíže popisuje McLeod (2008) ve své práci a vypadá takto:

$$n'(\tau) = \frac{2\sum_{j=0}^{W-1-\tau} x_j x_{j+r}}{\sum_{j=0}^{W-1-\tau}(x_j^2 + x_{j+\tau}^2)} = \frac{2r'(\tau)}{m'(\tau)} \tag{2.7}$$

2.3 Rozdílová funkce

Existuje ještě jedna modifikace, která je mnohem méně výpočetně náročná než ACF a dostatečně rychlá pro zpracovávání v reálném čase. Schéma celého algoritmu je na obrázku 2.5. Nejdříve jsou spočteny hodnoty AMDF (Average magnitude Difference Function) funkce pro časové okno. Poté jsou tyto hodnoty transformovány do binární posloupnosti s použitím prahu. Tento binární signál se pomocí binární operace AND a ACF rychle přepočítá na funkci, ve které lze hledat vrcholy. Tento algoritmus jsem pojmenoval AMDF+ACF.

Pro periodický signál $s(n)$ s periodou T bude rozdílová funkce $x_{j+n} - x_j$ blízká nule pro $n = 0, \pm T, \pm 2T, \ldots$, protože je zvuk kvazi-periodický (Hui et al., 2006). Této vlastnosti využívá AMDF funkce $a'(n)$, která bude:

$$a'(n) = \frac{1}{N-n} \sum_{j=0}^{N-n-1} |x_{j+n} - x_j| \tag{2.8}$$

kde x_j je časové okno se zvukovými vzorky, N délka signálu a n nabývá hodnot od 0 do $N-1$. Ve vzniklé funkci $a'(n)$ se hledá minimum, kde vznikl nejmenší součet, tedy největší shoda částí okna. Tento přístup snadno vede k poloviční nebo násobné chybě a dostáváme harmonickou frekvenci nikoli dominantní frekvenci.

Funkce je formálně shodná s $r'_U(n)$ (ACF typu II nezkreslená). Má i některé její nevýhody jako nestability pro $n \to N$.

Získané AMDF hodnoty ořízneme podle prahu. K tomu potřebujeme znát maximální a minimální hodnotu amplitudy ze zpracovaného okna funkce $a'(n)$. Zvolíme faktor ořezávání α. Úroveň ořezávání je pak $\Theta = \alpha(V_{max} + V_{min})$. Osvědčil se

faktor α kolem $0,4$. AMDF vzorky s amplitudou menší než Θ se nastaví na 1 a naopak s větší na 0.

Pohledem na rovnici 2.4 zjistíme, že součin bude jedna pouze v případě kdy budou oba vzorky x_j a x_{j+n} rovny jedné. Díky tomu lze nahradit násobení operací AND. Do sumy se přičítá jednička nebo nula. Celý proces je rozebrán na obrázku 2.6.

Při testování této metody jsem narazil na zásadní problém, protože výsledný binární signál nebyl symetrický kolem pravého vrcholu, vznikala nepřípustná chyba. Tento problém jsem částečně odstranil průměrováním vzdáleností mezi více vrcholy, ale tím jsem jen zakrýval nedokonalost. Opustil jsem tedy od této modifikace, přestože vypadala slibně.

Zmíněné důvody mě vedli k AMDF-M (Average magnitude Difference Function modiffied) funkci, ořezávání a hledání vrcholu popsané v další kapitole. Popsaná AMDF funkce není moc vhodná na detekci tónu. Každá hodnota $a'(n)$ s parametrem n obsahuje $N-n$ rozdílových hodnot. Požadovaný parametr n dostáváme při nízkém výsledku rozdílu $|x_{j+n} - x_j|$. Jakmile $(N-n) \times T_r$ (T_r je vzorkovací perioda) je menší než perioda tónu, se funkce $a'(n)$ stává nevypovídající pro získání periodických vlastností.

Například pro mé příklady s 500 vzorky, vzorkovací frekvencí $f_r = 44100$ a tónu 440 Hz byla nevhodná již od 400. vzorku. Protože nevíme jaké frekvence jsou v okně obsaženy, je lepší vyhnout se případnému problému. AMDF-M funkce $a(n)$ s pevným počtem členů v sumě vypadá následovně:

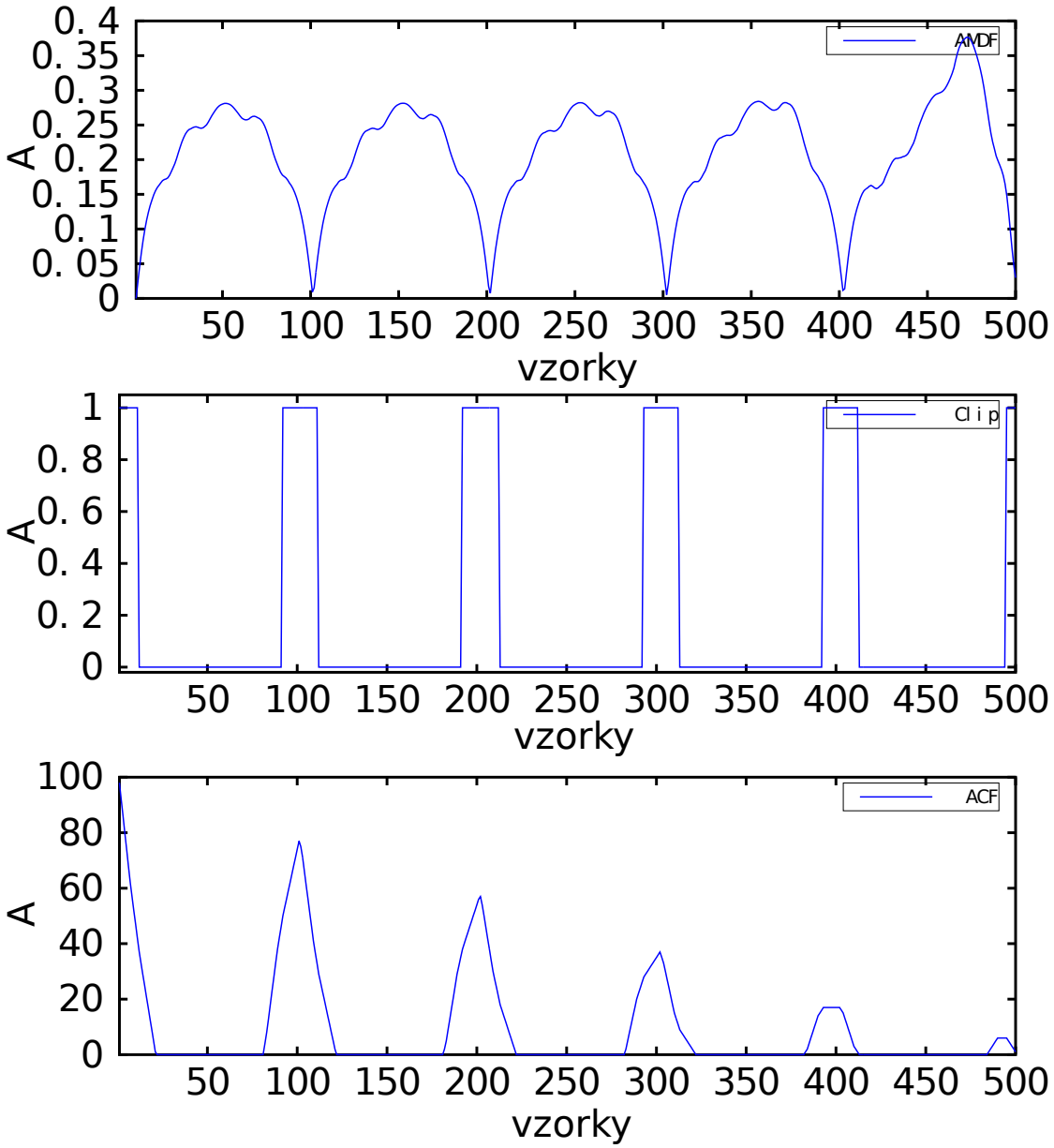
$$a(n) = \frac{1}{N-1} \sum_{j=2N-1}^{N} |x_{j-n} - x_j|$$

Zpracovávané okno je složeno z předchozího a nového okna. Vstupní délka je tedy $2N$. Výsledné hodnoty nejsou zatíženy chybou od 200. vzorku jako je ukázáno na obrázku 2.7.

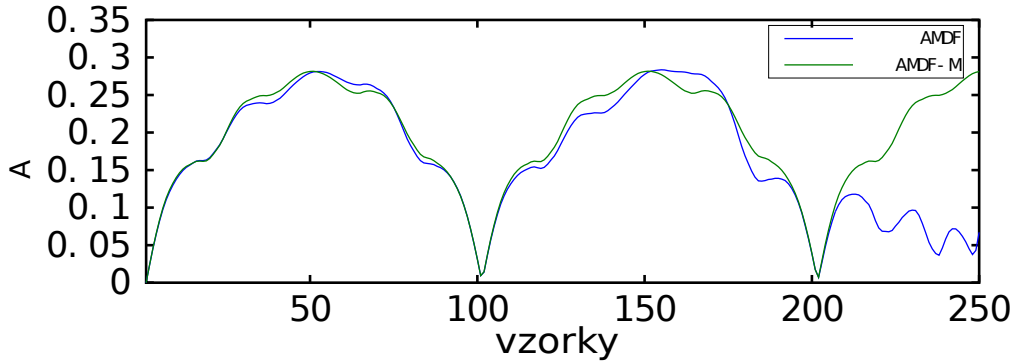
$$s = \frac{f_r}{f} \tag{2.9}$$

Velikost okna je zdola omezena nejnižší frekvencí, kterou je potřeba analyzovat a měla by být co nejmenší, kvůli výpočetní náročnosti a rychlejší odezvě. Vzorkovací frekvence závisí na největší frekvenci, protože nesmí být větší než Nyquistova frekvence (kapitola 1.1). Zvolil jsem délku okna 1024 vzorků, protože není potřeba ladit nižší frekvence než 220 Hz. V okně stačí zaznamenat alespoň 2 periody. Pro funkci $a(n)$ rozdělím okno na polovinu a získám předchozí a nové okno o délce 512 vzorků. Výsledná funkce bude dlouhá také 512 vzorků.

Implementoval jsem tedy čisté AMDF a jen provedl ořezávání tímto způsobem:



Obrázek 2.6: Rozložení metody AMDF+ACF na fáze



Obrázek 2.7: Srovnání AMDF a AMDF-M

```
for it = 1:N
  if (resAMDF(it) > tr)
    resAMDF(it) = 0;
  else
    resAMDF(it) = tr - resAMDF(it);
  endif
endfor
```

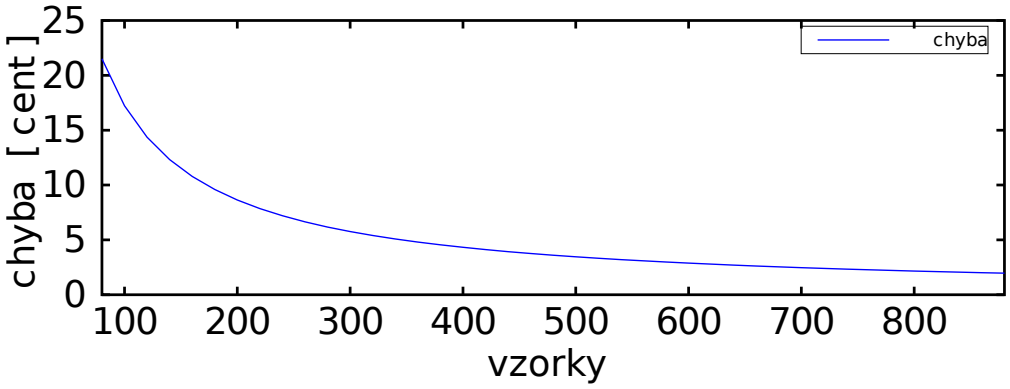
2.4 Hledání vrcholu

Následující kapitola trochu odbočí od jednotlivých metod a bude se věnovat společné úloze a to hledání vrcholů jednorozměrné funkce. V matematické analýze se jedná o úlohu hledání extrémů. Známe-li předpis funkce, pak můžeme převést úlohu na hledání nulové derivace, čili nulové změny. *Rolova věta* zaručuje že funkce na intervalu $\langle a, b\rangle$ má extrém rovnají-li se funkční hodnoty v krajních bodech a, b.

Obecně ale zkoumanou funkci neznáme. Víme ale například jak vznikla a jaký přibližně má průběh. V případě výsledku AMDF jsme sčítali jedničky a nuly a tedy nelze očekávat velké skoky. Může se ale vyskytnout lokální minimum a maximum.

Kdyby byla funkce spojitá, dokážeme určit vrchol jednoznačně. U diskrétní funkce záleží na interpretaci a je důležité uvědomit si, že celočíselná hodnota periody nemusí být přesně ta, kterou hledáme. Ve zvukovém záznamu znamená odchylka jednoho vzorku hyperbolickou odchylku frekvencí. Na obrázku 2.8 jsou vidět poměry souřadnic sousedních vzorků přepočítané na centy.

Z této vlastnosti plyne, že bychom chtěli hledat co nejnižší frekvence z konce okna. Jenomže metody často dávají výsledek na začátku a na konci okna selhávají. Ve vztahu 2.10 jsem ukázal, že tato chyba nezávisí na vzorkovací frekvenci (vycházím ze vztahu 2.1).



Obrázek 2.8: Rozdíl ve frekvenci sousedních vzorků.

$$\frac{\frac{f_r}{d_1}}{\frac{f_r}{d_2}} = \frac{d_2}{d_1} = \frac{101}{100} \tag{2.10}$$

Z tohoto důvodu jsem zavedl do vyhledávání aproximaci bodů pomocí nejmenších čtverců. Algoritmus je následující:

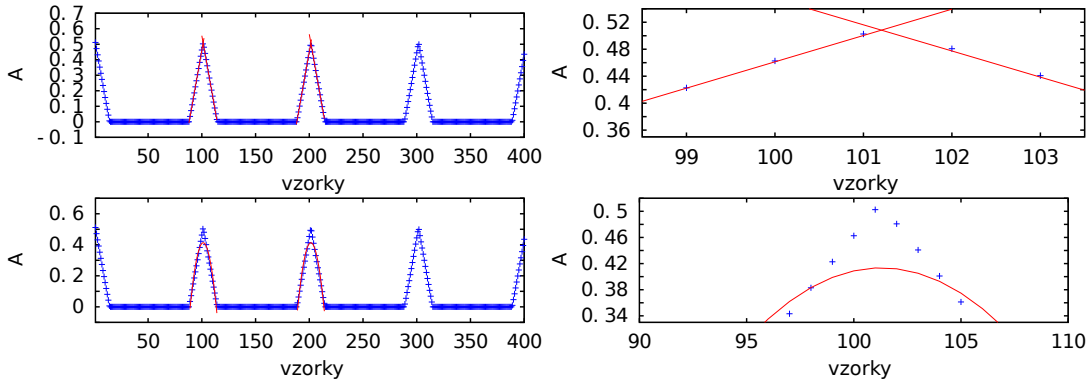
1. najdi nenulovou hodnotu → START
2. najdi nulovou hodnotu → END
3. vypočti aproximaci a její maximum z bodů mezi START a END-1
4. přidej nový vrchol pokud je menší/větší než dosud nalezený a hledej dál.

Celému cyklu předchází přeskočení prvních hodnot, které jsou typicky nezajímavé, protože vznikly korelací s malým posunem signálu. Nejsilnější korelace bude právě se signálem samotným.

V algoritmu jsem záměrně neurčil jakou aproximaci použít. Zkoušel jsem jak lineární tak kvadratickou funkci. Aproximace lineárním polynomem dávala lepší výsledky a navíc je rychlejší, takže to byla jasná volba. Na obrázku 2.9 je jasně vidět, že parabola je nevhodná. Polynom 4. stupně byl ještě horší.

Řešení leží mezi 100. a 101. vzorkem a maximum vyšlo v tomto případě na 101. Jak již bylo řečeno na začátku kapitoly, chyba na 100 vzorcích je poměrně velká a dosahuje až 17 centů, proto velmi záleží na zvolené aproximační funkci. Porovnal jsem vzdálenosti prvního vrcholku a potom rozdíl mezi prvním a druhým. Tím jsem dosáhnul dokonce přesnosti až jedné setiny.

vzdálenost	**přesná**	**lineární**	**kvadratická**
$1-0$	$100,23$	101.20	101.28
$2-1$	$100,23$	100.24	100.18



Obrázek 2.9: Lineární a kvadratická L2 aproximace pro AMDF

Cílem L2 aproximace je minimalizace odchylky funkce od tabulkových hodnot. Interpolace není vhodná protože pracuji s velkým množstvím bodů a stupeň polynomu by byl příliš vysoký. Interpolace Spline funkcemi je zas příliš silný nástroj. Minimalizuji tedy chybu $R(f, \varphi)$:

$$R(f, \varphi) = \sum_{i=0}^{n} [f(x_i) - \varphi(x_i)]^2$$

$f(x_i)$ jsou změřené hodnoty, funkce $\varphi(x)$ představuje aproximující funkci. Použitím lineární funkce $\varphi(x) = a + bx$, postačujících podmínek minima $\frac{\partial R}{\partial a} = 0$ a $\frac{\partial R}{\partial b} = 0$ získám soustavu rovnic (Weisstein, 2014):

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \end{bmatrix}$$

Nalezený interval START, END rozdělím na dvě části. Hodnoty před a za maximem. Hledám tedy aproximace $\langle \text{START}, \text{MAX} \rangle$ a $\langle \text{MAX}, \text{END} \rangle$. Tyto dvě přímky mají jeden společný průsečík a to hledané maximum. Koeficienty přímek označím (a, b) a (k, q).

Nyní když mám koeficienty aproximujících funkcí mohu spočítat vrchol. Průsečík dvou přímek zjistím rovností:

$$\begin{aligned} kx + q &= ax + b \\ kx - ax &= b - q \\ x &= \frac{b - q}{k - a} \end{aligned} \tag{2.11}$$

Funkční hodnotu získám dosazením do předpisu jedné z přímek. Ukázalo se, že nejlepší výsledky jsou mezi 1. a 3. vrcholkem. Počítal jsem průměr vzájemných vzdáleností jako v nejjednodušším případě 2.1.

2.5 Různá zlepšení výsledků

U všech dosud zmíněných metod je problém změna amplitudy v okně. V případě stacionárního signálu, je AMDF dostačující. Pokud je potřeba rozpoznávat například hlasové vzorky, dostáváme se k signálu s proměnnou amplitudou a frekvencí. Navržená metoda potom nedává přesné výsledky. Jeden návrh je popsán v (Guangyu – Shize, 2009). Signál se transformuje pomocí amplitudové funkce, která zarovná změny v amplitudě.

Při analýze signálů s pevnou délkou okénka nastává problém na začátku a konci signálu. To se dá omezit například pro čisté ACF, použitím Hammingova okna:

$$x_m = \frac{N}{N-m-1} \sum_{n=0}^{N-m-1} w_n s_n w_{n+m} s_{n+m}$$

Váhovací funkce je definována jako:

$$w_n = 0.53836 - 0.46164 * \cos\left(\frac{2\pi n}{N-1}\right)$$

Použít Hammingovo okno pro AMDF není moudré, protože jak bylo již zmíněno v úvodu této metody, tato metoda je závislá na amplitudě. Aby se omezil vliv nevýznamných frekvencí je ale možné použít filtr typu dolní propusť (Smith, 1998). Častým algoritmem v časovém pásmu je klouzavý průměr definovaný:

$$y_i = \frac{1}{M} \sum_{j=-\frac{M-1}{2}}^{\frac{M-1}{2}} x_{i+j}$$

kde M je počet vyhodnocovaných vzorků signálu (např. 5). Tento filtr průměruje vstupní signál a omezí tak vysoké frekvence. Aby byl rozsah symetrický, je třeba, aby číslo M bylo liché. Tato metoda je velmi jednoduchá, ale dává dobré výsledky.

2.6 Fourierova transformace

O zvukovém signálu víme, že se nalézá v tzv. časovém pásmu a je tedy funkcí času. Dostáváme tak *časově-amplitudovou* reprezentaci. V této formě ale vůbec nevíme jaká informace se v signálu skrývá, proto hledáme spektrum signálu. Ze *spektrálně-amplitudového* pásma už víme jaké frekvence jsou v signálu obsaženy a dokonce jak silně. Jednoduché příklady jsem uvedl již na začátku na obrázku 1.2.

Jedním ze způsobů jak určit spektrální vlastnosti signálu je Fourierova transformace (FT). Tato transformace nalezla uplatnění v mnoha oborech. Asi nejznámější

je její využití v kompresních algoritmech, které provádí filtrace ve frekvenčním pásmu a zpětnou transformací a dalšími algoritmy vytváří komprimovaný signál. Tento proces je výpočetně náročný. Základní DFT (Diskrétní furierova transformace) dosahuje složitosti $\mathcal{O}(N^2)$ komplexních součinů. Definována je:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i}{N} nk} \qquad k = 0, \ldots, N-1$$

Transformaci lze použít pro zpracování v reálném čase za použití Rychlé fourierovy transformace (FFT), která dosahuje složitosti $\mathcal{O}(N \log N)$ díky metodě rozděl a panuj. Inverzní transformace znamená jen změnu ze záporného znaménka exponenciely na kladné. Všimněte si, že délka okna zůstává stejná jako původní signál. Množina frekvencí, které FT dokáže rozlišit souvisí se vzorkovací frekvencí a délkou okna. Nejvyšší zaznamenatelná frekvence odpovídá vzorkovací frekvenci f_r. Například pro $f_r = 44100$ a přesností na 1 Hz, by musel být signál dlouhý 44100 vzorků, tj. 1 s. Rozlišitelnost v čase a frekvence jde proti sobě. Čím větší bude okno, tím více frekvencí bude obsaženo ve spektru a bude méně jednoznačné.

2.7 Cepstrum

Všechny dosud zmíněné metody hledaly dominantní frekvenci. Při ladění tento výsledek stačí. Existuje ale metoda nalezení základní frekvence používaná hlavně k analýze hlasu.

Vychází se z toho, že hlas $x(t)$ vzniká konvolucí zdrojového zvuku $s(t)$ a filtrem $h(t)$. Zdrojový zvuk vzniká chvěním hlasivek a filtr je tvořen rezonancí v ústní dutině. Fourierovou transformací lze tyto frekvence oddělit. Nejdříve získáme spektrum $X(t) = \ln(|\mathrm{FFT}(x(t))|)$. Opakováním FFT dostaneme Cepstrum v jednotkách *quefrency*. Nízké quefrence odpovídají modulační funkci $H(t)$ a zbylé odpovídají $S(t)$. Nalezené maximum pak odpovídá F_0. Více se lze dočíst v (McLeod, 2008).

Kapitola 3

Implementace

Historie této aplikace sahá až na začátek roku 2012, kdy se jednalo o první pokusy aplikace pro Android[1]. Začalo to jako metronom a postupně jsem přidával další součásti. Nejdříve tónový generátor a pak rekordér. Nejlákavější součást byla ale ladička. Když jsem zjistil, kolik různých metod existuje, napadlo mě že by to bylo dobré téma bakalářské práce.

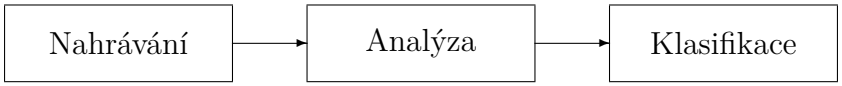
3.1 Příprava v Octave

Pro potřeby tohoto textu jsem implementoval všechny algoritmy v matematickém nástroji Octave. Jedná se volnou alternativu komerčního programu Matlab. Vytvořené zdrojové soubory přikládám k této práci. Důležitá byla zpětná kontrola s finální implementací. Mnohem snáz se totiž ladí chyby v Octave, než ve finální aplikaci. Ladění na mobilu jsem se samozřejmě nevyhnul, ale minimalizoval.

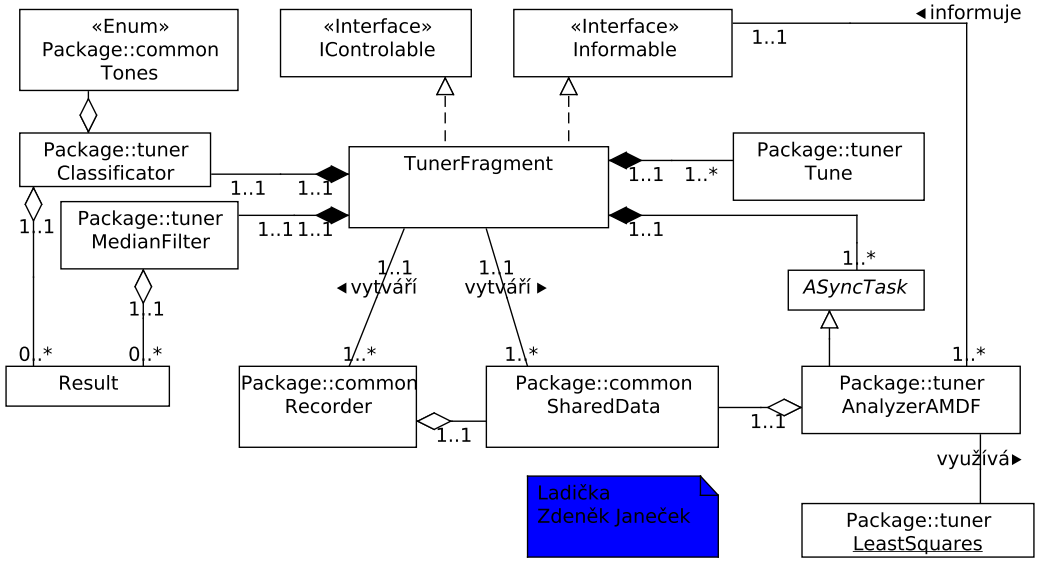
V tomto jazyce mi nevyhovovalo číslování indexů $1 \ldots N$, naproti obvyklému $0 \ldots N-1$. Některé indexy jsem musel posouvat o jedna a pak zase zpátky a vznikla tím nejedna chyba v kódu. Nevýhodu zas kompenzují funkce `sum`, `min`, `polyfit` (aproximace bodů polynomem), `polyval` (vyčíslení polynomu) a `plot` (kreslení grafů) šetřící práci.

Testování všech metod jsem prováděl jak na reálném zvuku, tak na přesné sinusoidě o známé frekvenci. V případě Octave jsem generoval zvuk pomocí interní funkce sinetone (FREQ, RATE, SEC, AMPL), kde parametry znamenají požadovanou frekvenci, vzorkovací frekvenci, délku v sekundách a amplitudu. Zvukové nahrávky lze načítat do vnitřní proměnné Octave pomocí **wavread** (FILENAME). Používal jsem WAV, protože jej lze vytvořit a otevřít kdekoli. Implementoval jsem si také vlastní načítání a zápis pro potřeby mé aplikace *Muassist*. Formát souboru

[1] stále dostupné z `<https://code.google.com/p/metronome-for-android>`



Obrázek 3.1: Blokové schéma ladičky



Obrázek 3.2: Diagram tříd

je popsán například na stránce `<https://ccrma.stanford.edu/courses/422/projects/WaveFormat/>`.

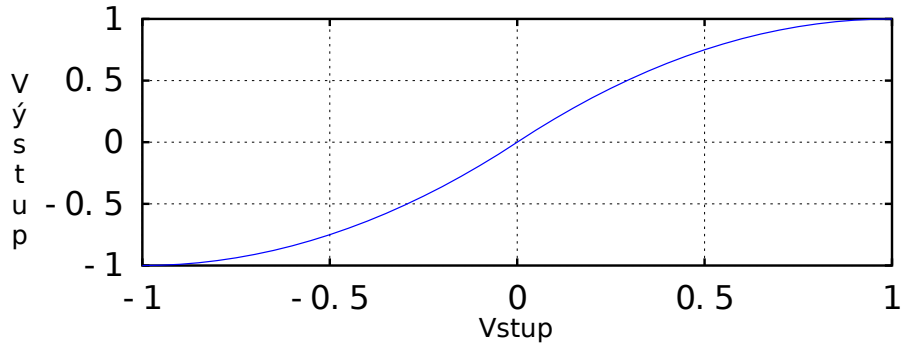
Nevýhodou reálného zvuku flétny, který jsem nahrával je, že nevím jakou frekvencí přesně zní. Navíc lze snadno zjisti, že se příliš neliší od čisté sinusoidy, takže nic nebrání tomu testovat na ní a přidat šum z funkce **randn**(M,N), kde M je délka a N dimenze.

3.2 Android

Pro cílovou implementaci jsem zvolil Android API[2]. Úloha nevyžaduje příliš vysoké nároky aby byla psána v nativním kódu. Problémy s takovým kódem by převážily výhody. Při psaní jsem mohl také ladit kód v Java SE, protože jazyk je stejný.

V aplikaci se stále dokola opakuje proces popsaný blokovým schématem 3.1. Každý krok odpovídá více spolupracujícím modulům, a proto zde uvádím pro

[2]http://en.wikipedia.org/wiki/Comparison_of_Java_and_Android_API



Obrázek 3.3: Transformace chyby pro uživatele

lepší orientaci v textu UML diagram 3.2, který jsem omezil pouze na názvy tříd. V příloze se nachází jeho rozšířená verze.

Hlavní třídou ladičky, která vytváří a zobrazuje výsledky je TunerFragment. Její životní cyklus určuje rodičovská třída Fragment a běží na UI vlákně. Obsahuje vlastní komponentu Tune, která zobrazuje odchylku od přesné hodnoty. Její hodnota se změní voláním setVal() a musí být mezi 0 a 1. TunerFragment splňuje rozhraní Informable, takže nové frekvence získává metodou postInformation(). Její tělo je následující:

```
public void postInformation(Double freq) {
  // klasifikuje aktuální frekvenci a přidá do seznamu
  // instance MedianFilter
  mf.addValue(classify.findTone(freq));

  // najde medián ze seznamu MedianFilter
  Result r = mf.getMedian();

  // vypíše výsledek do návěští tunerText
  tunerText.setText(String.format("%s\n%.1f Hz",
      r.getTone(), r.getFreq()));

  if (r.getError() >= 0)
    f = 1 - (r.getError() - 1)^2;
  else
    f = (-r.getError() - 1)^2 - 1;

  // nastaví vizuální odchylku
  bar.setVal(f);
}
```

Získanou frekvenci od analyzátoru nejdříve zpracuji klasifikátorem. Klasifikátor vrací instanci třídy Result, která obsahuje informaci o frekvenci, tónu a odchylce. Přidání do MedianFilter zajišťuje zařazení do seřazeného seznamu v čase $\mathcal{O}(n)$ a medián najdu za $\mathcal{O}(1)$. Medián je prostřední hodnota seřazeného seznamu. Výslednou chybu nezobrazuji lineárně, ale upravím posunutou parabolou. To mi zajistí větší citlivost kolem nuly a uživatel lépe vnímá výsledek. Funkce je na obrázku 3.3. Ve zdrojovém kódu nahoře jsem zkrátil zápis druhé mocniny, ve skutečnosti je výraz delší.

Výběr referenční frekvence vybírá uživatel pomocí NumberPicker. Tato změna se propaguje do klasifikátoru metodou changeRef(). Historicky se totiž mnohokrát měnila až do roku 1939, kdy bylo navrženo mezinárodní konferencí 440 Hz. Možné je tedy volit frekvence od 390 do 460 Hz.

3.3 Klasifikátor

V prvním kroku spočítám poměr analyzované frekvence a referenční, poté spočítám jeho dvojkový logaritmus. Tento přepočet odpovídá centům jen s tím rozdílem, že nenásobím 1200 (vztah 1.1). Pak se zbavuji násobných oktáv, abych omezil množinu tónů jen na 12 půltónů tj. interval $\langle 0; 1)$ tím, že odříznu celou část. Nalezenou hodnotu pojmenuji d. U frekvencí nižších než referenční posouvám d o jedničku.

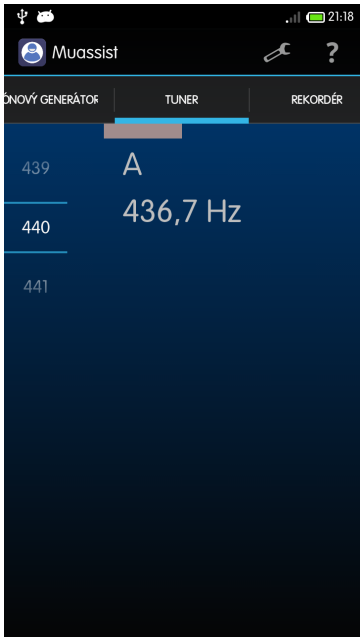
Hodnotu d vynásobím 1200 a tím dostanu hodnotu v centech. V závislosti na zvoleném ladění v nastavení aplikace zavedu d_k jako pole intervalů z tabulky 1.1. Hledaný tón se pak nachází nejblíže k jednomu z nich:

$$\arg_k \min_{k=0}^{12} \{|d - d_k|\}$$

Nalezený index k odpovídá tónu A, ale výčet Tones je počítán od C, takže musím provést posun o 9. Největší vzdálenost mezi půltóny je v průměru 100 centů a tedy chyba od přesné hodnoty může být 50, protože je přesně v polovině. Chybu dostanu mezi $\langle -1; 1\rangle$ vydělením 50. Výsledek klasifikace je tedy třída Result s atributy nalezeného tónu, chybou a analyzovanou frekvencí.

3.4 Použité prostředky

Každá otevřená aplikace v Androidu dědí od třídy Activity (česky aktivita). Životní cyklus mobilní aplikace (aktivity) se liší od Deskopové tím, že je potřeba minimalizovat spotřebu baterie. Jakmile se přesune aplikace na pozadí, měla by omezit své nároky na procesor. Minimalizování prohlížeče na počítači jen uvolní prostor na obrazovce, ale aplikace běží typicky dále.



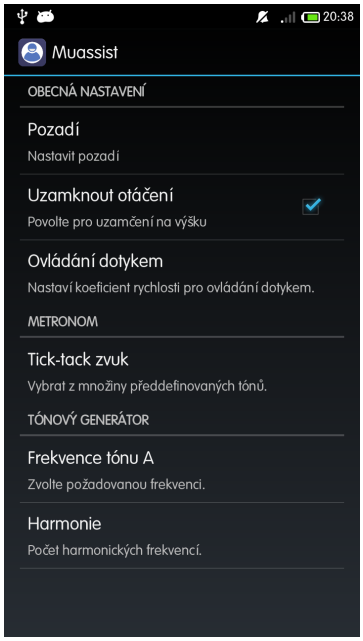
Obrázek 3.4: Okno ladičky

Od verze *Honeycomb* je možné přidávat do aktivity nejen widgety, ale také fragmenty. Fragment má podobné vlastnosti jako aktivita, jen s tím rozdílem, že je lze volně přidávat na obrazovku stejně jako widgety (tlačítka, textová pole, obrázky, . . .). Vytvořil jsem tedy pro každou součást jeden fragment. Pro přepínání mezi fragmenty jsem využil záložky, které spravuje ActionBar. Jejich umístění se mění podle místa na obrazovce. Při orientaci na výšku vypadají jak na obrázku 3.4. Po otočení na šířku se změní na drop-down navigaci a umístí vedle loga a názvu.

Pro ukládání stavů a uživatelem definované hodnoty využívám *Shared Preferences*. Jedná se o prostor, kde si může každá aplikace uložit svá data. Přistupovat lze k nim jen za pomoci dodaného API a jsou ostatním skryta. Není potřeba žádného speciálního oprávnění. Čtení a zápis jsem implementoval ve třídě SharedPref. Uživatel může nastavit některé parametry pomocí *Preference Activity* (Obrázek 3.5).

Součástí je také balík *Android Support*, díky kterému lze aplikaci přizpůsobit i pro starší verze systému. Mezi lidmi je stále mnoho telefonů, které jsou z řady 2.x. Kvůli tomu jsem musel použít Widget výběru čísla hostovaného na `<https://github.com/michaelnovakjr/numberpicker>`.

Při návrhu aplikace jsem musel velmi často využít oddělené vlákno. Každé zdržení na UI (user interface — uživatelské rozhraní) vlákně znamená horší odezvu aplikace. Vedle standardních nástrojů, které jsou přejaty z Javy, disponuje API



Obrázek 3.5: Nastavení aplikace

ještě dalšími. Velmi často je třeba pustit nějakou úlohu na pozadí a výsledek vykreslit na obrazovku. Všechna tato kreslení musí probíhat na UI vlákně. Toho lze dosáhnout buď zavoláním post() na handleru, který lze dostat z jakékoli komponenty UI. Mnohem jednodušší je ale využít třídu ASyncTask, která se postará o správné vykonání. Využít lze následující metody:

- onPreExecute – spuštěno jednou před začátkem na UI vlákně
- doInBackground – běží odděleně, obdoba run() z Thread.
- onProgressUpdate – pro aktualizace stavu progressbaru, textu spuštěno pomocí publishProgress (Progress ... values).
- onPostExecute – spuštěno na konci úlohy, tj. při opuštění doInBackground.

Této třídy jsem využil pro analyzátor ladičky, kvůli potřebě aktualizovat UI. Tam kde to ale není potřeba jsem použil standardní Thread. Takovým příkladem je nahrávání zvuku. Sjednotil jsem nahrávání pro rekordér a ladičku do jedné třídy. Tyto dva moduly se liší jen ve velikosti požadovaného bufferu. Zatímco okno pro analýzu musí být co nejmenší. Zápis do souboru je vhodnější po větších částech.

Pro sdílení dat mezi vlákny používám rendezvous bod a vzájemné vyloučení zajišťuji pomocí monitoru. Vytvořením instance třídy SharedData vzniká současně

také monitor nad touto třídou. Této úloze se říká producent-konzument. Obecně může být více producentů a konzumentů, ale tato aplikace je nepotřebuje. Kód konzumenta funguje následovně:

```
synchronized (sd) {
  while (!sd.available) {
    sd.wait();
  }
  // zpracuj sd.shortBuffer;
  sd.available = false;
  sd.notify();
}
```

Producent je následující:

```
synchronized (sd) {
  while (sd.available) {
    sd.wait();
  }
  // zapiš do sd.shortBuffer;
  sd.available = true;
  sd.notify();
}
```

Test na pravdivost a přiřazení logické hodnoty patří mezi atomické hodnoty. Proměnná available indikuje připravenost dat. Tělo monitoru je ohraničeno pomocí **synchronized**.

Pro nahrávání zvuku z interního mikrofonu slouží třída AudioRecord. Rozhraní AudioRecord.OnRecordPositionUpdateListener definuje callbacky v případě, že rekordér naplnil svůj buffer na požadované množství. To muže být vymezeno množstvím vzorků v interním bufferu, nebo počtem požadovaných vzorků. Pokud programátorovy nevyhovují callbacky je možné přímé čtení pomocí read():

```
recorder.read(sd.byteBuffer, 0, sd.byteBuffer.length);
```

S přímým čtením mám jistotu, kdy čtení skončí. U callbacků jsem narážel na deadlock při vypnutí. To že jsem ukončil rekordér, ještě neznamenalo, že se callback už nezavolá. Z toho důvodu jsem přešel na přímé čtení i s tím rizikem, že může vzniknout v signálu díra. Operace read totiž není blokující a chybějící vzorky zaplní nulami. Tento jev se projevoval jen výjimečně, a proto ho nepovažuji za důležitý.

Čtením získávám pole bajtů. Jenomže zvuk nahrávám 16 bitový a bohužel není v dokumentaci uvedena výsledná endianita. Našel jsem, že se jedná o Little Endian, přestože Dalvik[3] je primárně Big endian. Nejspíš je tomu z důvodu formátu WAV,

[3]běhové prostředí pro systém Android

který je právě Little endian. Vytvoření jednoho čísla **short**, dělám pomocí bitových operací. Stačí zajistit, aby LSB (nejméně významný) bit byl na nejvyšší adrese. Ten se u Little endianu nachází na `i`-té adrese. Metoda je následující:

```
void prepareResults(byte[] src, short[] dst) {
  for (int i = 0, j = 0; i < src.length; i += 2, j++) {
    dst[j] = (short) (u_byte(src[i]) |
              (u_byte(src[i + 1]) << 8));
  }
}
```

Metoda u_byte() převádí správně ne-znaménkový bajt na integer. Java totiž nezná ne-znaménkové datové typy a chybně je interpretuje. Proto jsem zavedl konverzní metodu:

```
int u_byte(byte b) {
  return b & 0xFF;
}
```

Práce s binárními daty není doména Javy. Výhodou oproti jazyku C je zaručená endianita datových typů, takže mohu vždy počítat s Big endian, přestože procesory ARM umějí pracovat v obou režimech.

Třída Recorder po spuštění čeká až budou prázdná data a běží na samostatném vlákně. Ukončí se zavoláním stopRecording(), jehož stavem isRecording() by se měli řídit také všichni konzumenti. Po jeho skončení už nelze znovu nastartovat, protože je vlákno ukončeno a musí se znovu vytvořit. Opětovné zavolání start() vyvolá výjimku.

Poslední společnou částí je implementace dotykového ovládání. Při vytvoření aktivity vytvářím třídu TouchControl, které se přiřazuje fragment jako callback. Každý fragment proto splňuje rozhraní IControlable:

```
interface IControlable {
  void onValueChange(TouchControl t, int val);
  void onToggle(TouchControl t, int state);
  void onPositionChange(TouchControl t, int x, int y);
}
```

Změna souřadnice je měřena v pixelech za vteřinu, *toggle* změnu stavu a změna pozice je jen přesměrování události se souřadnicí. Změna stavu nastává po dvojitém poklepání. Prodleva dvojího poklepání je pevně stanovena na 500 ms, ale dá se zvolit rychlost změny pohybu.

3.5 Uživatelská dokumentace

Překlad ze zdrojových kódů uživatel běžně nedělá. Dnes každý stahuje hotové balíčky z obchodu Google Play (bývalý Market), kde si může přečíst hodnocení druhých a dostávat automaticky aktualizace. Muassist doporučuji stahovat a instalovat především odsud. Stále podporovaná možnost je stažení APK balíčku z `<http://home.zcu.cz/~janecekz/Muassist>`, kam nepravidelně přidávám balíčky.

Zdrojové kódy se nachází na veřejném GIT repozitáři na adrese `<https://github.com/qwertzdenek/Muassist>`. Aplikace je vyvíjena v Eclipse ADT a je také preferován jako překladové prostředí. K práci jsem přiložil projekt připravený na překlad jen za pomocí nástroje Ant. Jediné co je potřeba je Android SDK, které lze stáhnout ze stránky `<http://developer.android.com/sdk/index.html>`. Některé Linuxové distribuce poskytují také svůj balíček. Například v Archlinuxu instaluji:

```
yaourt -S android-sdk android-sdk-build-tools \
    android-sdk-platform-tools
```

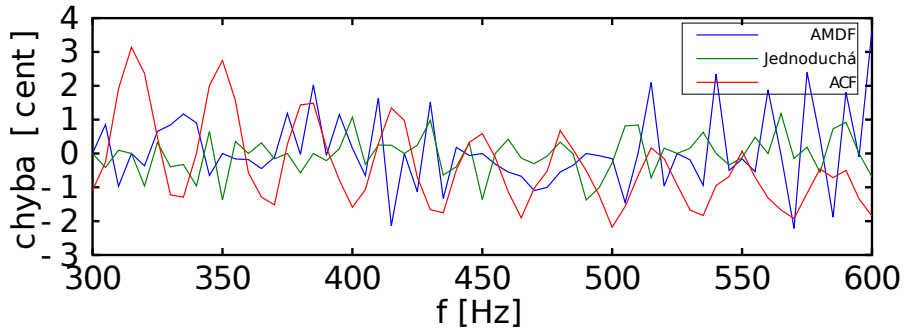
Poté je třeba stáhnou pomocí aplikace android potřebné SDK, vůči kterému bude kód překládán. Mělo by být menší nebo rovno cílenému, které je uvedeno v AndroidManifest.xml.

Nyní je potřeba vygenerovat soubory pro překlad v projektu a také v závislostech. Po rozbalení souboru se zdrojovým kódem by měli vzniknout tři složky app, lib a picker. Nejprve je potřeba zjistit id <api> příkazem android list targets. Poté lze zavolat:

```
android update project --name Muassist \
   --target <api> --path app
android update lib-project --target <api> --path lib
android update lib-project --target <api> --path picker
```

Nyní lze pustit překlad jako ant release, nebo ant debug. Release balíček se vytvoří, ale nelze ho nainstalovat, protože neobsahuje podpis. Ten lze dodat, ale lepší je použít debug, který obsahuje klíč se známým heslem a je určen jen na účely testování. Balíček stačí nakopírovat na SD kartu a nainstalovat. Některá zařízení se brání instalování balíčků z jiných zdrojů než market, a proto je potřeba jejich instalaci povolit v nastavení zabezpečení.

Po spuštění se otevře jako první metronom. Jeho ovládání stejně jako zbytek aplikace je navrženo pro ovládání jedním prstem. Dotykem na obrazovku lze měnit tempo, které lze přečíst na číselníku vpravo. Dvojitým ťuknutím se vypne/zapne příslušná funkce. Jediná ladička této funkce nevyužívá a běží po celou dobu. Jediné co se dá nastavit, je komorní A a to pomocí číselníku vlevo.



Obrázek 3.6: Závislost chyby na frekvenci čisté sinusoidy

	sinus [Hz]	flétna [Hz]	housle [Hz]	hlas [Hz]
Jednoduchá	499,87	876,46	435,20	329,1
ACF	500,08	878,50	440,18	673,6
AMDF-M	499,95	878,98	438,67	344,62
Přesné	**500**	**880**	**450**	**648**

Tabulka 3.1: Srovnání vybraných metod.

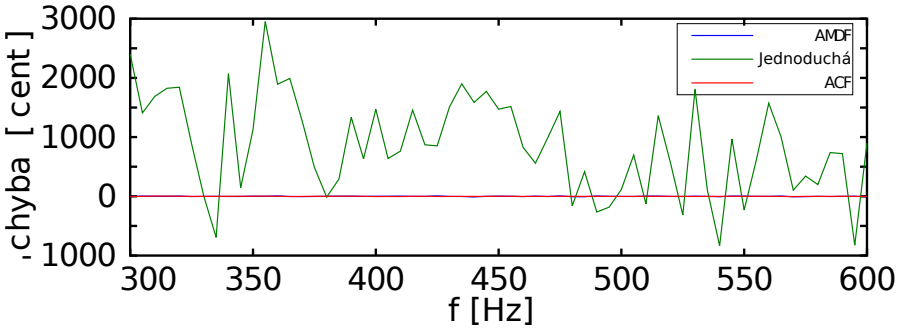
V pravém horním rohu se nachází tlačítko „o aplikaci“ a „nastavení“. V nastavení lze změnit barvu pozadí, vypnout otáčení a rychlost změny například tempa u metronomu. V současné době lze nastavit zvuk metronomu, komorní A pro tónový generátor a počet harmonických (částkových) tónů. Ladičku lze přepnout také na různá ladění.

3.6 Zhodnocení a budoucnost

V této práci jsem experimentoval celkem se třemi variantami autokorelace, square difference funkcí a jejich spojením ve formě SNAC a také rozdílovou AMDF. U všech metod byl nakonec největší problém nalezení vrcholků. Někdy jsem použil lineární regresi, parabolu, nebo průměroval přesně nalezené vrcholky. Neexistuje nejlepší metoda pro všechny. Vždy závisí na konkrétním uspořádání bodů, a proto jsem zkoušel různé možnosti.

Každá metoda musela splňovat podmínku nezávislosti na frekvenci a šumu. První podmínku jsem testoval na frekvencích v různém rozsahu a zobrazil jí na jednom grafu 3.6. Vstupem byla přesná sinusoida bez šumu. Naivní metoda hledání vrcholků funguje dokonce lépe než všechny ostatní.

U zašuměného signálu náhodnými hodnotami ale selhává (obr. 3.7). V tabulce 3.1 jsou uvedeny konkrétní hodnoty. Vybral jsem ACF typu I s Hammingovo oknem jako nejlepší reprezentant všech autokorelací.



Obrázek 3.7: Závislost chyby na frekvenci zašuměného signálu

Zvolil jsem AMDF kvůli nižšímu počtu operací násobení v desetinných číslech. Nevyhnul jsem se mu ale při hledání vrcholků. Dlouhou dobu jsem věřil modifikaci v sekci 2.3, ale nedařilo se mi dostat pod chybu 5 centů. Zjistil jsem, že největší problém vytvářelo prahování na binární posloupnost, které posouvalo vrchol.

Při hledání materiálů jsem narazil na zajímavé klasifikační metody rozpoznávání nástrojů. Využití algoritmu popsaného v (Marques – Moreno, 1999), by byla zajímavá věc navíc. Primárně slouží na automatické značkování rozsáhlých hudebních katalogů.

Do budoucna bych rád aplikaci rozšiřoval. Nový uživatel by ocenil jasnější ovládání, více nastavení a další funkce. Průběžně proto aktualizuji stránku `<https://github.com/qwertzdenek/Muassist/wiki/Improvement-ideas>`, kde dávám dohromady různá rozšíření.

Kapitola 4

Závěr

V první části jsem popsal jak tón vzniká a jak ho zaznamenáváme. Objasnil jsem základní pojmy digitálního signálu a upozornil na případné problémy s ním. Popsal jsem, jak probíhá ladění u různých nástrojů a také jejich charakteristiky. Dále jsem objasnil různé druhy ladění od historických až po dnes sjednocené a všeobecně uznávané rovnoměrně temperované.

Poté jsem popsal různé metody pro nalezení periodických vlastností signálu. Ověřil jsem jejich vlastnosti implementací v matematickém nástroji Octave a uvedl příklady výsledků jak pro nezašuměný signál, tak pro zašuměný. Metody jsem srovnával podle přesnosti a stability, ale zohlednil jsem i výpočetní náročnost. Cílem byla implementace na reálné zařízení se systémem Android se zpracováním v reálném čase. Z testovaných metod jsem do mé aplikace vybral AMDF, protože nejlépe splňovala podle mého požadované vlastnosti. Podrobné výsledky jsem shrnul v kapitole *Zhodnocení a budoucnost*. Navrhl jsem metodu hledání vrcholků a klasifikace tónu podle různých druhů ladění.

V poslední části jsem vysvětlil rozdíl testovacího prostředí s cílovým a uvedl použité nástroje. Pro neznalé Android API jsem přiblížil, jak vypadá zevnitř aplikace pro Android. Použil jsem vestavěné API k získání zvuku, provedl potřebné úpravy signálu a vlákna pro oddělení od uživatelského prostředí. S použitím vláken vznikla potřeba ochrany dat před souběhem více vláken. Nakonec jsem popsal implementaci klasifikace, zobrazování výsledků uživateli, všechny další potřebné součásti aplikace a překlad ze zdrojových kódů. Výslednou aplikaci jsem publikoval na Google Play a mám ji v plánu dále rozšiřovat.

Příloha A

Dodatky

A.1 Hudební teorie

Tabulka intervalů:

#	tón	interval
0	D	prima
1	Es	malá sekunda
2	E	velká sekunda
3	F	malá tercie
4	Fis	velká tercie
5	G	čistá kvarta
6	As	Tritón
6	Gis	zvětšená kvarta
7	A	čistá kvinta
8	B	malá sexta
9	H	velká sexta
10	C	malé septima
11	Cis	velká septima
12	D	oktáva

půltón nejmenší vzdálenost mezi tóny, odpovídá přibližně 100 centům

interval vzdálenost mezi dvěma tóny

akord více tónů najednou

A.2 Obsah CD

src zdrojové kódy aplikace

doc text bakalářské práce

octave testovací m-soubory pro Octave včetně testovacích dat

A.3 Seznam zkratek

STFT Short Time Fourier Transformation

ACF Autocorrelation Function

SDF Square Difference Function

SNAC Special Normalisation of the Autocorrelation

AMDF Average Magnitude Difference Function

FFT Fast Fourier Transform

FT Fourier Transform

ADT Android Developer Tools

API Application programming interface

Java SE Java Second Edition

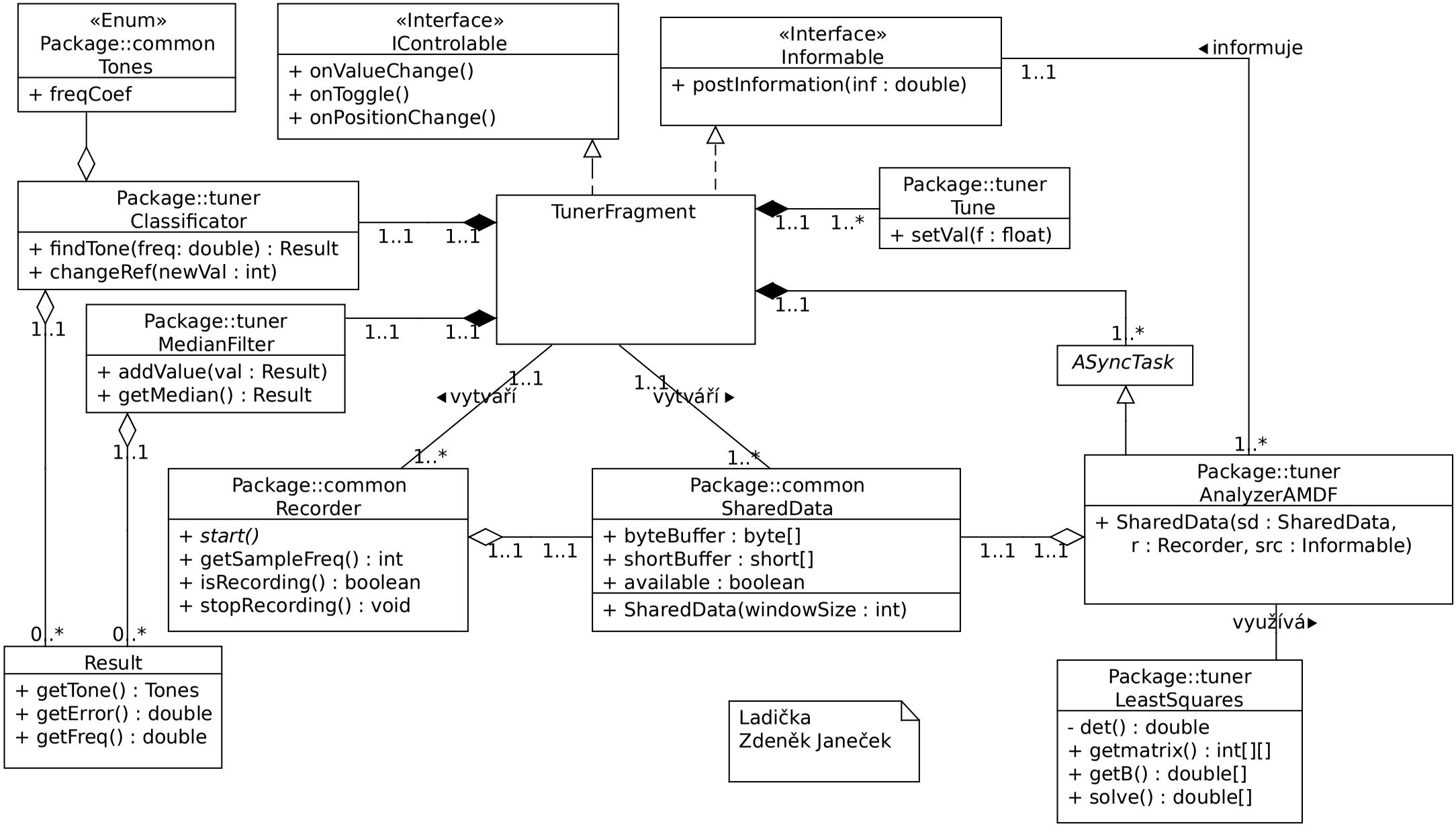
UML Unified Modeling Language

UI User Interface

LSB Least Significant Bit

APK Android application package

GIT Distributed version control system


«Enum»
Package::common
Tones
+ freqCoef
«Interface»
IControlable
+ onValueChange()
+ onToggle()
+ onPositionChange()
«Interface»
Informable
+ postInformation(inf : double)
informuje
1..1
Package::tuner
Classificator
+ findTone(freq: double) : Result
+ changeRef(newVal : int)
TunerFragment
Package::tuner
Tune
+ setVal(f : float)
1..1
1..*
Package::tuner
MedianFilter
+ addValue(val : Result)
+ getMedian() : Result
ASyncTask
vytváří
Package::common
Recorder
+ start()
+ getSampleFreq() : int
+ isRecording() : boolean
+ stopRecording() : void
Package::common
SharedData
+ byteBuffer : byte[]
+ shortBuffer : short[]
+ available : boolean
+ SharedData(windowSize : int)
Package::tuner
AnalyzerAMDF
+ SharedData(sd : SharedData,
r : Recorder, src : Informable)
využívá
Result
+ getTone() : Tones
+ getError() : double
+ getFreq() : double
Package::tuner
LeastSquares
- det() : double
+ getmatrix() : int[][]
+ getB() : double[]
+ solve() : double[]
Ladička
Zdeněk Janeček
0..*

Literatura

Guangyu, K. – Shize, G. Improving AMDF for Pitch Period Detection. Technical report, The Ninth Internation Conference on Electronic Measurement & Instruments, 2009. lijunbao@hit.edu.cn.

Hui, L. – Dai, B. – Wei, L. A pitch detection algorithm based on AMDF and ACF. Technical report, Microsoft Key Laboratory of Multimedia Computing and Communication, University of Science and Technology of China, 2006.

Jacob, B. – Sondhi – Yiteng, H. *Springer Handbook of Speech Processing*, Pitch, s. 65. Springer-Verlag, Berlin Heidelberg, 2008. ISBN 978-3-540-49125-5.

Marques, J. – Moreno, P. J. A Study of Musical Instrument Classification Using Gaussian Mixture Models and Support Vector Machines. Technical report, Cambridge Research Laboratory, 1999.

McLeod, P. Fast, Accurate Pitch Detection Tools for Music Analysis. Master's thesis, University of Otago, Dunedin, May 2008.

Montgomery, C. *A Digital Media Primer For Geeks* [online]. 2010. Dostupné z: <http://goo.gl/AQij>.

Polikar, R. *The Wavelet Tutorial* [online]. 1996. [cit. 24. dubna 2014]. Dostupné z: <http://goo.gl/asHsk>.

Rusňák, K. Skládání rovnoběžných kmitů. *Přednášky z FYA1*. 2008.

Smith, S. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1998.

Tervaniemi, M. et al. Pitch discrimination accuracy in musicians vs nonmusicians: an event-related potential and behavioral study. Technical report, Springer-Verlag, 2004.

Weisstein, E. *Least Squares Fitting* [online]. MathWorld – A Wolfram Web Resource, 2014. [cit. 3. dubna 2014]. Dostupné z: <http://goo.gl/gjtQqP>.

Zenkl, L. *ABC hudební nauky*, Tónové systémy a soustavy ladění. Supraphon, Palackého 1, Praha, 1982. Třetí, nezměněné vydání.