ŽILINSKÁ UNIVERZITA V ŽILINE

Fakulta riadenia a informatiky

BAKALÁRSKA PRÁCA

Určovanie základnej frekvencie pre hudbu algoritmom YIN

Žilina, 2014 Martin Dindoffer

ŽILINSKÁ UNIVERZITA V ŽILINE

FAKULTA RIADENIA A INFORMATIKY

URČOVANIE ZÁKLADNEJ FREKVENCIE PRE HUDBU ALGORITMOM YIN

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	9.2.1 Informatika
Školiteľ:	Ing. Ondrej Škvarek, PhD.

Žilina, 2014

Martin Dindoffer

Poďakovanie

Chcem sa poďakovať Davidovi Gilbertovi, tvorcovi knižnice JFreeChart, za nápad prekryť v triede NumberAxis metódu refreshTicks a takisto mojej rodine a Romanovi Kompaníkovi, Dis. art. za veľkú trpezlivosť a ochotu pomôcť.

Abstrakt

Cieľom práce je implementácia a otestovanie funkčnosti algoritmu YIN pri detegovaní tónov v monofonických nahrávkach hudobných nástrojov. V práci je najskôr uvedený prehľad základných algoritmov pre detekciu základnej frekvencie a popis algoritmu YIN. Ďalej je v nej opísaná implementácia tohto algoritmu v Jave, generovanie hudobných nahrávok a následná evalvácia výsledkov získaných algoritmom YIN z týchto nahrávok. Na záver hodnotí použitie algoritmu YIN v hudobnom prostredí.

Kľúčové slová: pitch detection algorithm, YIN, Java, hudobné nástroje

Abstract

The objective of this work is to implement and test functionality of the YIN algorithm for detection of pitches in monophonic recordings of musical instruments. It contains an overview of basic pitch detection algorithms and a description of the YIN algorithm. Apart from that it contains a description of the implementation of this algorithm in Java, a description of the generation of musical recordings and subsequent evaluation of the results acquired from these recordings using the YIN algorithm. Finally it judges the use of the YIN algorithm in a musical environment.

Keywords: pitch detection algorithm, YIN, Java, musical instruments

Obsah

Zoznam ilustrácií

Zoznam skratiek a značiek

- ACF – AutoCorrelation Function, funkcia autokorelácie
- AIFF – Audio Interchange File Format, formát súborov určený pre uchovávanie nekomprimovaných zvukových záznamov
- AMDF – Average Magnitude Difference Function, vážená diferenčná funkcia
- AU – Formát súborov určený pre uchovávanie nekomprimovaných zvukových záznamov
- BPM – Beats Per Minute, počet dôb za minútu
- CMNDF – Cumulative Mean Normalized Difference Function, diferenčná funkcia normalizovaná kĺzavým kumulatívnym priemerom
- DFT – Discrete Fourier Transform, diskrétna Fouriarova transformácia
- EDT – Event Dispatch Thread, vlákno v Jave, ktoré spracúva GUI udalosti
- F_0 – Fundamentálna frekvencia
- FFT – Fast Fourier Transform, rýchla Fourierova transformácia
- GUI – Graphical User Interface, grafický užívateľský interfejs
- HPS – Harmonic Product Spectrum, spektrum harmonických zložiek signálu
- JIT – Just-in-time, dynamická kompilácia počas behu programu
- MIDI – Musical Instrument Digital Interface, komunikačný protokol a interfejs pre komunikáciu hudobných nástrojov
- PDA – Pitch Detection Algorithm, algoritmus detegujúci výšku tónu
- SMF – Standard MIDI File, formát súborov určených pre uchovávanie MIDI záznamov
- τ_{max}– Maximálna hodnota τ, t.j. najväčšia hľadaná perióda vo vzorkách
- VST – Virtual Studio Technology, softvérový interfejs pre softvérové syntetizátory, zvukové efekty a zvukové editory
- WAVE – Waveform Audio File Format, formát súborov určený pre uchovávanie nekomprimovaných zvukových záznamov
- ZCR – Zero Crossing Rate, frekvencia pretnutia nuly signálom

1 Úvod

Určovanie fundamentálnej frekvencie signálu má veľmi široké využitie, najmä v systémoch rozpoznávania reči. V hudobnom nasadení má veľké využitie napr. pri automatizovanej transkripcii hudobných diel, či ladení hudobných nástrojov. Estimácia základnej frekvencie F_0 vo zvukovom signáli je často používanou metódou určovania výšky tónu. Sú známe výnimky, kde F_0 (a prípadne aj ďalšie alikvotné tóny) v signáli chýba a napriek tomu môže ľudský mozog vnímať zvuk ako tón s výškou F_0, ktorá býva typicky najväčším spoločným deliteľom zvyšných harmonických frekvencií [1]. Tento jav sa zvykne nazývať chýbajúci základný tón a využíva sa napríklad v telefónii [2, s. 125]. Sú známe aj iné špeciálne prípady, keď výška vnímaného tónu nezodpovedá F_0 [1]. Vo väčšine prípadov sú však F_0 a výška tónu zameniteľné hodnoty. Vďaka tomu sa pre algoritmy určujúce F_0 ustálil názov „pitch detection algorithms“ (skrátene PDA, algoritmy detegujúce výšku tónu) [3].

Napriek pokračujúcemu výskumu neexistuje exaktný algoritmus pre výpočet F_0 z digitalizovaného zvuku a žiadny PDA nepodáva 100% spoľahlivé výsledky. Okrem rôznych alikvotných tónov v signále (či už harmonických násobkov, alebo nie) komplikuje výpočet polyfónia a šum. Jedným z hlavných rozdielov medzi použitím v inštrumentálnej hudbe a rozpoznávaním reči je hľadaný rozsah F_0. Pre ľudský hlas sa môže F_0 pohybovať v rozmedzí zhruba od 77 Hz po 634 Hz, zatiaľ čo typický rozsah 88 klávesového klavíra (ako zástupcu bežného akustického nástroja s jedným z najväčších rozsahov) je pri použití rovnomerne temperovaného ladenia v rozmedzí od 27,5 Hz až po 4186,01 Hz [4]. Takýto veľký rozsah hľadaných frekvencií napomáha k vzniku chybných odhadov. Takisto je treba počítať s aliasingom, ktorý sa dá minimalizovať zvyšujúcou sa vzorkovacou frekvenciou. Interpolačné navýšenie vzorkovacej frekvencie už digitalizovaného zvuku sa nazýva upsampling a pre mnohé PDA môže priniesť spresnenie výsledkov. Keďže však upsampling iba zjemňuje rozlíšenie už vzorkovaného signálu, na frekvenčnú hranicu digitalizovaného signálu, zodpovedajúcej Nyquistovmu teorému, má vplyv iba prvotné vzorkovanie. Na zlepšenie úspešnosti algoritmov sa tiež často používajú rôzne postprocessing algoritmy ako napr. mediánové vyhladzovanie. Takisto je pre mnohé algoritmy výhodné prefiltrovať signál low-pass resp. high-pass filtrom (lineárny filter, ktorý neprepustí signál vyšších resp. nižších frekvencií, než na akú je nastavený). Týmto metódam sa však v tejto práci nevenujem.

Neexistuje žiadne oficiálne delenie algoritmov pre detekciu F_0 a tak je možné stretnúť sa s rôznymi protichodnými názormi. Väčšina algoritmov sa však dá rozdeliť do troch skupín [5, 6, 7]. Prvá skupina operuje nad časovou doménou signálu. Tieto algoritmy odhadujú periódu kváziperiodického signálu, z čoho následným invertovaním získavajú frekvenčný odhad. Zvyčajne nie sú vhodné pre analýzu polyfónie. Druhá skupina operuje nad frekvenčnou doménou signálu. Tieto algoritmy pracujú s frekvenčným spektrom vytvoreným zvyčajne krátkodobou Fourierovou transformáciou. Do tretej skupiny môžeme zaradiť ostatné algoritmy, ktoré môžu pracovať buď na inom princípe, alebo ako kombinácia predchádzajúcich dvoch prístupov. To ponúka možnosť potlačiť chybu náchylnú na vznik pri jednom prístupe druhým prístupom.

Algoritmus YIN spadá do prvej kategórie, keďže teoreticky stavia na autokorelácii. Pridaním ďalších krokov však znižuje chybovosť algoritmu a zvyšuje jeho presnosť. YIN je možné implementovať dostatočne výpočtovo nenáročne pre použitie v reálnom čase, nekladie frekvenčný strop pre F_0, a tak je dobrým kandidátom pre použitie v hudobnom prostredí. Aj keď je jeho autormi predstavený variant pre analýzu polyfonického signálu [3], v tejto práci sa zaoberám implementáciou a overením funkčnosti na hudobných nahrávkach základnej varianty, ktorá považuje signál za monofonický, t.j. hľadá v jednom časovom okamihu iba jednu základnú frekvenciu.

2 Prehľad algoritmov

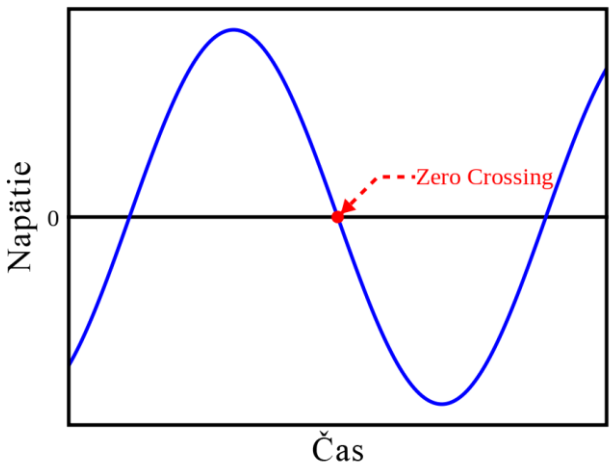
V tomto prehľade uvediem niektoré najznámejšie algoritmy detegujúce výšku tónu.

2.1 Algoritmy operujúce nad časovou doménou

Jedna skupina týchto algoritmov vyhľadáva vo vlnovom priebehu signálu určité charakteristické znaky a na základe časových rozostupov ich výskytu určí periódu. Druhá skupina vychádza z autokorelácie, alebo ju priamo využíva.

2.1.1 ZCR – Zero Crossing Rate

Meranie frekvencie pretnutia nuly je pravdepodobne najjednoduchší spôsob detekcie frekvencie. Vo vlnovom priebehu hľadá miesta, kde signál pretne nulovú hodnotu, t.j. kde sa signál mení zo záporného na kladný a naopak (Obrázok 1). Sínusoida pretne nulu dvakrát za periódu, a tak vzdialenosť medzi troma pretnutiami je prehlásená za periódu. Takýto odhad samozrejme funguje iba pri jednoduchom monofonickom signáli a zlyháva pri chýbajúcom základnom tóne, či signáli obsahujúcom silné harmonické tóny, nakoľko môžu spôsobiť viacero pretnutí nuly za periódu. Keďže je ale algoritmus jednoduchý, dá sa implementovať veľmi efektívne. Zvykne sa používať pri analýze reči, konkrétne pri rozhodovaní, či je spoluhláska znelá alebo neznelá. Ukázalo sa totiž, že frekvencie získané pomocou ZCR sú nízke pri znelých spoluhláskach a vysoké pri neznelých [8].



Obrázok 1 Vizuálne znázornenie signálu pretínajúceho nulu [9]

2.1.2 ACF – Autocorrelation function

Autokorelácia je vzájomná korelácia signálu so sebou samým v určitých časových posunoch. Dá sa povedať, že jej výsledkom je miera podobnosti signálu ako funkcia

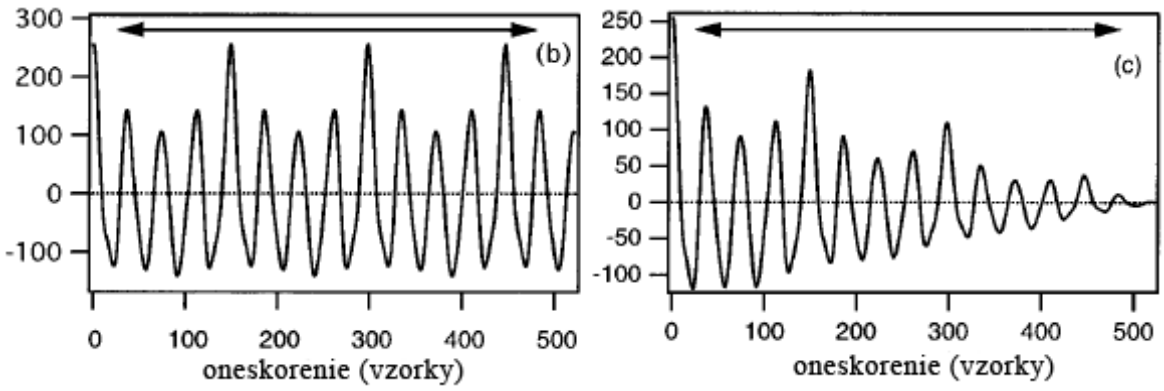
časového oneskorenia dvoch rovnakých vlnových priebehov. Je zrejmé, že najvyššia miera podobnosti sa nachádza na nulovom posune a pri periodickom signáli ďalšie maximá vykazuje na násobkoch periódy [3]. V praxi sa používa tzv. krátkodobá autokorelačná funkcia, keďže sa neočakáva, že jedna základná frekvencia bude hraná počas celej nahrávky. Táto krátkodobá ACF sa počíta iba nad krátkou sekvenciu dát, označovanou ako okno (W - window). Veľkosť tohto okna sa väčšinou volí dostatočne malá na to, aby spĺňalo predpoklad, že v ňom ostáva F_0 približne stabilné. Z toho vyplýva, že pre rýchlo meniace sa frekvencie môže byť odhad v danom okne nepresný. Na druhú stranu, dĺžka okna by mala byť dostatočne dlhá, aby obsiahlo aspoň dve najdlhšie očakávané periódy [3, 5]. Matematické vyjadrenie autokorelačnej funkcie diskrétneho signálu v danom čase je zobrazené v rovnici (1), kde *t* je časový index, τ je časový posun, W veľkosť integračného okna a x_j je hodnota diskrétneho signálu v čase *j* [5].

$$r_t(\tau) = \sum_{j=t+1}^{\frac{W}{2}-1} x_j x_{j+\tau}, \qquad 0 \le \tau \le W/2 \tag{1}$$

Pri spracovaní signálu je však bežné používať mierne odlišnú definíciu [3, 5]:

$$r'_t(\tau) = \sum_{j=t+1}^{t+W-\tau} x_j x_{j+\tau}, \qquad 0 \le \tau < W \tag{2}$$

Pri použití definície v rovnici (2) sa počet sčítavaných členov zmenšuje so zväčšujúcim sa τ. To má na výsledok autokorelácie zužujúci efekt, t.j. hodnoty majú sklon lineárne klesať až k nule pri $\tau = W$(Obrázok 2).



Obrázok 2 Príklad výsledku autokorelácie za použitia rovnice (1) (b), a rovnice (2) (c) [3]

Po použití autokorelácie ostáva na určenie periódy iba vybrať správne lokálne maximum. Tu nastávajú dve možnosti. Prvou je, že sa prehľadáva celý rozsah τ a vyberie sa hodnota τ

v globálnom maxime. Tu však nastáva problém – ak sa neurčí spodný limit pre τ, prípadne je veľmi blízko nule, algoritmus môže vybrať maximum z prvotného vrchola v $\tau = 0$. A naopak, ak je horný limit príliš veľký, môže sa chybne vybrať lokálne maximum vyššieho rádu (ktoré vznikne na celočíselných násobkoch periódy) [3]. Druhá možnosť je zvoliť prvé lokálne maximum, keďže sa pri jednoduchých zvukoch často stáva, že je to práve hľadaná perióda [5]. Násobenie veľkého počtu hodnôt môže byť výpočtovo náročné, a tak často používaný spôsob implementácie autokorelácie je pomocou FFT (Fast Fourier Transformation), algoritmu pre výpočet diskrétnej Fourierovej transformácie a jej inverzie.

2.1.3 AMDF – Average Magnitude Difference Function

AMDF je podobná autokorelácii. Jej definícia je zobrazená v rovnici (3) [5].

$$\gamma(\tau) = \frac{1}{W}\sum_{j=0}^{\frac{W}{2}-1}\left|x_j - x_{j+\tau}\right|, \qquad 0 \le \tau \le W/2 \tag{3}$$

Stavia na predpoklade, že ak je signál pseudoperiodický, tak si sú dve susedné periódy vlnového priebehu podobné tvarom. Ak sa odčíta od každej hodnoty signálu tá istá hodnota, výslednou sumou bude 0. Keďže niektoré hodnoty sú záporné a niektoré kladné, sčítavajú sa absolútne hodnoty rozdielov, aby narastajúci súčet odrážal zväčšujúci sa rozdiel v porovnávaných vlnových priebehoch. Z algoritmu sa teda výsledná perióda určí výberom takého τ, pre ktoré je hodnota funkcie najmenšia. Pre zrýchlenie behu na špecializovanom úspornom HW, pre ktorý robí násobenie problém, sa občas vynecháva $\frac{1}{W}$ [5].

2.2 Algoritmy operujúce nad frekvenčnou doménou

Algoritmy pracujúce nad frekvenčnou doménou sú často schopnejšie analyzovať polyfonický signál. Neanalyzujú vlastnosti pôvodného signálu v čase, ale transformujú ho do frekvenčnej domény (vytvárajú spektrum). Na to používajú diskrétnu Fourierovu transformáciu. Pôvodný signál je opäť rozdelený do časových okien W, ktoré sa ale väčšinou čiastočne prekrývajú. Matematické vyjadrenie DFT na časovom okne W:

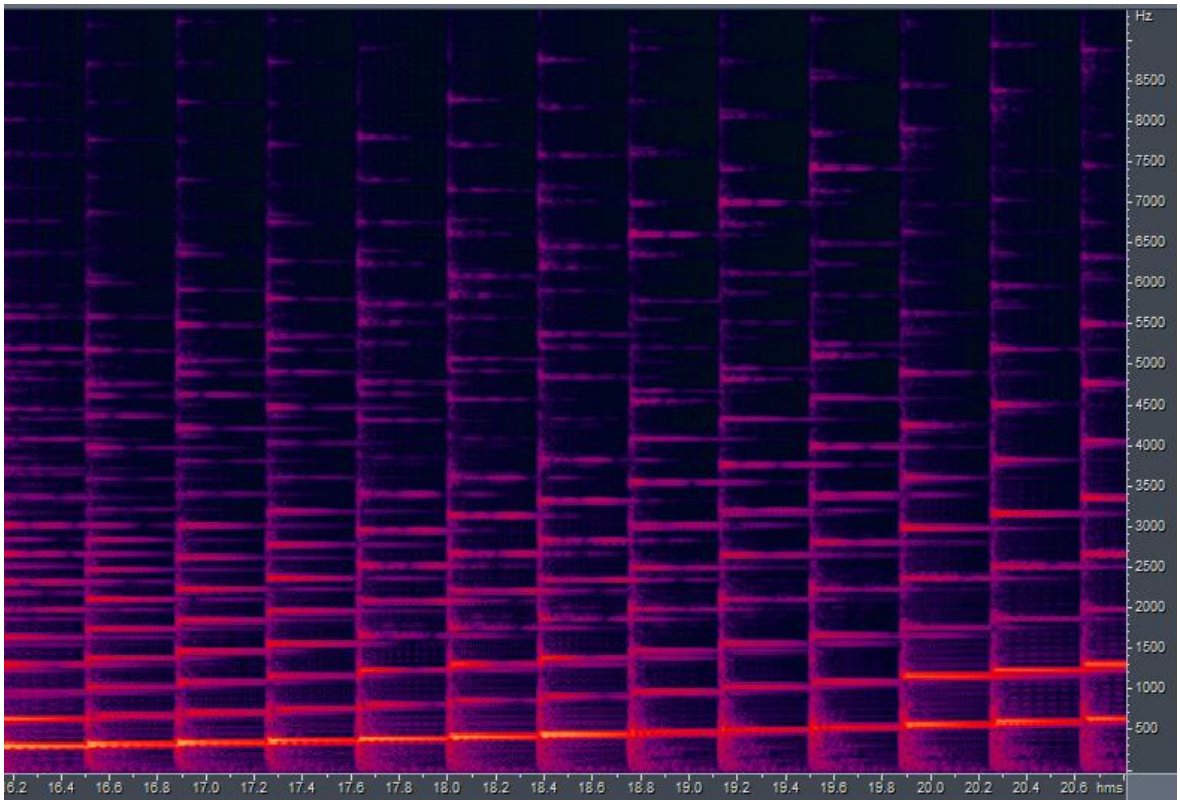
$$X_f = \sum_{t=0}^{W-1} x_t e^{-\frac{2\pi i f t}{W}} \tag{4}$$

, kde Xf sú Fourierove koeficienty v komplexnom tvare. Pre výpočet DFT existuje však efektívnejší algoritmus, nazývaný FFT (Fast Fourier Transform, rýchla Fourierova

transformácia), ktorý sa používa v analýze signálu. FFT znižuje výpočtovú zložitosť DFT z $O(W^2)$ na $O\big(W\ log(W)\big)$ [5]. Pôvodná sekvencia dát sa dá z Fourierovych koeficientov získať pomocou inverznej DFT:

$$x_t = \frac{1}{W}\sum_{f=0}^{W-1} X_f e^{\frac{2\pi i f t}{W}} \tag{5}$$

Z výsledkov FFT sa dá vytvoriť tzv. spektrogram, t.j. vizuálne zobrazenie frekvenčných zložiek signálu v čase. Pri 2D zobrazení sa zvyčajne vykresľuje čas na horizontálnu os, frekvencie na vertikálnu os a farba, prípadne jas predstavuje amplitúdu danej frekvencie. Príklad spektrogramu vytvorený z nahrávky vzostupnej chromatickej stupnice hranej na klavíri v programe Cool Edit Pro je na obrázku (Obrázok 3).

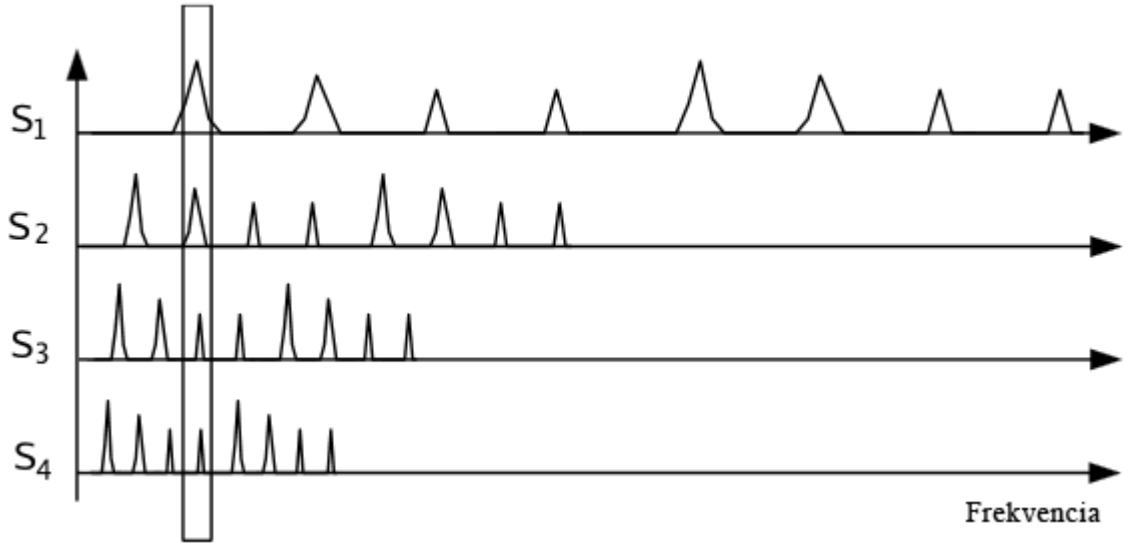


Obrázok 3 Príklad spektrogramu (úsek vybraný zo vzostupnej chromatickej stupnice hranej na klavíri)

Najjednoduchší algoritmus pre estimáciu F_0 za použitia DFT teda môžeme zostrojiť tak, že v čase *t* vyberieme takú hodnotu frekvencie, ktorá má najväčšiu amplitúdu. To je ale veľmi naivný algoritmus, pretože sa často stáva, že dominantná frekvencia nie je F_0, ale nejaký harmonický tón. Tento prístup samozrejme zlyhá tiež v prípade chýbajúceho základného tónu.

2.2.1 HPS – Harmonic Product Spectrum

Použitie HPS vychádza z predpokladu, že signál obsahuje viaceré harmonické zložky, umiestnené na celočíselných násobkoch F_0. Tým pádom ak signál downsamplujeme celočíselným koeficientom *i*, tak sa *i*-ty harmonický tón zarovná s F_0 pôvodného signálu. Algoritmus vyzerá nasledovne: Najskôr vypočítame spektrum S_1 pôvodného signálu (napr. pomocou FFT). Potom S_1 downsamplujeme koeficientom 2, čím dostaneme spektrum S_2. Druhý harmonický tón by sa tým mal zarovnať s F_0. Následne vytvoríme spektrum S_3 downsamplovaním S_1 koeficientom 3, čím by sa mal zarovnať tretí harmonický tón s F_0. Takto postupne vytvárame spektrá S_i, až dokým *i* nie je rovné počtu harmonických tónov, ktoré chceme zahrnúť do porovnania. Výsledné spektrá sa medzi sebou vynásobia a za F_0 sa prehlási frekvencia s najvyššou hodnotou amplitúdy [5]. Ilustrovaný postup je na obrázku (Obrázok 4).



Obrázok 4 Príklad frekvenčného spektra S_1 a jeho (downsamplingom) stlačených verzií. Obdĺžnik ohraničuje lokálne maximá zarovnané s F_0 [5].

Jednou z limitácií HPS je, že nefunguje dobre pri použití malých okien W, ktoré obsahujú iba 2 či 3 periódy signálu. Takže je limitovaná rozlíšením frekvencií spektra získanom pomocou DFT, kde sa kvôli diskrétnosti rozlíšenia môžu stratiť lokálne maximá. Zväčšovanie dĺžky okna pre výpočet krátkodobej Fourierovej transformácie, vďaka čomu môžu byť lokálne maximá lepšie separované vylepšuje úspešnosť algoritmu. Dôsledkom zväčšenia okna W je však zníženie časového rozlíšenia [5]. Pomocou HPS sme sa tak elegantne vyhli problému pôvodného naivného algoritmu, ktorý vyberal frekvenciu s najväčšou amplitúdou (ktorá nemusí byť F_0). HPS však samozrejme zlyhá v prípade čistého tónu bez harmonických zložiek.

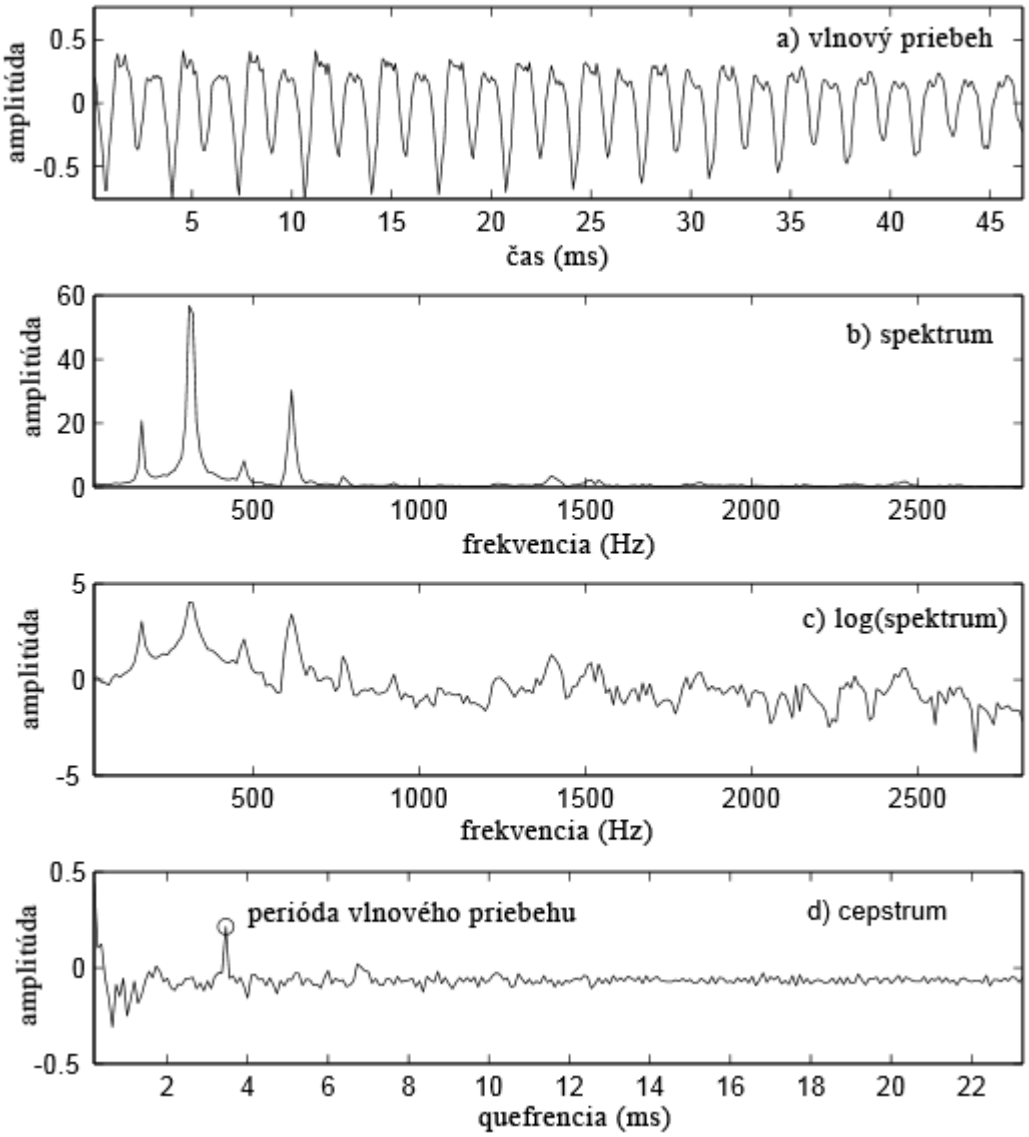
2.2.2 Cepstrálna analýza

Slovo cepstrum vzniklo ako prešmyčka slova „spectrum“. Podobne, nezávislá premenná pri cepstrálnej analýze sa nazýva „quefrencia“ [7]. Cepstrum je vo všeobecnosti definované ako inverzná Fourierova transformácia logaritmického spektra signálu [6]. Okrem mocninového cepstra existuje aj napr. komplexné, či reálne cepstrum. V signálnej analýze sa však častejšie využíva mocninové, ktoré sa od pôvodnej definície líši iba minimálne [10]. Postup na jeho vytvorenie je nasledovný [11]:

$$x_{pc}(t) = (F^{-1}\{log(|F\{x(t)\}|^2)\})^2 \quad \textbf{(6)}$$

, kde F^{-1} je inverzná Fourierova transformácia.

Základnou myšlienkou tejto metódy je fakt, že vo výsledku Fourierovej transformácie sa nachádzajú pravidelne rozložené lokálne maximá, vzniknuté z harmonických tónov v spektre. Keď sa spektrum zlogaritmuje, hodnoty týchto maxím sa znížia a dostaneme vlnový priebeh vo frekvenčnej doméne. Ako posledný krok použijeme inverznú Fourierovu transformáciu, ktorej globálne maximum bude podobne ako pri autokorelácii zodpovedať perióde rozloženia frekvenčných zložiek zastúpených v spektre [7]. Ilustrovaný postup je na obrázku (Obrázok 5).



Obrázok 5 Ilustrovaný postup pri použití cepstrálnej analýzy [7]

Táto metóda paradoxne zlyháva pri jednoduchých tónoch s iba jednou zložkou – keďže sa v spektre nevyskytujú ďalšie lokálne maximá, nie je z čoho odhadnúť periódu. Takisto, ak v signáli nie sú pravidelne rozmiestnené harmonické tóny, metóda vráti zlý odhad. Pôvodne bola vyvinutá pre analýzu reči, ktorá je spektrálne bohatá a má pravidelne rozmiestnené alikvotné tóny [7].

3 Algoritmus YIN

Meno algoritmu YIN pochádza z orientálnej filozofie (koncepty Yin a Yang). Jeho autormi sú Alain de Cheveigné a Hideki Kawahara. Je teoreticky založený na autokorelácii s niekoľkými modifikáciami, ktoré zvyšujú jeho úspešnosť. Nemá horný limit pre hľadaný frekvenčný rozsah, dá sa implementovať dostatočne efektívne pre použitie v reálnom čase. Navyše autori predložili niekoľko variantov, pre rôzne druhy odchýlky signálu od periodického modelu. Tie však prinášajú zlepšenie iba pre špecifické druhy signálov a naopak pre väčšinu signálov zväčšujú početnosť chýb [3].

Základným kameňom algoritmu je diferenčná funkcia. V ďalšom kroku algoritmu sa funkcia normalizuje, a potom sa na nej hľadá lokálne minimum zodpovedajúce perióde F_0. Minimá sa kvôli spresneniu odhadu interpolujú parabolou a nakoniec algoritmus prehľadáva okolie bodu prvotnej analýzy pre „kvalitnejší“ odhad. Tieto kroky sú bližšie popísané v nasledovných podkapitolách.

3.1 Diferenčná funkcia

Autormi navrhovaná diferenčná funkcia sa veľmi podobá AMDF. Tiež stavia na predpoklade že odčítaním hodnôt periodického signálu vzdialených o celočíselný násobok periódy dostaneme nulu. Na rozdiel od AMDF sa však záporným sčítancom vyhne umocnením na druhú :

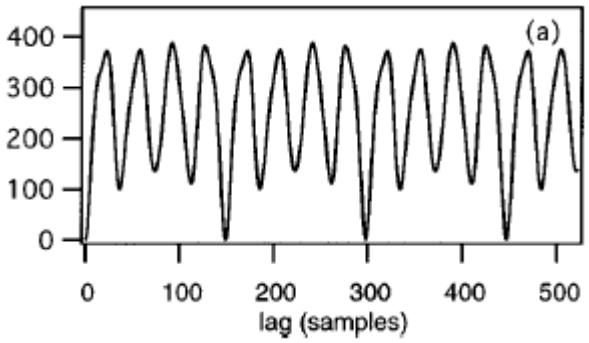
$$d_t(\tau) = \sum_{j=t+1}^{t+\frac{W}{2}} \left(x_j - x_{j+\tau}\right)^2 \quad (7)$$

Implementácia podľa tohto vzorca však môže byť výpočtovo dosť náročná. Existuje však riešenie – rozvinutím sumy sa dá funkcia vyjadriť pomocou autokorelácie definovanej v rovnici (1), ktorá sa dá rýchlo vypočítať pomocou FFT:

$$d_t(\tau) = r_t(0) + r_{t+\tau}(0) - 2r_t(\tau) \quad (8)$$

Hlavnou výhodou tejto diferenčnej funkcie oproti autokorelácii je fakt, že autokorelácia je veľmi citlivá na zmeny amplitúdy (pre za sebou nasledujúce periódy signálu). Napr. pri zväčšujúcej sa amplitúde sa zväčšujú aj amplitúdy ACF, čo pri výbere globálneho maxima znamená, že sa chybne vyberie dlhšia perióda. Pri stabilných amplitúdach signálu však ostávajú amplitúdy ACF konštantné a pri zmenšujúcej sa amplitúde sa výsledné amplitúdy

zmenšujú. Naproti tomu pri použití diferenčnej funkcie sa prejav rozdielnosti periód zvyšuje vo všetkých prípadoch zmeny amplitúdy [3]. Keďže však pri diferenčnej funkcii hľadáme τ, pre ktoré je funkcia nulová (v ideálnom prípade, inak hľadáme globálne minimum), jej použitie má sklon zvoliť jedno z prvých lokálnych miním (kratšiu periódu). Príklad diferenčnej funkcie je na obrázku (Obrázok 6).



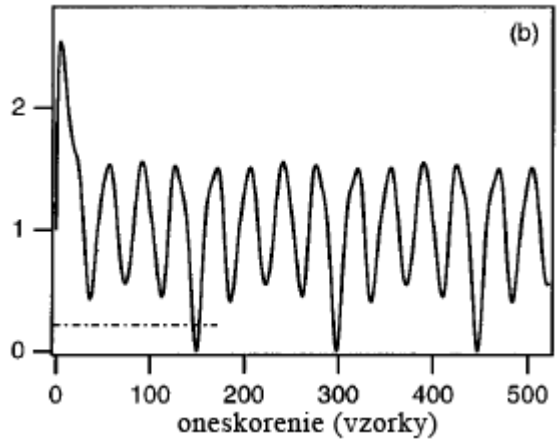
Obrázok 6 Príklad diferenčnej funkcie [3]

3.2 Diferenčná funkcia normalizovaná kumulatívnym kĺzavým priemerom

CMNDF (Cumulative mean normalized difference function) stavia na predchádzajúcej diferenčnej funkcii. Tá je vždy nulová pre $\tau = 0$ a často nenulová na násobkoch periódy, kvôli nedokonalej periodicite. Rovnako ako pri autokorelácii môžeme zvoliť spodný limit pre hľadané τ, ale to nie je najlepšie riešenie. Môže sa totiž stať že silná rezonancia alikvotných tónov môže vytvoriť sériu druhotných lokálnych miním, z ktorých jedno môže byť menšie než lokálne minimum reprezentujúce periódu F_0 [3]. Ako riešenie z pôvodnej diferenčnej funkcie vytvoríme normalizovanú:

$$d'_t(\tau) = \begin{cases} 1, & pre\ \tau = 0 \\ \dfrac{d_t(\tau)}{\left(\frac{1}{\tau}\right)\sum_{j=1}^{\tau} d_t(j)}, & inak \end{cases} \quad \textbf{(9)}$$

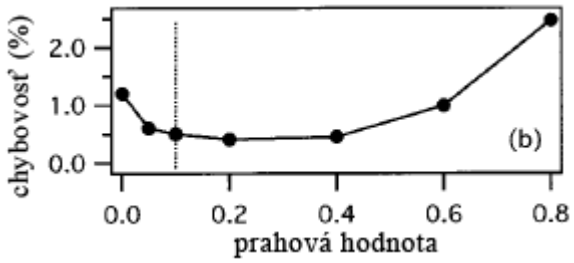
Táto nová funkcia je výsledkom vydelenia každej hodnoty diferenčnej funkcie priemerom predchádzajúcich hodnôt. Začína s hodnotou 1 namiesto 0, má tendenciu ostať spočiatku na relatívne vysokých hodnotách a spadne pod 1 iba tam, kde je hodnota diferenčnej funkcie menšia než priemer (Obrázok 7). Pre takto normalizovanú funkciu už nemusíme voliť spodný limit pre τ (tým pádom odpadol aj vrchný frekvenčný limit).



Obrázok 7 Príklad normalizovanej funkcie [3]

3.3 Prahová hodnota

Tento krok vlastne popisuje výber lokálneho minima pre výslednú periódu. Keďže sa môže stať, že niektoré z neskorších lokálnych miním je menšie než samotné minimum zodpovedajúce perióde F_0, pri výbere globálneho minima by sme opäť vybrali nesprávnu hodnotu. Táto chyba pri autokorelačných algoritmoch sa niekedy nazýva oktávová chyba, kvôli tomu, že väčšinou je vybrané lokálne minimum zodpovedajúce násobku periódy F_0 a nejakej celočíselnej mocniny čísla 2 (t.j. výsledný tón sa bude líšiť oktávou, napr. A4 vs. A5). Ako riešenie je zvolená nejaká prahová hodnota (threshold). Namiesto globálneho minima CMNDF sa algoritmus snaží vybrať také prvé lokálne minimum, ktoré má hodnotu $d'(\tau)$ menšiu než zvolený prah. Ak také neexistuje, zvolí sa globálne minimum.



Obrázok 8 Porovnanie chybovosti algoritmu v závislosti od prahovej hodnoty [3]

Ako sa ukázalo, na samotnej hodnote tohto prahu veľmi nezáleží a teda si nevyžaduje jemné dolaďovanie (Obrázok 8). Preto by hodnota 0,1 mala byť dostatočne vhodnou pre široké použitie.

3.4 Parabolická interpolácia

Odhad periódy je presný, iba ak je perióda celočíselným násobkom vzorkovacej periódy, inak sa môže odhad líšiť až o polovicu vzorkovacej periódy. Navyše, lokálne minimum

spojitej funkcie môže byť vzorkovaním vynechané a to môže narušiť výber spomedzi lokálnych miním (či už v predchádzajúcom kroku pri porovnávaní voči prahovej hodnote, alebo v nasledujúcom kroku, pri porovnávaní medzi) [3]. Na zvýšenie presnosti frekvenčného odhadu sa niekedy používa upsampling signálu. Ten je však výpočtovo náročnejší než parabolická interpolácia lokálnych miním signálu.

Každé lokálne minimum CMNDF a jeho bezprostredne susedné body sú interpolované parabolou a ordinát vrchola tejto paraboly je použitý pri výbere lokálnych miním. Abscisa vybraného vrchola potom predstavuje odhad periódy. Tento odhad získaný z CMNDF je však jemne skreslený, preto sa použije abscisa získaná interpoláciou pôvodnej nenormalizovanej diferenčnej funkcie.

3.5 Najlepší lokálny odhad

Veľkosť okna W sa rovnako ako pri ACF volí dostatočne malá na to, aby spĺňalo predpoklad, že v ňom ostáva F_0 približne stabilné. Akékoľvek fluktuácie v danom časovom okne by nemali byť brané ako skutočné. Pre nestacionárne signály sa môže stať, že odhad zlyhá na určitej fáze periódy, čo sa kryje s relatívne vysokou hodnotou CMNDF pre $\tau = T$, kde T je odhad periódy. Na inej fáze môže byť odhad správny a hodnota CMNDF v T menšia [3]. Pre využitie tohto faktu algoritmus hľadá v okolí každého bodu analýzy vhodnejší odhad.

Pre každý časový index t (vo vzorkách) z intervalu $\langle t - \frac{T_{max}}{2}, t + \frac{T_{max}}{2} \rangle$ sa algoritmus vykoná znova s obmedzeným rozsahom hľadania periódy. Z výsledných odhadov na časových indexoch sa vyberie ten, pre ktorý je hodnota v danej CMNDF najmenšia. Hľadaný rozsah periódy sa určí ako percentuálna odchýlka od pôvodného odhadu v t (napr. $\pm 20\%$). Tento krok pripomína mediánové vyhladzovanie, avšak na rozdiel od neho nie je založený iba na kontinuite hodnôt, ale aj na kvalitatívnom ohodnotení odhadov, reprezentovaných hodnotou CMNDF.

4 Implementácia algoritmu YIN

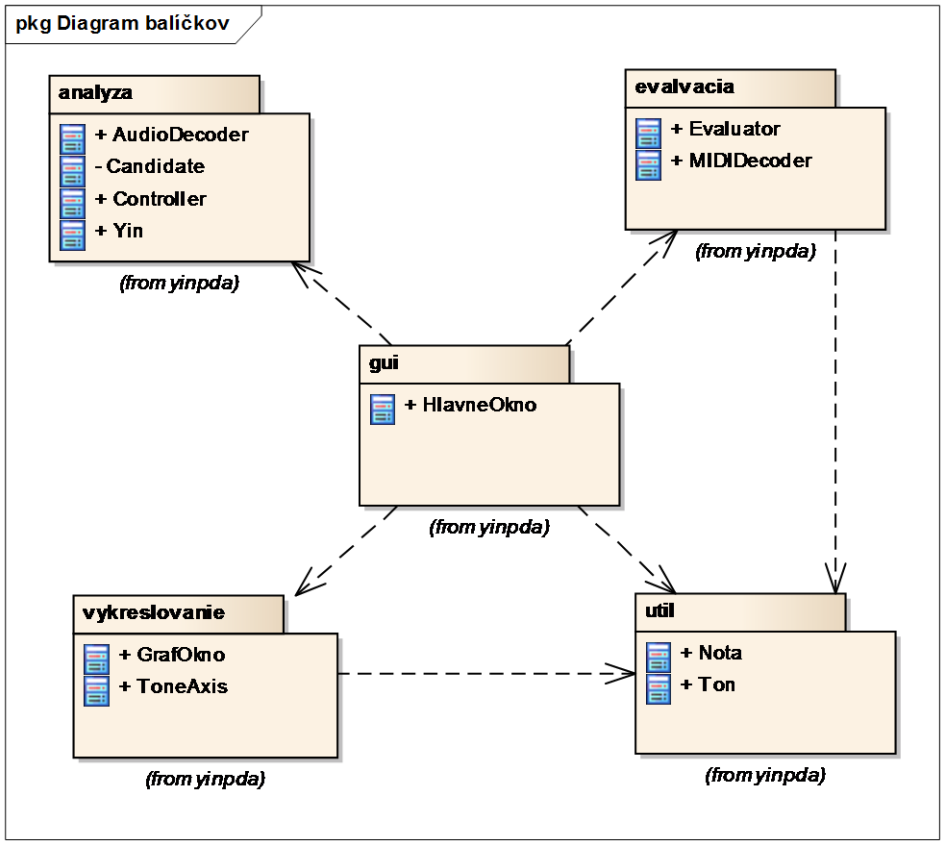
Ako implementačný jazyk som zvolil Javu SE 7, keďže ju ovládam najlepšie spomedzi ostatných vysokoúrovňových jazykov, má vstavanú podporu pre dekódovanie WAVE, AIFF, AU a SMF súborov a napriek bežne kolujúcim mýtom je Java (respektíve HotSpot) rýchlosťou porovnateľná s natívnym kódom. Pre vykresľovanie grafu som však zvolil open source knižnicu JFreeChart.

4.1 Architektúra aplikácie

Samotný návrh architektúry je jednoduchý. Fakt, že program pracuje v danom čase vždy iba s jednou zvukovou nahrávkou dovolil navrhnúť viacero tried staticky. Celý návrh programu veľmi zjednodušuje fakt, že je jednoúčelový, čo znižuje mieru abstrakcie. Napríklad algoritmus pre analýzu je iba jeden, čím odpadáva nutnosť použitia interfejsu, abstraktnej triedy alebo rovno strategy patternu. Tým pádom návrh obsahuje jednoduchšie a pevnejšie väzby medzi triedami. Takisto pri evalvácii výsledkov sa počíta iba s rozsahom nôt A0-C8 v 12 tónovom rovnomerne temperovanom ladení.

Triedy sú členené do piatich balíčkov podľa príbuzných funkcií (Obrázok 9), za ktoré sú zodpovedné (prístup package by feature). Sú to:

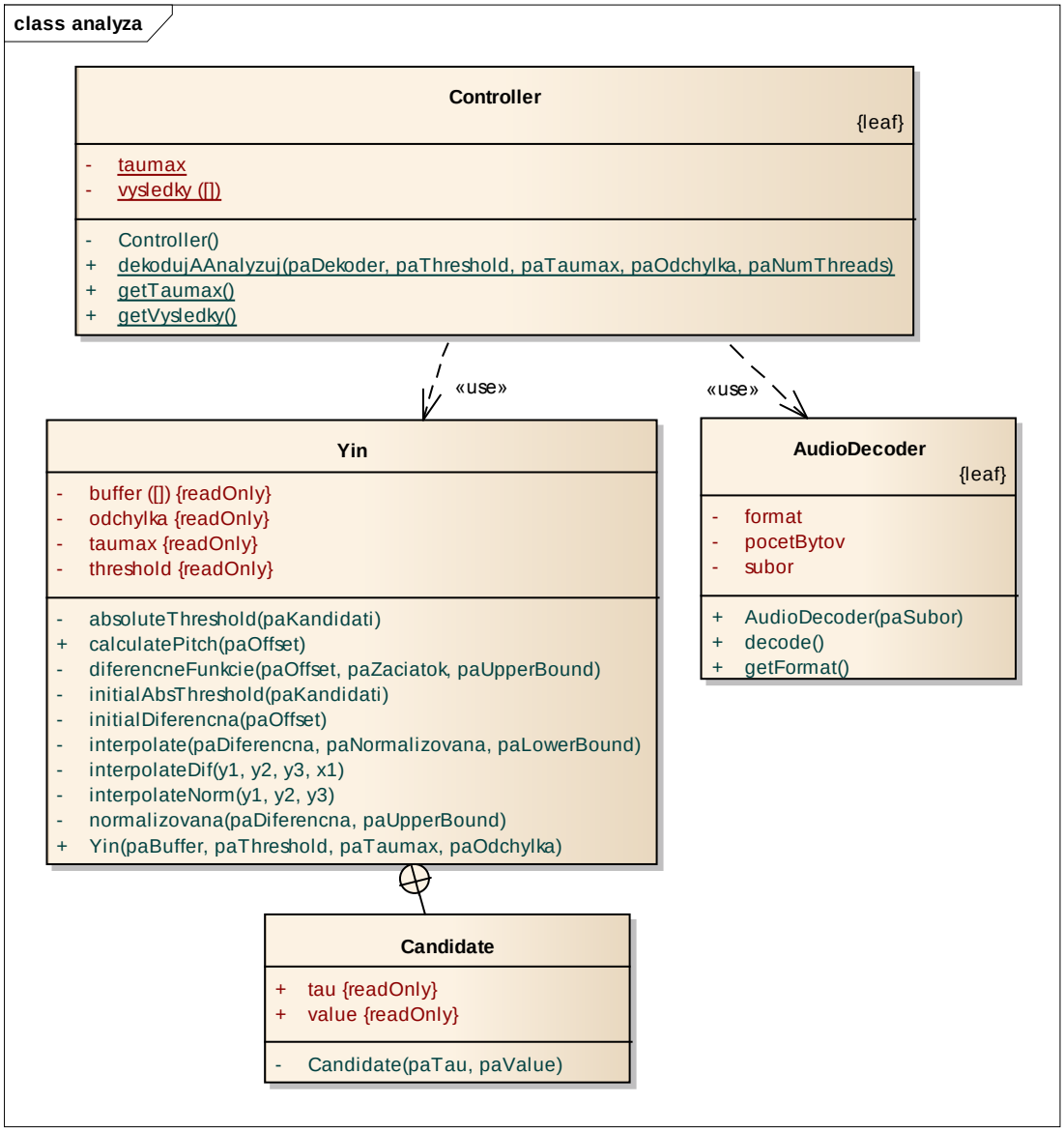
- yinpda.analyza – triedy, ktoré sa starajú o analýzu zvukového súboru
- yinpda.evalvacia – triedy, ktoré sa starajú o evalváciu výsledkov algoritmu
- yinpda.util – pomocné triedy, tzv. utility triedy
- yinpda.vykreslovanie – triedy, ktoré sa starajú o vykresľovanie grafu
- yinpda.gui – hlavné GUI aplikácie



Obrázok 9 Diagram balíčkov tried aplikácie

4.1.1 Balíček yinpda.analyza

Tento balíček je zodpovedný za dekódovanie zvukového súboru a jeho analýzu algoritmom YIN. Pre dekódovanie súboru do poľa vzoriek slúži trieda AudioDecoder a jej metóda decode(). Každá jej inštancia má v sebe atribút predstavujúci spracovávaný súbor. Inštancia sa však vytvára už pri „otvorení“ audio súboru v aplikácii, pričom sa skontroluje jeho validita a zistí sa jeho formát. Túto inštanciu potom používa trieda Controller, ktorá ju prijíma ako parameter pri spustení analýzy. Controller najskôr zavolá decode() na danom objekte. Potom zo zadaných parametrov a dekódovaných vzoriek vytvorí inštanciu triedy Yin, v ktorej prebieha samotný odhad periódy v danom čase. Controller sa zároveň stará o paralelné rozdelenie práce algoritmu na vzorkách zo súboru. Candidate je vnútorná trieda triedy YIN, predstavujúca dočasných kandidátov na výslednú periódu počas analýzy. Keďže aplikácia pracuje v danom čase vždy iba s jednou zvukovou nahrávkou, Controller je implementovaný ako statická trieda. Diagram tried bez špecifikovaných dátových typov je na obrázku (Obrázok 10).



Obrázok 10 Diagram tried balíčka yinpda.analyza bez uvedených dátových typov

4.1.2 Balíček yinpda.evalvacia

Balíček evalvacia je zodpovedný za vyhodnotenie úspešnosti algoritmu. Tú vyhodnocuje na základe porovnania výsledkov s údajmi získanými z MIDI súboru. Obe jeho triedy (Evaluator aj MIDIDecoder) sú implementované opäť ako statické triedy, keďže sa výsledky porovnávajú v danom čase vždy iba s jedným súborom. MIDIDecoder má iba jednu metódu decode(File), ktorá zadaný SMF súbor dekóduje a nájde v ňom noty, ktoré potom vráti. S tými potom pracuje Evaluator, ktorého jediná metóda evaluate vráti percentuálnu zhodu výsledkov.

4.1.3 Balíček yinpda.util

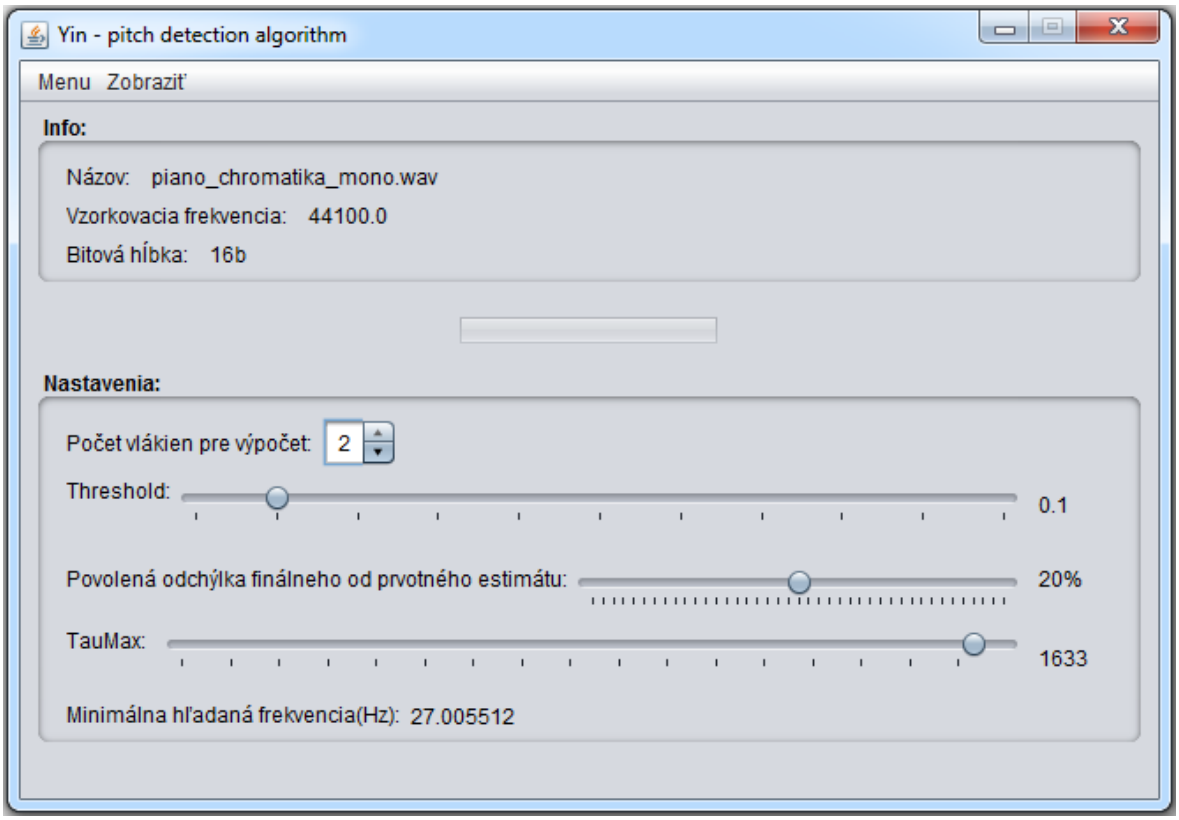
V tomto balíčku sa nachádzajú pomocné triedy, ktoré sú užitočné pre viacero balíčkov. Prvou z nich je enumerácia Ton, ktorá predstavuje hudobný tón. Každý tón má svoj názov a F_0. Druhou triedou je Nota, ktorá vlastne obaľuje danú inštanciu tónu. Jej inštancia totiž predstavuje daný tón s udanými časovými značkami pre jej začiatok a koniec.

4.1.4 Balíček yinpda.vykreslovanie

Triedy balíčka vykreslovanie sa starajú o zobrazovanie grafu. ToneAxis je potomkom triedy NumberAxis z knižnice JFreeChart a predstavuje os grafu s tónmi z enumerácie Ton. GrafOkno je reprezentáciou GUI okna, ktoré obsahuje vykreslený graf, prepínače medzi zobrazením frekvenčnej a tónovej osi a prípadne výsledok evalvácie.

4.1.5 Balíček yinpda.gui

V tomto balíčku sa nachádza iba jedna trieda HlavneOkno, ktorá predstavuje hlavné okno aplikácie, kde užívateľ otvára súbory, nastavuje parametre algoritmu, spúšťa analýzu atď. Príklad zobrazeného hlavného okna je na obrázku (Obrázok 11).



Obrázok 11 Príklad zobrazeného hlavného okna aplikáce s otvoreným súborom.

4.2 Trieda Yin

Samotný algoritmus je implementovaný ako klasická trieda Yin. Na rozdiel od použitia singletonu či statickej triedy, tak môže Garbage Collector po skončení analýzy automaticky uvoľniť miesto alokované inštanciou Yin (hlavne dekódovanými vzorkami). Užívateľ si volí prahovú hodnotu (Threshold), povolenú odchýlku výsledného od prvotného odhadu v percentách a hodnotu τ_{max}(Taumax). Od pomeru τ_{max} voči vzorkovacej frekvencii sa logicky odvíja aj spodný limit pre hľadanú frekvenciu. Pre YIN je však veľkosť okna W nezávislá od hodnoty τ_{max}. Napriek tomu ostáva platiť pravidlo, že W by malo obsiahnuť dvojnásobok najdlhšej hľadanej periódy. Síce W môže byť dlhšie, ale to znamená menšie časové rozlíšenie výsledkov [3]. Preto som sa rozhodol, že W bude vždy dvojnásobok hodnoty τ_{max}.

Keďže výpočet funkcie definovanej v rovnici (7) pre každý časový index a pre každé τ je výpočtovo náročný, autori predstavili dve riešenia [3]. Jedno spočíva v použití rovnice (8) za pomoci FFT. Druhé riešenie je implementovať rovnicu (7) rekurzívnym spôsobom, kde sa v každom nasledujúcom časovom indexe odpočíta starý člen a pripočíta nový. Ja som zvolil druhý spôsob.

4.2.1 Voľba dátových typov

Pre uloženie dekódovaných vzoriek som použil pole typu short. Short je v Jave 16 bitové číslo so znamienkom. To pokrýva väčšinu bežných hudobných nahrávok, nakoľko CDDA, WAVE, AIFF a AU kódujú signál najmä v 16 bitovom lineárnom PCM formáte so znamienkom. Pri výpočte diferenčných funkcií som však musel použiť typ long (64b), nakoľko ich hodnoty môžu presiahnuť rozsah integeru a v extrémnych prípadoch si vyžadovať až 47 bitov. To sa môže stať ak v diferenčnej funkcii sčítame najskôr dve vysoké hodnoty typu short ($2^{16}+2^{16} = 2^{17}$). Potom sa medzivýsledok umocňuje na druhú (2^{34}) a nakoniec sa tieto hodnoty sčítavajú do výslednej sumy. Počet týchto sčítancov závisí od hodnoty τ_{max} a jej hranica prístupná užívateľovi sa zase odvíja od vzorkovacej frekvencie nahrávky. Táto hranica sa určuje ako $\frac{1}{26}$ zo vzorkovacej frekvencie (rozsah hľadanej frekvencie je zdola obmedzený na 26 Hz). Pre vzorkovaciu frekvenciu 192 kHz je tak τ_{max} najviac 7384 (2^{13}) a výsledná suma má 47 bitov ($2^{34} * 2^{13}$). Prvé delenie a teda aj možný výskyt desatinných čísiel sa vyskytuje až pri výpočte normalizovaných funkcií. Preto pre ne používam polia typu double. Pre výber finálneho odhadu z kandidátov ich dočasne ukladám do LinkedListu, pretože dopredu nie je známe koľko ich bude

a spracovávam ich vždy iba sekvenčne. Nakoľko má samotný kandidát okrem svojej polohy voči offsetu aj kvalitatívne ohodnotenie, vytvoril som pre ne v algoritme vnútornú triedu Candidate s dvoma atribútmi – tau a value.

4.2.2 Inicializácia

Pri vytvorení inštancie sa do atribútu buffer uloží celý dekódovaný súbor. Vstupným bodom je metóda calculatePitch(paOffset), ktorá vráti výslednú periódu na okne vzdialenom od začiatku nahrávky o zadaný offset udávaný v počte vzoriek. Hneď na začiatku bolo treba vyriešiť ako sa zachovať pri výpočte odhadu na úplnom začiatku nahrávky, keď prvá polovica intervalu $\langle t - \frac{T_{max}}{2}, t + \frac{T_{max}}{2} \rangle$ neexistuje. Keďže by posunutie offsetu znamenalo založiť prvotný odhad na inom bode než skutočnom začiatku nahrávky, zvolil som možnosť, že pre tento prvý odhad periódy sa počíta iba druhá polovica intervalu. Podobný problém vznikol aj pri konci nahrávky. Ak by sa okno, ktorého počiatok je z daného intervalu nezmestilo do konca nahrávky, rozhodol som sa offset výpočtu posunúť späť tak, aby sa posledné okno intervalu zarovnalo s koncom.

4.2.3 Prvotný odhad

Pre prvotný odhad prebieha hľadanie v celom rozsahu $1 - \tau_{max}$ a nepočítajú sa funkcie pre okolité časové indexy. Preto bolo prehľadnejšie vytvoriť preň dve osobitné metódy vytvárajúce diferenčnú funkciu a voľbu kandidáta – initialDiferencna a initialAbsThreshold. Prvá z nich teda vypočíta diferenčnú funkciu presne podľa rovnice (7) a druhá zo zadaného zoznamu kandidátov nájde prvého vyhovujúceho, prípadne globálne minimum. V záujme vyhnutia sa duplicity kódu som teda initialAbsThreshold využil aj v metóde absoluteThreshold. Interpolácia a normalizácia sú však implementované tak, že sa dajú využiť aj pri prvotnom odhade – vo svojich parametroch majú dolnú, respektíve hornú hranicu pre hľadaný rozsah τ.

Po vypočítaní prvotného odhadu algoritmus pristupuje k výpočtu finálneho, k čomu potrebuje nájsť kandidátov z funkcií v okolí offsetu. Z prvotného odhadu sa takisto určí nový rozsah τ pre hľadanú periódu v okolí (lowerBound a upperBound).

4.2.4 Diferenčné funkcie

Diferenčné funkcie v okolí offsetu sa počítajú v metóde diferencneFunkcie. Tu sa zároveň určí počet počítaných funkcií. Prvá funkcia sa vypočíta podľa rovnice (7) ale iba pre τ menšie než vrchný strop nového hľadaného rozsahu. Do lokálnej premennej sa uložia prvé

členy, t.j. prvé sčítance pre jednotlivé hodnoty τ. Potom sa v cykle (Obrázok 12) ku každej funkcii pripočítajú nové členy a staré sa odčítajú, pričom sa prvé členy znova uložia do lokálnej premennej.

```
for (int i = 1; i < pocetFunkcii; i++) {
    int t = lavyOffset + i;//aktuálny offset
    int tPlusTaumax = t + paUpperBound; // toto by malo ušetriť nejaké sčítavania
    for (int tau = 1; tau <= paUpperBound; tau++) {
        long temp = buffer[tPlusTaumax] - buffer[tPlusTaumax + tau];
        dif[i][tau] = dif[i - 1][tau] - prveCleny[tau - 1] + temp * temp;
        temp = buffer[t] - buffer[t + tau];
        prveCleny[tau - 1] = temp * temp;
    }
}
```

Obrázok 12 Ukážka cyklu pre výpočet okolitých diferenčných funkcií

4.2.5 Interpolácia

Pre nájdenie a interpoláciu všetkých lokálnych miním vo funkcii slúži metóda interpolate. Postupne prechádza všetkými bodmi normalizovanej funkcie od dolnej hranice hľadanej periódy (1 pre prvotný odhad) a porovnáva jej susedné hodnoty. Ak zistí prechod z klesania funkcie na rast, nad daným lokálnym minimom zavolá metódy na samotnú interpoláciu. Interpolácia sa vykonáva zároveň na poli typu long a na poli typu double, preto musia existovať dve metódy interpolateNorm a interpolateDif. Tie vrátia ordinátu resp. abscisu vrchola paraboly, t.j. kandidáta na výslednú periódu. Tento kandidát sa pridá do zoznamu kandidátov pre danú funkciu a ak jeho hodnota spadá pod prahovú hodnotu, hľadanie ďalších sa ukončí.

Keďže riešenie sústavy troch rovníc pre každé minimum je dosť neefektívne, využil som fakt, že všetky susedné body funkcií sú od seba horizontálne vzdialené o konštantnú veľkosť (jeden časový index) a zostrojil som nasledovný postup:

Z troch rovníc paraboly zostavíme sústavu:

$$\begin{aligned} y_1 &= ax_1^2 + bx_1 + c \\ y_2 &= ax_2^2 + bx_2 + c \\ y_3 &= ax_3^2 + bx_3 + c \end{aligned} \tag{10}$$

Ak vychádzame z predpokladu, že $x_1 = 0$, potom je $x_2 = 1$ a $x_3 = 2$. Po dosadení dostávame sústavu uvedenú v rovnici (11).

$$y_1 = c$$
$$y_2 = a + b + c \quad (11)$$
$$y_3 = 4a + 2b + c$$

Z druhej rovnice teda môžeme vyjadriť:

$$a = y_2 - b - y_1 \quad (12)$$

Po dosadení do tretej rovnice a ekvivalentných úpravách dostávame:

$$b = -\frac{y_3 - 4y_2 + 3y_1}{2} \quad (13)$$

Abscisu vrchola potom získame ako $-\frac{b}{2a}$. Keďže sme ale použili predpoklad $x_1 = 0$, k tejto abscise musíme ešte prirátať skutočnú hodnotu x_1. Ordinátu vrchola získame dosadením abscisy do získanej rovnice paraboly.

4.3 Trieda Controller

Ovládanie analýzy sa deje v triede Controller. Tá po zavolaní metódy dekodujAAnalyzuj najskôr zavolá decode v AudioDecoder-i a vytvorí inštanciu algoritmu. Pre paralelný výpočet vytvorí fixný thread pool so zadaným počtom vlákien. Na implementáciu úlohy je v metóde zadefinovaná lokálna trieda Worker implementujúca Callable. Analýza sa vykonáva v bodoch vzdialených od seba o hodnotu τ_{max}. Pre každý odhad F_0 v čase sa vytvorí inštancia triedy Worker so zadaným offestom, ktorá sa posunie thread poolu na spracovanie. Po dokončení výpočtu sa z výsledku každého objektu Future (ako výsledku práce vlákna) získa frekvencia a uloží sa do poľa výsledkov.

4.4 Trieda AudioDecoder

Aplikácia podporuje tie isté formáty zvukových súborov ako Java – AIFF, AU, WAV, ale iba s jedným kanálom a so 16b hĺbkou.

Dekódovanie sa vykonáva v metóde decode() v triede AudioDecoder. Najskôr sa prečítajú všetky bajty z AudioInputStream-u a uložia sa do bajtového poľa. Tým pádom je dĺžka zvukovej stopy v bajtoch limitáciou dĺžky zvukovej nahrávky, ktorú možno dekódovať. Maximálna bezpečná veľkosť poľa v Jave má hodnotu Integer.MAX_VALUE – 8. „Bezpečná“ preto, lebo niektoré VM si vyhradzujú miesto pre hlavičku poľa a pokus o alokáciu väčšieho tak môže vyvolať chybu OutOfMemoryError [12]. Do tohto obmedzenia sa však zmestí pri 16 bitovej hĺbke a jednom kanáli so vzorkovacou frekvenciou 44,1 kHz takmer 7 hodinová nahrávka. Po tomto načítaní sa pole obalí

bajtovým bufferom pomocou ByteBuffer.wrap, potom sa zistí či sú dáta vo formáte little-endian alebo big-endian a nakoniec sa z tohto bajtového buffera čítajú v cykle čísla typu short (čiže dva bajty).

4.5 Trieda MIDIDecoder

Aplikácia podporuje dekódovanie SMF (Standard Midi File) súborov typu 1. Tie na rozdiel od typu 0 môžu mať viac ako jednu stopu (track). Typ 2 zasa môže mať viacej sekvencií. Jedna sekvencia môže mať v typoch 1 a 2 viacero stôp. Jednotlivé stopy obsahujú samotné dátové správy. Je však bežne používanou konvenciou, že pri type 1 v stope č. 0 sú uložené iba rôzne meta správy, hlavne v nej býva uložené tempo [13].

Metóda decode(File) v triede MIDIDecoder najskôr zistí typ delenia času v sekvencii a jeho rozlíšenie. V MIDI štandarde sa časové značky udalostí udávajú pomocou tickov a tempo je udávané v μs za štvrťovú notu. Od použitého delenia závisí aj druh rozlíšenia. Pri použití PPQ (Pulses Per Quarter note) rozlíšenie udáva počet tickov za štvrťovú notu a pri použití niektorého z SMPTE (Society of Motion Picture and Television Engineers) delení udáva počet tickov za rámec (frame). SMPTE delenia sú 4:

- 24 rámcov/s
- 25 rámcov/s
- 29.97 rámcov/s
- 30 rámcov/s

Aplikácia podporuje všetky typy delenia času (PPQ a všetky druhy SMPTE). Tempo však nemusí byť nastavené vôbec, vtedy sa používa východzie nastavenie (500 000 μs za štvrťovú notu).

Najskôr sa teda prehľadáva track č. 0 pre správu s kódom 0x51 (SET_TEMPO), ktorá nastavuje hodnotu tempa. Pre zachovanie jednoduchosti sa berie do úvahy iba prvé tempo ktoré sa nájde, potom sa prehľadávanie ukončí. Toto tempo sa teda považuje za platné v celej sekvencii.

V metaspráve SET_TEMPO je hodnota tempa uložená ako trojbajtové celé číslo vo formáte big-endian, ktoré je v Jave interpretované ako pole troch bajtov. V Jave je int 32 bitové celé číslo so znamienkom vo formáte big-endian. Keďže sa v Jave byte konvertuje pred použitím bitovej operácie na int [14], je treba na danú hodnotu použiť masku. Na

uloženie tempa z bajtového poľa teda stačí zobrať každý bajt tempa, aplikovať naňho masku 0xff a bitovo posunúť na správne miesto, ukážka kódu je na obrázku (Obrázok 13).

```
byte[] data = mm.getData();
tempo = (data[0] & 0xff) << 16 | (data[1] & 0xff) << 8 | (data[2] & 0xff);
```

Obrázok 13 Konvertovanie bajtového poľa s tempom na int

Po získaní tempa sa zistí dĺžka trvania jedného ticku v µs. Ak je typ delenia PPQ, vydelí sa hodnota tempa rozlíšením, t.j. počtom tickov za štvrťovú notu. Pre delenie typu SMPTE vydelíme hodnotu 1 000 000 násobkom počtu tickov za rámec a počtu rámcov za sekundu.

Následne sa začnú v stope č. 1 vyhľadávať samotné noty. Pre „zapnutie“ noty sa v midi používa kód správy NOTE_ON (0x90). Táto správa v sebe obsahuje dva bajty informácií – prvý predstavuje číslo tónu, ktorá sa má hrať a druhý jeho silu (hlasitosť). Spôsob pre „vypnutie“ noty však môže byť dvojaký. Prvý je použitie kódu správy NOTE_OFF (0x80). Tu opäť prvý bajt dát určuje ktorý tón sa má vypnúť. Druhý spôsob je použiť NOTE_ON správu s druhým dátovým bajtom rovným nule. Dekodér teda sleduje zapnutia nôt a ich vypnutia a na základe ich časových značiek zaradí do zoznamu noty s príslušným časovým začiatkom a koncom. Kvôli vyhnutiu sa polyfónii dekodér predčasne ukončuje hranú notu, ak pred jej signálom na ukončenie začne hrať druhá. V prípade že by takáto mala nulovú dĺžku, do zoznamu nôt sa nepridá.

Noty mimo rozsahu A0-C8 sa ignorujú. Takisto sa ignorujú všetky ostatné správy, ako napr. HOLD_PEDAL, ktorým sa určuje či má byť držaný damper pedál – t.j. noty nemajú byť tlmené ani po ich vypnutí správou NOTE_OFF, ale majú prestať hrať až pri pustení pedálu. Keďže sa noty hľadajú iba v stope č. 1, treba v nej mať pre správnu funkčnosť uložené noty na dekódovanie.

4.6 Trieda Evaluator

Pomocou dekódovaných nôt sa určuje úspešnosť algoritmu v triede Evaluator. Počíta sa iba s rozsahom nôt A0-C8 v rovnomerne temperovanom ladení. Výsledkom evalvácie je percentuálna zhoda odhadov frekvencií s očakávanými notami z MIDI súboru. Frekvencia je prehlásená za ten tón, ku ktorému je najbližšie. Pri nerozhodnosti je automaticky považovaný výsledok za chybný. Na poli tónov z enumerácie Ton získaných metódou values() sa pomocou Arrays.binarySearch hľadá daná frekvencia. Z vrátenej hodnoty vieme medzi ktorými tónmi sa nachádza. Zistíme ku ktorému z nich je bližšie a ak je bližšie k tónu zodpovedajúcemu note v danom čase, pripočítame zhodu. Toto

porovnávanie je vždy vykonávané pre každý odhad frekvencie, ktorý sa nachádza v čase, kde je hraná nejaká nota, t.j. výsledky počas ticha sa do evalvácie nezarátavajú.

4.7 Trieda ToneAxis

Na vykresľovanie grafu používam knižnicu JFreeChart. Keďže nepodporuje namapovanie vlastných značiek (reťazcov reprezentujúcich určité hodnoty) na os grafu, musel som si vytvoriť vlastnú os. Trieda ToneAxis dedí triedu NumberAxis z knižnice. Keďže je os používaná iba vo vertikálnom tvare, stačilo prekryť dve metódy – refreshTicksVertical a calculateVisibleTickCount. Pri vytvorení jej inštancie sa uloží do atribútov pole tónov z enumerácie Ton. Metóda refreshticksVertical sa volá vždy pri zmene grafu, keď treba prekresliť os (ak je vertikálna) a vracia List viditeľných značiek tónov. Prekrytá metóda calculateVisibleTickCount() zistí aktuálny rozsah zobrazenej osi a vráti počet tónov, ktoré do tohto rozsahu (vo frekvenciách) spadajú. Pomocná metóda calculateLowestVisibleTickIndex() vráti index najnižšieho viditeľného tónu v poli. Metóda refreshTickVertical teda využije tieto dva údaje a vytvorí zoznam značiek pre zobrazenie.

4.8 GUI

Pre grafické rozhranie som zvolil platformovo nezávislý Swing toolkit. GUI pozostáva z dvoch okien. Jedno je reprezentované triedou GrafOkno a slúži na zobrazovanie výsledkov. Sú v ňom prepínače osi grafu (frekvenčná/tónová os), JLabel pre zobrazenie percentuálnej evalvácie a ChartPanel, pre zobrazenie samotného grafu. Časová os grafu má maximálne rozlíšenie 1ms, preto sa pri vytváraní okna zistí časové rozlíšenie analýzy v milisekundách, a každému odhadu frekvencie sa pridelí čas v ms. To má pri použití kratšieho okna než 1ms za následok to, že ak do jedného vykresľovaného úseku s dĺžkou 1ms spadá viacero frekvenčných výsledkov, prekryjú sa posledným (na čo používateľa upozorňuje aplikácia už pri nastavení malej hodnoty τ_{max}). Na samotnú evalváciu to však samozrejme vplyv nemá.

Pomocou hlavného okna (trieda HlavneOkno) používateľ otvára súbory, spúšťa analýzu, či zobrazuje vykreslené grafy. Okno zobrazuje základné informácie o otvorenom súbore a dáva možnosť používateľovi nastaviť parametre analýzy:

- Threshold v rozmedzí 0 až 1 s krokom 0,1. Východzia hodnota je 0,1
- Povolenú percentuálnu odchýlku finálneho od prvotného odhadu v rozmedzí 0% až 40%. Východzia hodnota je 20%

- τ_{max} v rozmedzí závislom od vzorkovacej frekvencie signálu. Dolná hranica je vždy 2 a horná je $\frac{1}{26}$ vzorkovacej frekvencie v Hz. Východzia hodnota je $\frac{1}{27}$ vzorkovacej frekvencie.
- Počet konkurenčných vlákien použitých pri analýze v rozmedzí 1 až 8

Táto trieda je tiež zodpovedná za odchytávanie výnimiek a zobrazovanie chýb a varovaní. Keďže by spúšťanie analýzy z EDT (Event Dispatch Thread) zablokovalo GUI, analýza sa spúšťa na novom vlákne. Toho je docielené pomocou anonymnej triedy SwingWorker v metóde volanej po príkaze spustiť analýzu. Po dokončení práce si EDT prevezme výsledky analýzy a zobrazí okno s grafom.

5 Evalvácia

Keďže ľudská hra na nástrojoch by robila presnú anotáciu hudobných nahrávok takmer nemožnou, zvolil som možnosť generovania nahrávok na PC. To mi umožnilo vygenerovať aj MIDI notáciu, ktorá definuje presný začiatok a koniec hranej noty a teda aj očakávanej F_0. Samozrejme, MIDI správa pre vypnutie noty ešte neznamená, že po nej už daný tón nebude hrať. Môže mať (a často aj máva) určitý dozvuk, ktorý závisí od samotnej interpretácie syntetizátora. Tento dozvuk má však pri transkripcii a zisťovaní dĺžky nôt takmer nulovú váhu, lebo po prvé býva veľmi krátky a po druhé dĺžka noty určuje samotnú dobu hrania tónu (napr. stlačenej klávesy) bez ohľadu na dozvuk, či ozvenu vyplývajúcu z akustických vlastností priestoru.

5.1 Príprava nahrávok

Pre zápis hraných nôt som použil notačný program Sibelius 7.5.0 v jeho trial verzii. Pre napodobenie skutočnej interpretácie na nástroji som nastavil štýl interpretácie na Espressivo (vo verzii 2). Táto vlastnosť pridáva generovanému prednesu malé zmeny dynamiky hry, ktoré sa snažia frázovaním napodobiť ľudskú hru. Nastavenie Rubato som však ponechal na možnosti Meccanico, ktorá nedovoľuje robiť zmeny v tempe počas hry. Z týchto nôt som potom vygeneroval v Sibeliuse SMF súbory ako referenčný zápis nahrávok a takisto aj samotné zvukové nahrávky.

Pre vytvorenie samotných zvukových nahrávok som v záujme zachovania čo najväčšej vernosti s reálnymi nástrojmi siahol po VST plugine Kontakt player 5. V tomto plugine som použil vzorkované zvuky hudobných nástrojov z 560MB knižnice Kontakt Factory Selection. Pre klavírne nahrávky som však zvolil 265MB veľký, voľne šíriteľný virtuálny nástroj pre Kontakt player s názvom The City Piano od Joea Stevensa (Bigcat Instruments) a Joea Waltera. Je v ňom plne vzorkované krídlo Baldwin Baby Grand so štyrmi hlasitostnými vrstvami a vzorkami uvoľnenia kláves. Vygenerované zvukové stereo súbory formátu WAVE (16b, 44,1 kHz) som potom zlúčil na mono v open source programe Audacity a porovnal s prislúchajúcim MIDI súborom.

Všetky nahrávky boli vygenerované za použitia 12 tónového rovnomerne temperovaného ladenia s A4=440 Hz. V ňom je pomer F_0 medzi dvoma susednými tónmi $\sqrt[12]{2} \approx 1.059463$. Tento systém ladenia je v súčasnosti jedným z najpoužívanejších v západnej

hudbe, i keď je možné sa stretnúť aj s inými (didymické, stredotónové, rôzne nerovnomerne temperované ladenia a i.).

Rozsah je pre mnohé nástroje ťažké určiť, keďže často závisí od virtuozity hráča (hlavne pri dychových nástrojoch), či od použitého ladenia strún alebo iného nastavenia nástroja. Preto som pre rozsah použil najtypickejšie hodnoty. Pre 88 klávesový klavír to je A0 – C8, pre 6 strunové gitary v štandardnom ladení strún (E2, A2, D3, G3, B3, E4) E2 – E5 a pre C trúbku F#3 – D6. Vo všeobecnosti má gitara má hraný rozsah o oktávu nižšie než písaný, avšak pri generovaní nahrávok tento rozdiel zmizol, lebo VST plugin generoval tóny s F_0 pre písaný tón, čo umožnilo porovnanie s MIDI anotáciou.

Nástroje na ktorých teda prebehla evalvácia sú: klavír, čistá elektrická gitara (jazz guitar – bez overdrive-u), elektrická gitara s overdrive-om (rock guitar) a tlmená trúbka (muted trumpet). Najskôr som vytvoril syntetické nahrávky. Pre každý z nástrojov som najskôr vygeneroval vzostupnú chromatickú stupnicu v štvrťových notách, tenuto, pri 30 BPM (dĺžka noty 2 s) s udanou dynamikou mezzoforte. Rozsah tejto stupnice zodpovedal prirodzenému rozsahu nástroja. Druhá chromatická nahrávka je generovaná z tých istých nôt, ibaže s nastavením 160 BPM (dĺžka noty 0,375 s). V týchto nahrávkach neboli použité žiadne ďalšie artikulačné nastavenia a slúžia okrem iného na prehľadné zistenie, ako sa algoritmus dokáže popasovať s tónmi v celom rozsahu nástroja.

Pre nahrávky skutočnej hudby som prepísal melódiu z prvých 12 taktov s predtaktím z nokturna Frederica Chopina op.9 č.2 [15] s vynechaním pedálu a zmien tempa. Túto skladbu som zvolil preto, lebo je v nej viacero veľmi krátkych nôt a ozdôb (obsahuje aj trilky, mordenty či gruppettá), variabilnú dĺžku nôt, legato aj portato (hru viazane aj mierne oddelene), dynamiku v rozsahu od pianissima až po forte a má dobre rozlíšiteľnú melódiu vhodnú pre monofonický zápis. Takisto neobsahuje žiadne techniky hry špecifické pre niektorý z testovaných nástrojov (gitarové ohýbanie tónu a i.) a zároveň sa rozsah nôt zmestí do rozsahu všetkých testovaných nástrojov. Tým pádom je vhodným zástupcom širokej palety hudby. Pre gitarové nahrávky som celý notový zápis transponoval o oktávu nižšie, aby sa zmestil do ich hraného rozsahu.

5.2 Analýza

Pri evalvácii bola použitá prahová hodnota (Threshold) 0,1, povolená odchýlka 20% a hodnota τ_{max} sa odvíjala od použitého nástroja. Vždy som zvolil takú najmenšiu hľadanú frekvenciu aby približne zodpovedala tónu o poltón nižšie než najnižší tón z rozsahu

nástroja. Pre klavírne nahrávky som teda nastavil τ_{max} na 1696 (min. hľ. frekvencia 26 Hz), pre gitarové nahrávky na 569 (77,5 Hz) a pre nahrávky trúbky na 253 (174,3 Hz).

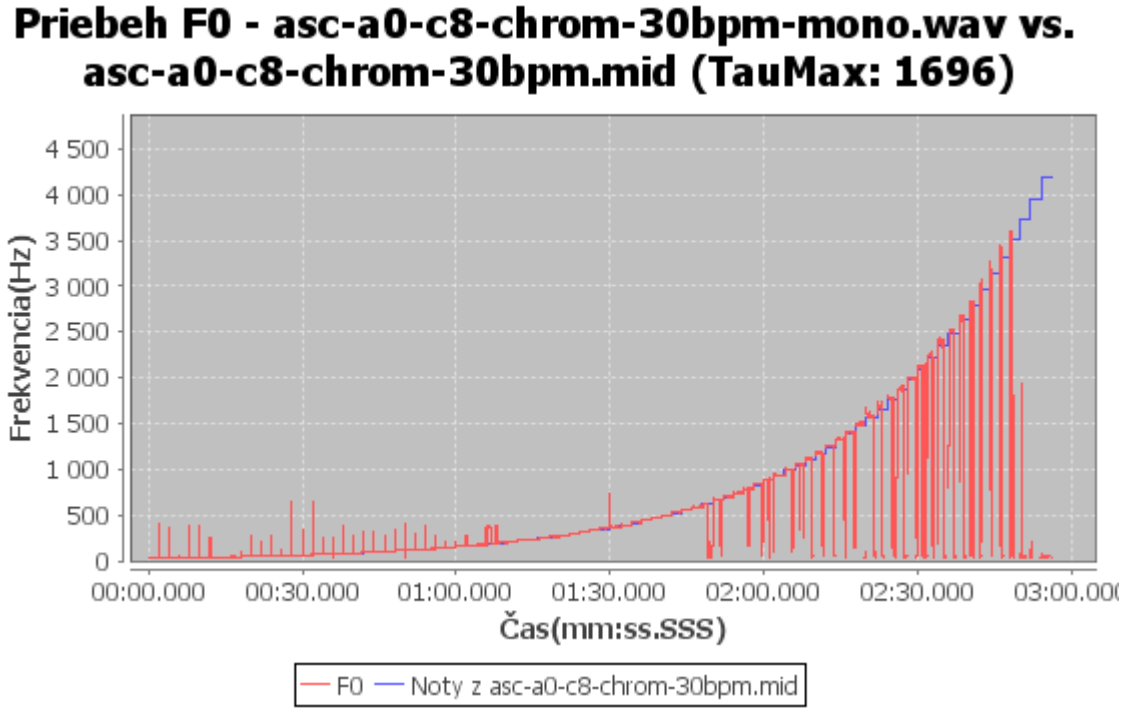
Výsledkom evalvácie je percentuálna zhoda odhadov tónov s očakávanými notami z MIDI súboru. Frekvencia je prehlásená za ten tón, ku ktorému je najbližšie a pri nerozhodnosti je automaticky považovaný výsledok za chybný. Každý odhad tónu, ktorý sa nachádza v čase, kde je podľa anotácie hraná nota sa porovná s anotáciou, čím sa vo výsledku určí pomer zhôd. Počty týchto vyhodnocovaných dohadov sa odvíjajú okrem iného aj od nastavenia τ_{max}. Tieto počty sú uvedené v Tab. 1.

Tab. 1 Počty vyhodnocovaných odhadov v nahrávkach

	Klavír	Tlmená trúbka	Gitary
Chromatika (160BPM)	946	2190	1112
Chromatika (30BPM)	4664	11537	5772
Nokturno op.9 č.2	1699	10622	4797

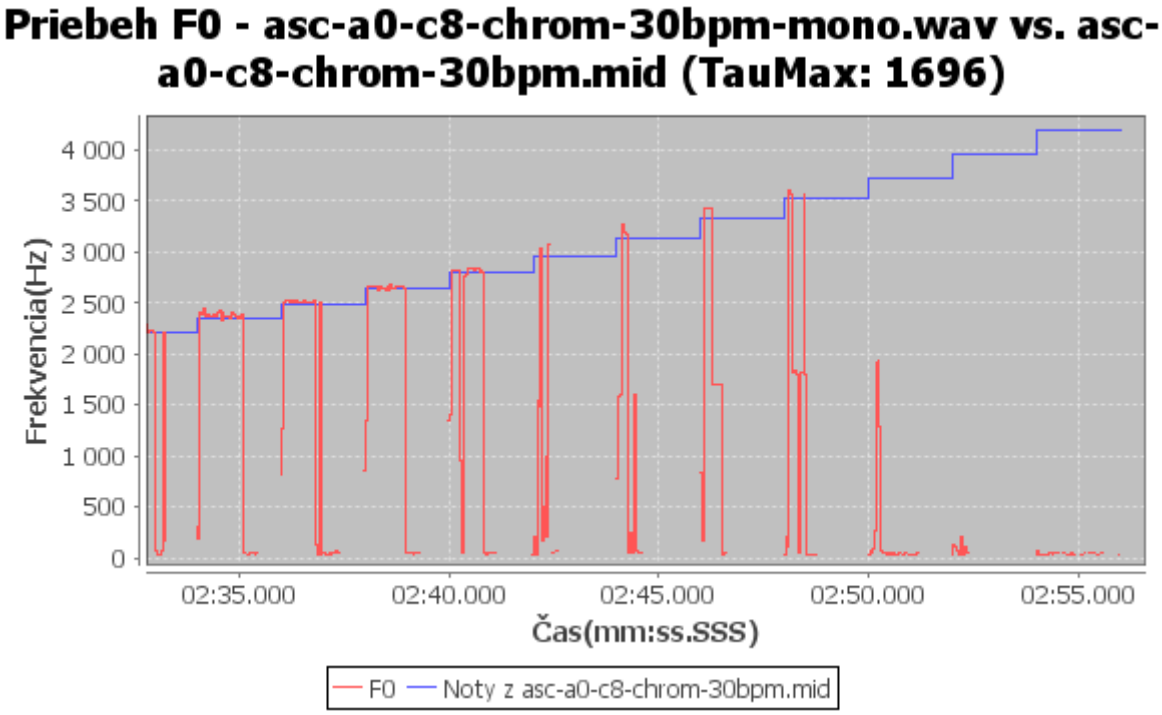
5.2.1 Klavír

Ako prvý nástroj pre analýzu som zvolil klavír a prvú nahrávku chromatickú stupnicu s 30 BPM. Výsledok celej analýzy je na obrázku (Obrázok 14).

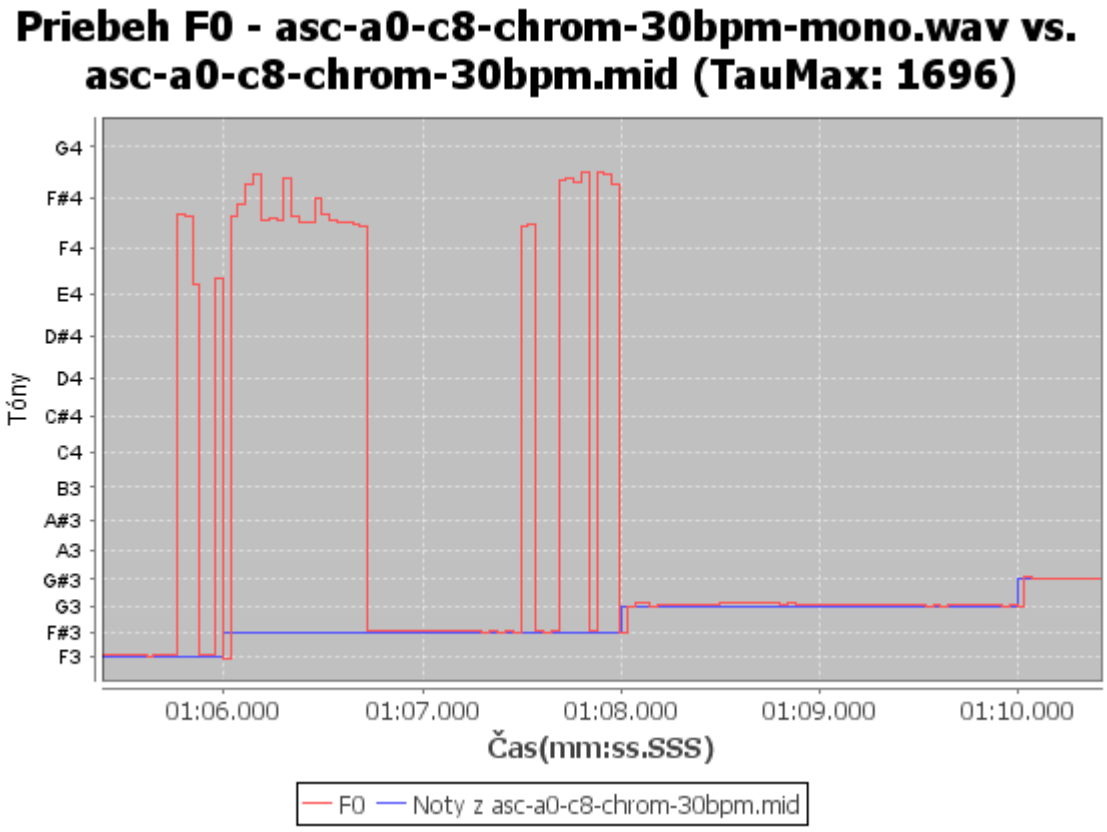


Obrázok 14 Priebeh F_0 pri chromatickej stupnici s 30 BPM na klavíri. Červenou farbou sú vykreslené odhadnuté frekvencie, modrou očakávané výsledky z MIDI anotácie

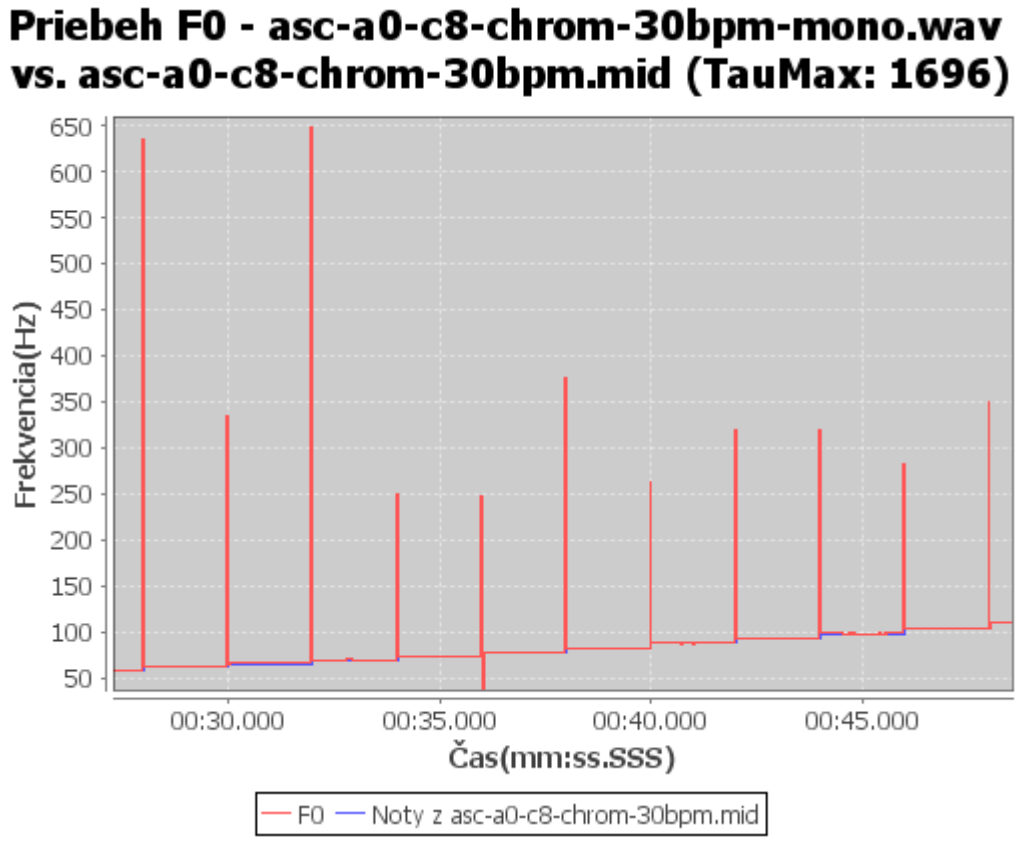
Pre túto nahrávku bola zhoda 80,017%. Najväčšie problémy mal algoritmus pri vrchných dvoch oktávach. Pre vrchné tri tóny neodhadol ani jednu F_0 správne (za predpokladu že nejakú našiel). To je pravdepodobne tým, že vyššie tóny klavíra majú kratší dozvuk, rýchlejšie sa tlmia a pre rovnakú vyvinutú silu hry vydávajú tichší zvuk. Do samotných tónov spôsobených vibráciou strún teda príliš zasahoval zvuk dopadajúceho kladivka a klávesy udierajúcej do plstenej výstuže pod ňou. Detail vrchnej oktávy je na obrázku (Obrázok 15). V strednom registri boli odhady väčšinou správne. Medzi najviditeľnejšie odchýlky patrí dobrý príklad oktávovej chyby na note F#3 (Obrázok 16), kde je jej veľká časť určená ako F#4. V spodnom registri je možné vidieť krátkodobé (väčšinou len jeden odhad) chyby vo veľkom rozsahu. Tie vznikli takmer výlučne len na prechode dvoch tónov. Detail je na obrázku (Obrázok 17).



Obrázok 15 Detail vrchnej oktávy chromatickej stupnice hranej na klavíri pri 30BPM

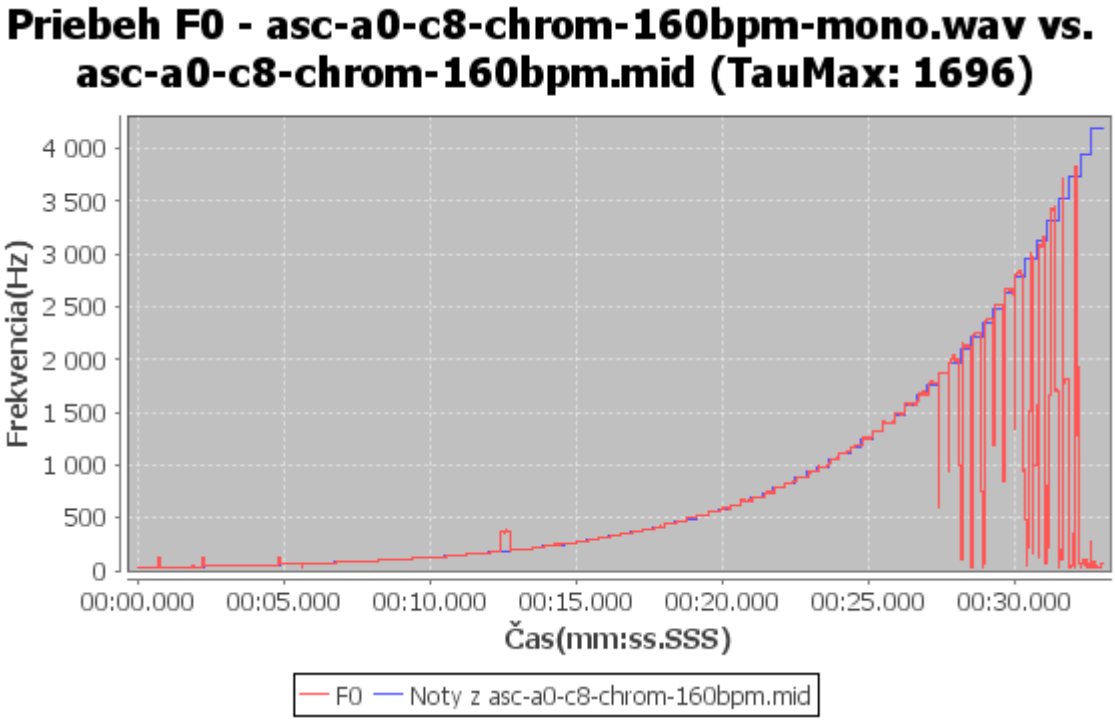


Obrázok 16 Oktávová chyba na chromatickej stupnici hranej na klavíri pri 30BPM

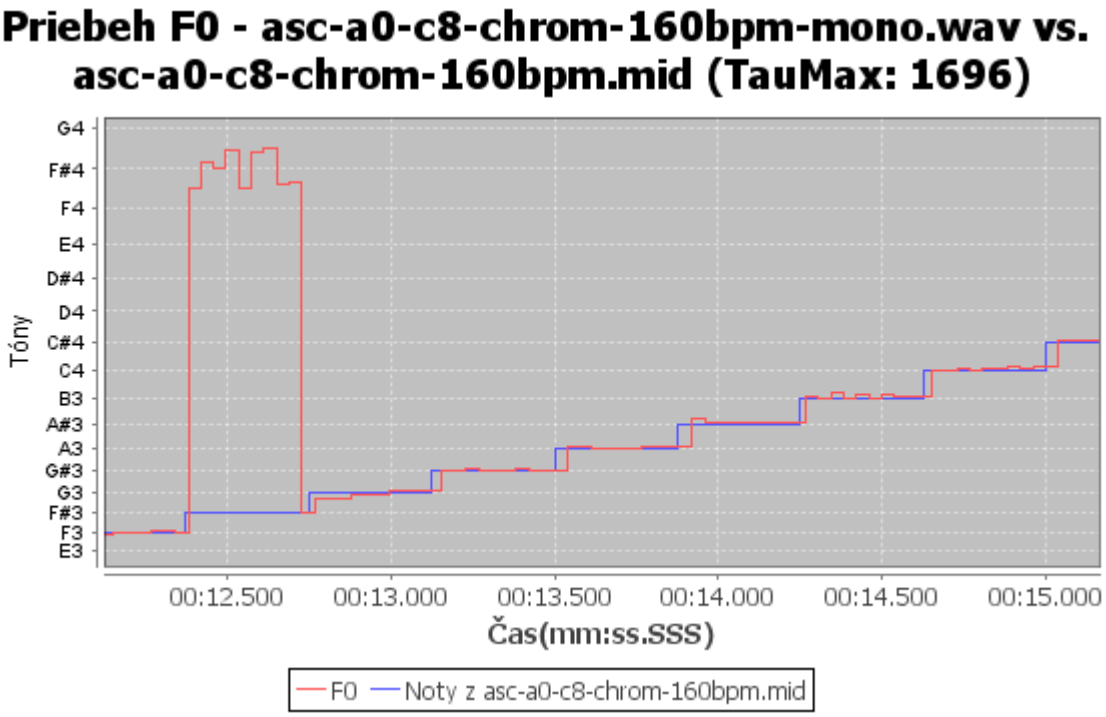


Obrázok 17 Chyby v prechodoch tónov na chromatickej stupnici hranej na klavíri pri 30BPM

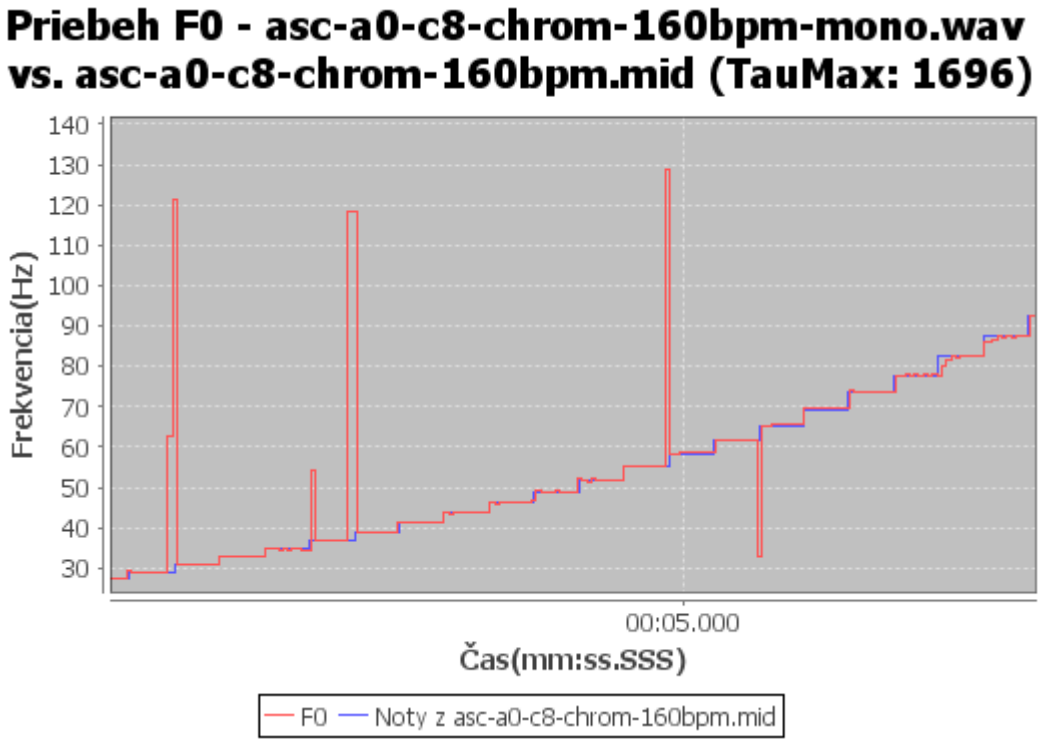
Druhá chromatická stupnica so 160 BPM dopadla veľmi podobne. Jej percentuálna zhoda bola 79,493% a jej graf je na obrázku (Obrázok 18). Je však vidno, že oktávová chyba na F#3 pokryla celé trvanie noty (Obrázok 19). Zaujímavým zistením bol fakt, že výrazne ubudol počet chýb v prechodoch tónov, ktoré bolo možné pozorovať pri 30 BPM. Detail zhruba rovnakej oblasti je na obrázku (Obrázok 20).



Obrázok 18 Priebeh F_0 pri chromatickej stupnici so 160 BPM na klavíri



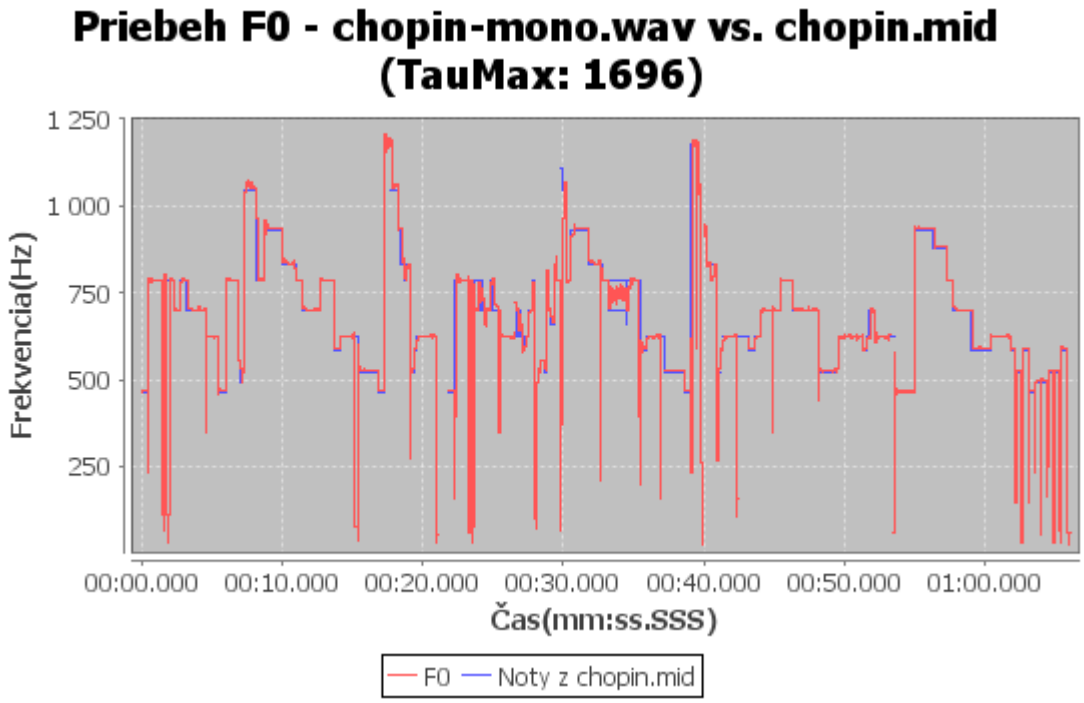
Obrázok 19 Detail oktávovej chyby na chromatickej stupnici hranej na klavíri pri 160 BPM



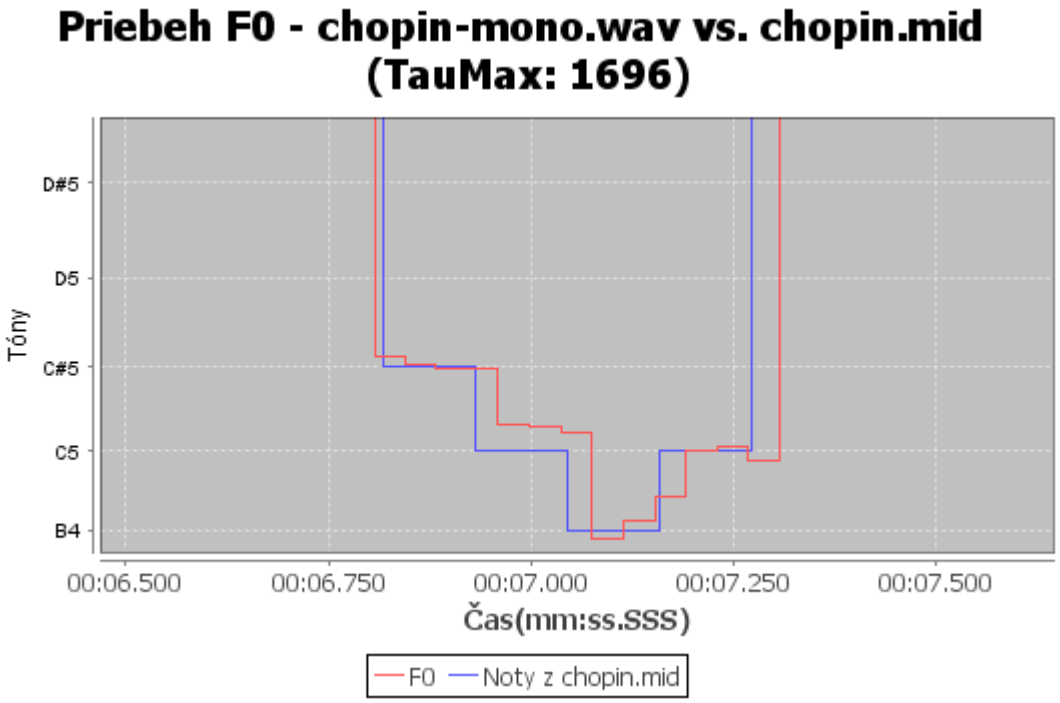
Obrázok 20 Chyby v prechodoch tónov na chromatickej stupnici hranej na klavíri pri 160 BPM

Poslednou testovanou nahrávkou bolo Chopinovo nokturno. Preň bola zhoda 83,932% a výsledný graf je na obrázku (Obrázok 21). Chybné odhady sa vyskytovali opäť najmä na prechodoch nôt, ale niekoľko sa ich našlo aj na ustálených notách. Nedá sa pozorovať žiadna korelácia medzi dynamikou hry, odhady sú väčšinou správne v celom dynamickom

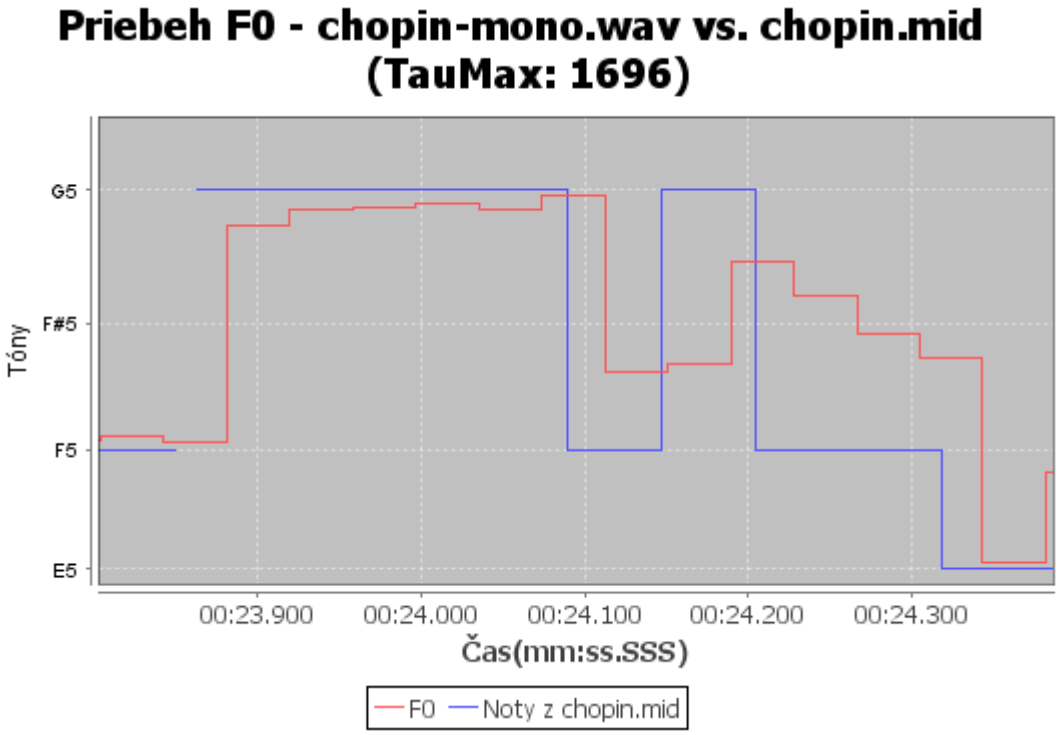
rozsahu, od pianissima až po forte. Odhady boli väčšinou správne aj pri rýchlej výmene nôt v gruppettách. Detail jedného gruppetta, kde dĺžka nôt dosahuje zhruba 110ms je na obrázku (Obrázok 22). Avšak pri mordentoch, kde dĺžka noty dosahovala len necelých 60ms sa dá povedať že analýza zlyhala. Trvania týchto tónov sú príliš krátke na to, aby sa na nich dokázal ustáliť správny odhad F0, a tak síce graf tvarom naznačuje zmeny nôt, sú odhady chybné. Ukážka vrchného mordentu na tóne F5 je na obrázku (Obrázok 23). Rovnaká situácia nastala aj pri trilkách (Obrázok 24) a dá sa aplikovať na všetky takto rýchle zmeny nôt (napr. appoggiatura).



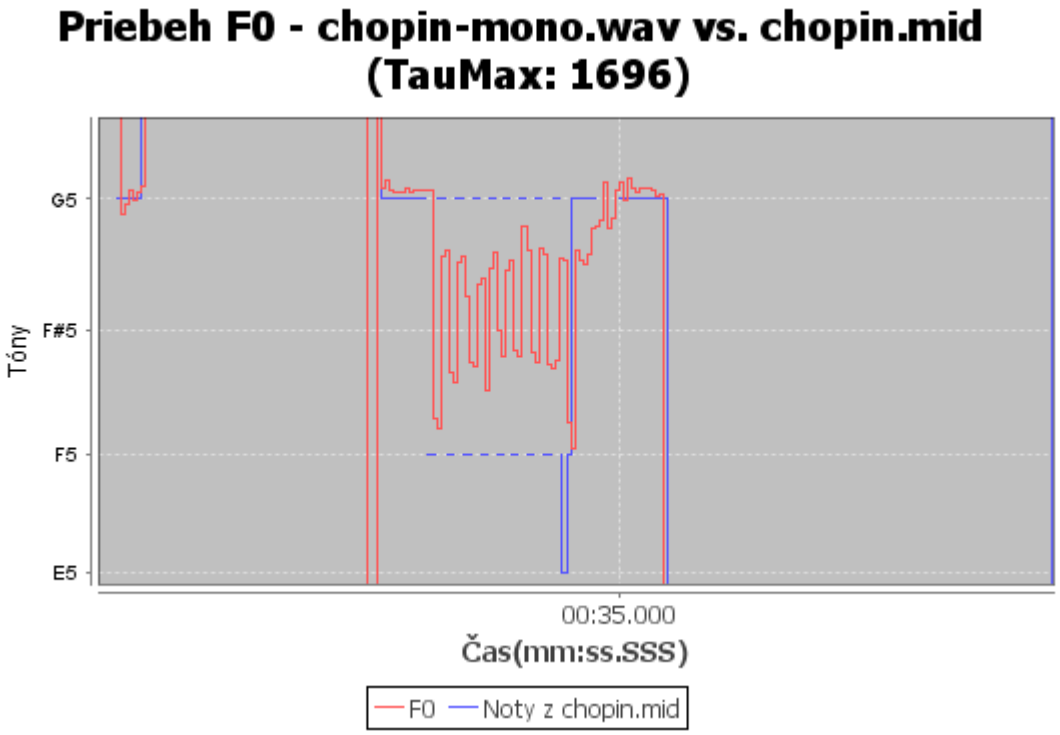
Obrázok 21 Priebeh F0 v nokturne na klavíri



Obrázok 22 Detail gruppetta na klavíri



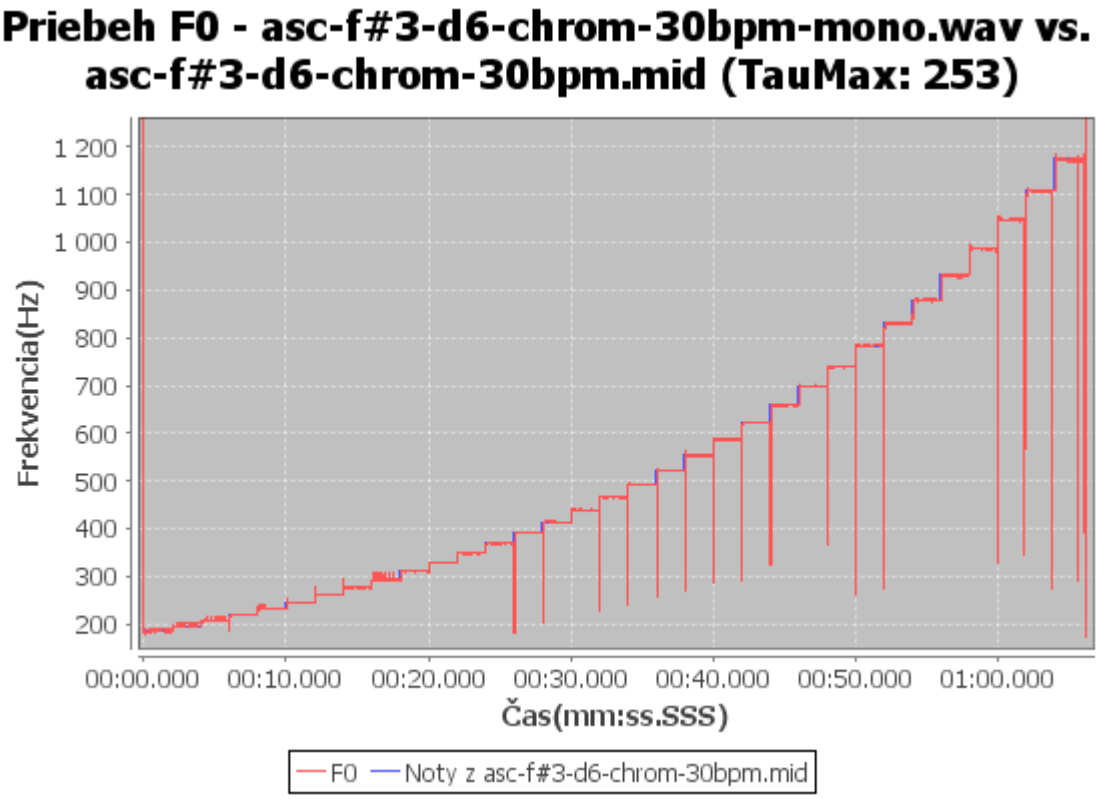
Obrázok 23 Detail vrchného mordentu na tóne F5 na klavíri



Obrázok 24 Trilky z nokturna hrané na klavíri

5.2.2 Tlmená trúbka

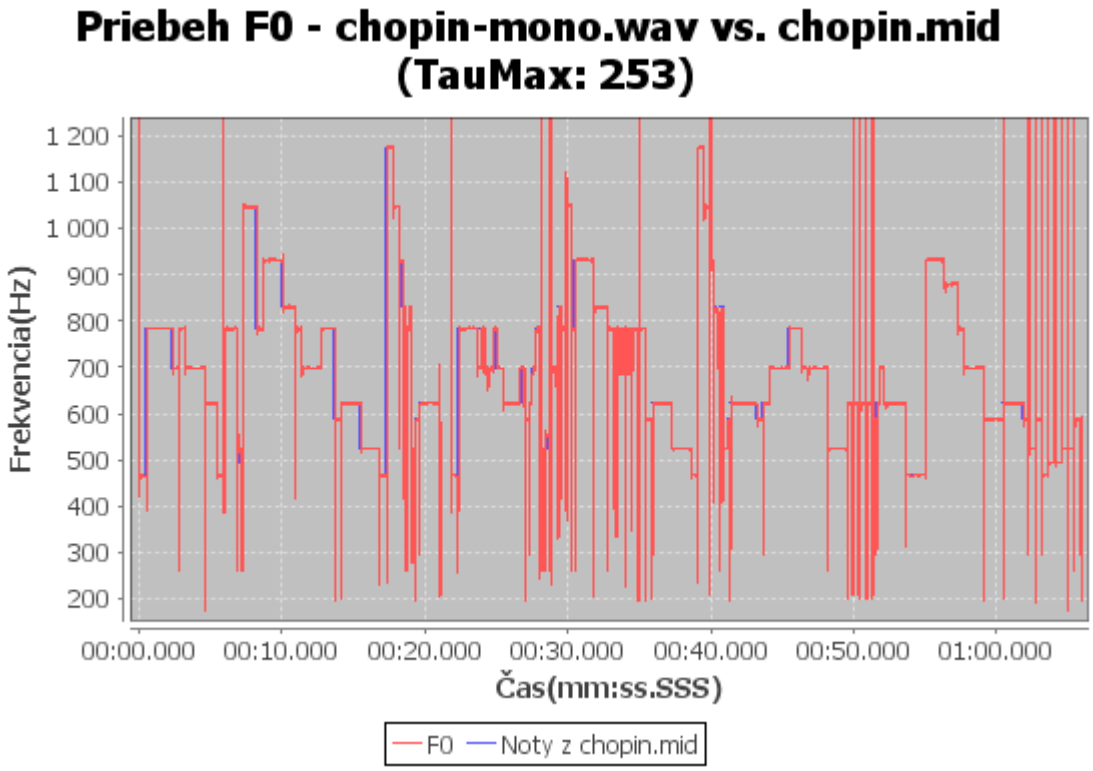
Pre chromatickú stupnicu s 30 BPM bola vypočítaná zhoda 94,392%. Väčšina chybných odhadov bola na spodných troch tónoch, kde chybne určili frekvenciu o poltón vyššie. Zvyšok chybných odhadov bol znova hlavne na prechodoch tónov (Obrázok 25).



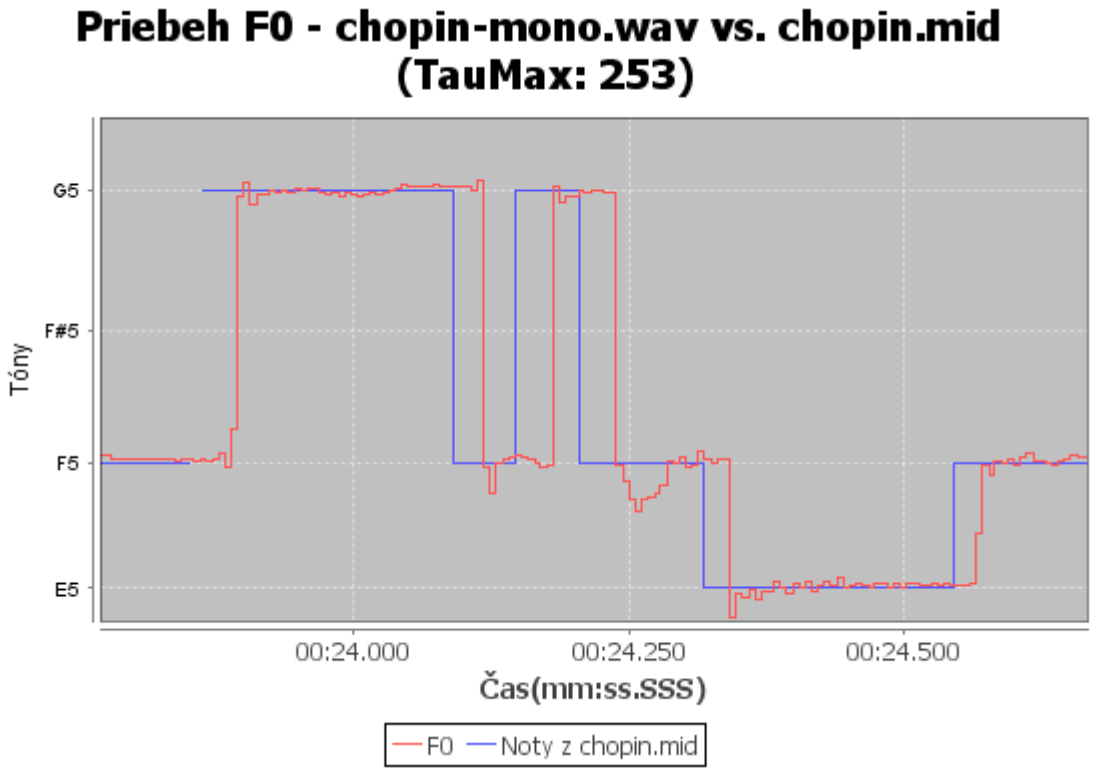
Obrázok 25 Priebeh F_0 na chromatickej stupnici hranej na trúbke

Druhá chromatická stupnica so 160 BPM sa líšila len minimálne, avšak väčšina chybných odhadov z prvých troch tónov zmizla. Zhoda bola 87,032%.

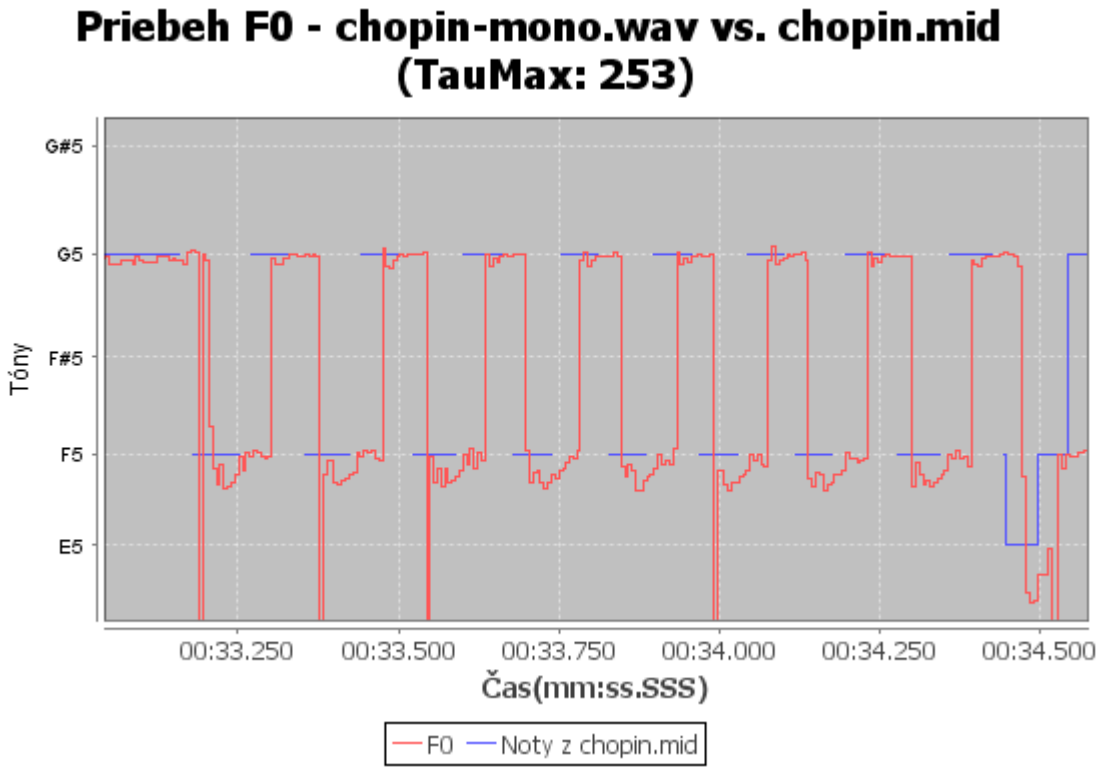
Pre nahrávku nokturna (Obrázok 26) vyšla zhoda 92,393%. Voči nahrávke klavíra vzrástol počet chybných odhadov pri prechode tónov. Pri mordentoch sa síce graf tvarom aj hodnotami podobal očakávanému výsledku viac ako pri klavírnej nahrávke, ale voči notám z MIDI súboru boli odhady posunuté v čase zhruba o 30ms, lebo algoritmus reagoval na zmenu tónu s oneskorením (Obrázok 27). Za to však nemôže hľadanie najlepšieho lokálneho odhadu, lebo τ_{max} je v tomto prípade veľmi malé (a teda aj rozsah časového intervalu v ktorom sa hľadá F0). Vysvetlením je silný presah dozvuku predchádzajúcej noty do ďalšej. To isté platí aj pre trilky (Obrázok 28).



Obrázok 26 Priebeh F_0 pri nokturne hranom na tlmenej trúbke



Obrázok 27 Mordent hraný na tlmenej trúbke

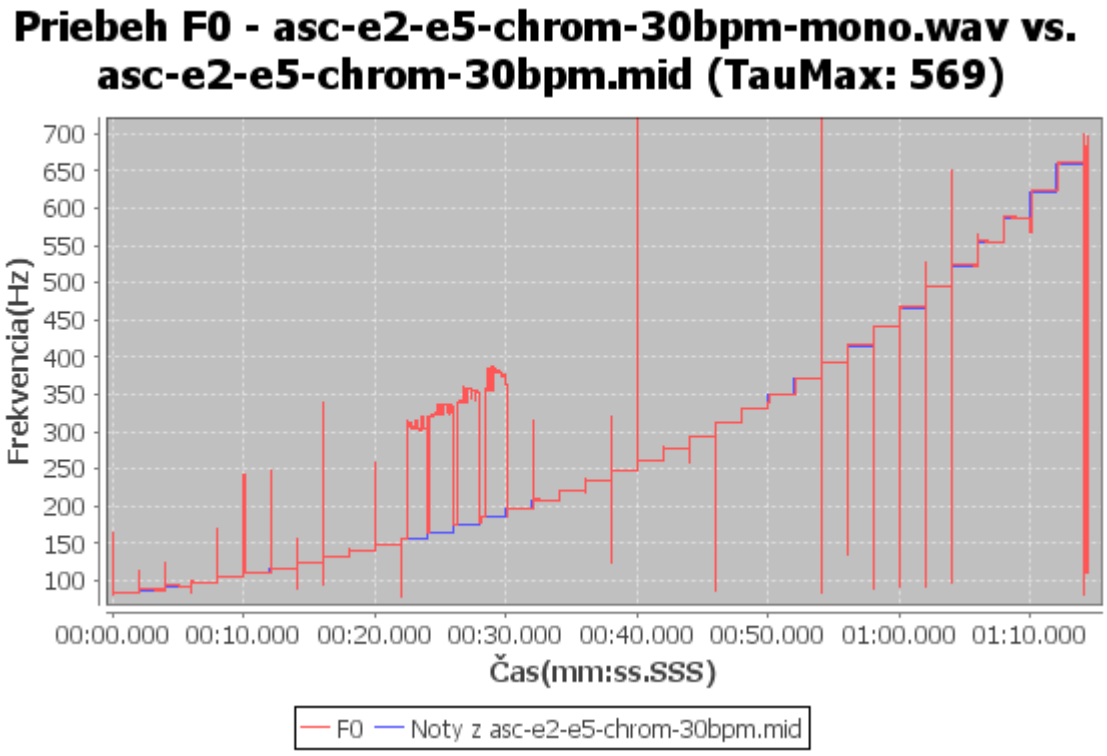


Obrázok 28 Trilky z nokturna hrané na tlmenej trúbke

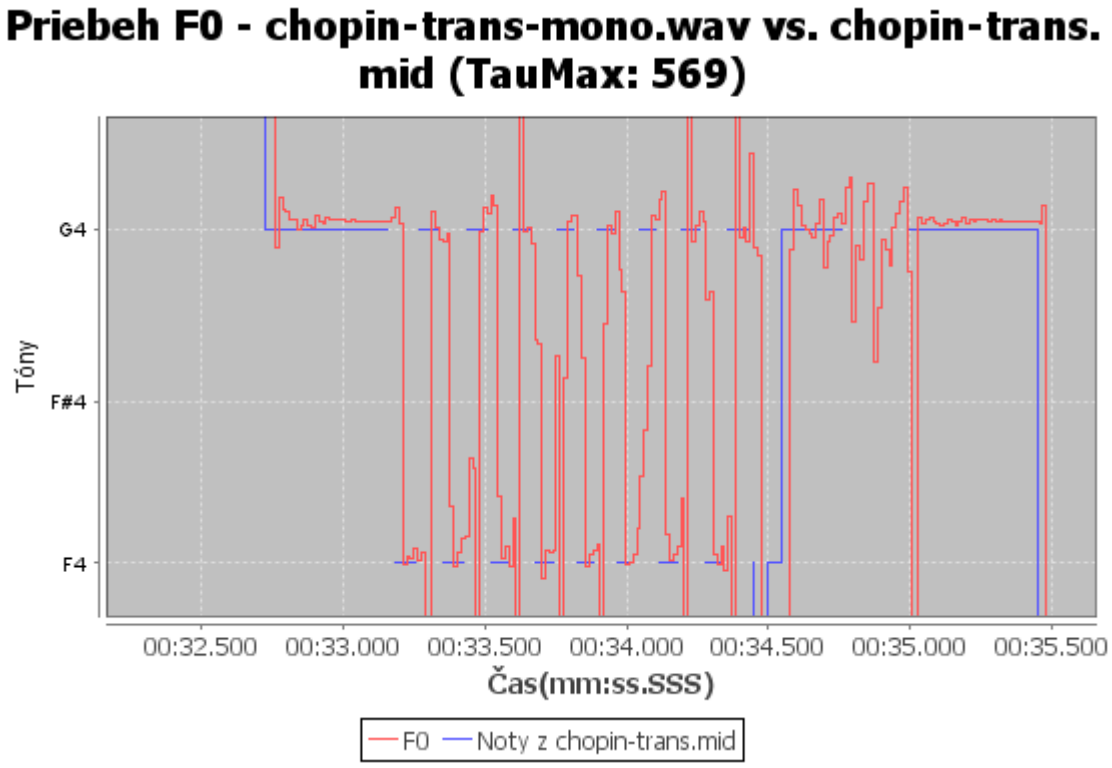
5.2.3 Čistá elektrická gitara

Pre prvú chromatiku s 30 BPM vyšla zhoda 89,432%. Okrem chýb na prechodoch tónov boli odhady chybné najmä na tónoch D#3 – F#3 (vrátane) (Obrázok 29). Chyby v takmer celej ich dĺžke trvania zodpovedali približne oktávovej chybe. Chromatika so 160 BPM dopadla obdobne so zhodou 88,869%, ale oktávové chyby z 30 BPM nahrávky sa prejavili v menšej miere – iba u E3 a F3.

Nahrávka nokturna mala zhodu 92,12% a výsledok sa veľmi podobal tomu z nahrávky trúbky. Tu takisto boli mordenty aj trilky určené viac menej dobre, až na to že boli posunuté zhruba o 30ms voči MIDI anotácii (Obrázok 30).



Obrázok 29 Priebeh F0 na chromatickej stupnici hranej na čistej el. gitare



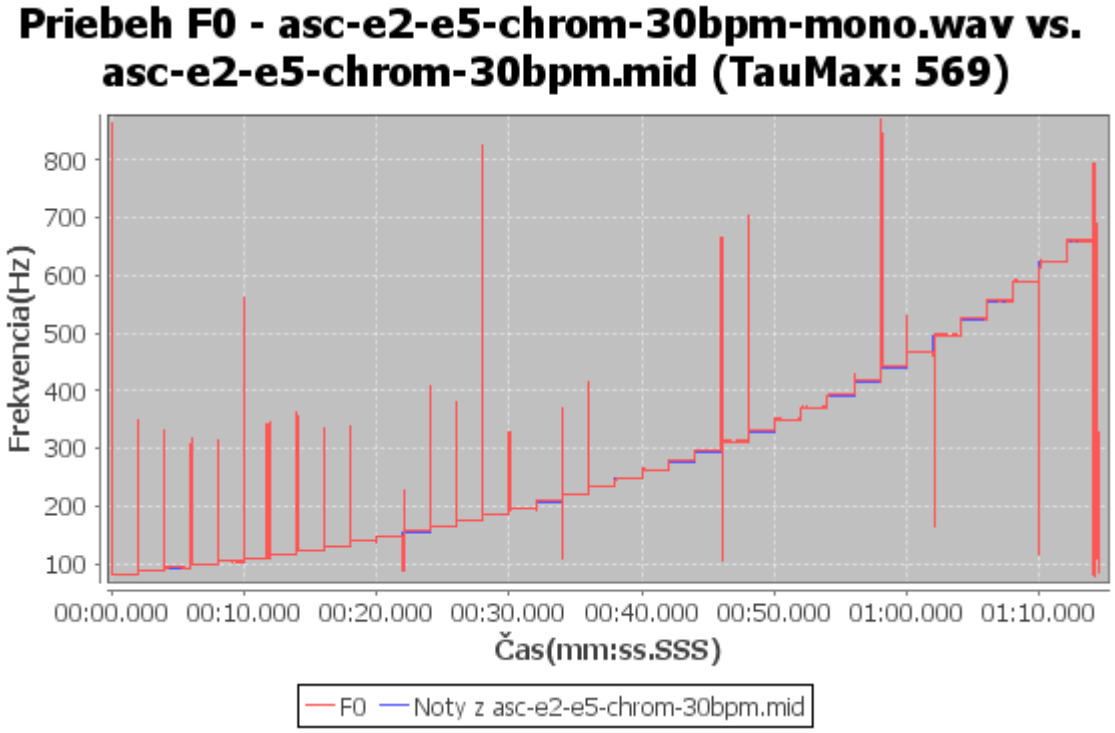
Obrázok 30 Trilky hrané na čistej el. gitare

5.2.4 El. gitara s overdrive-om

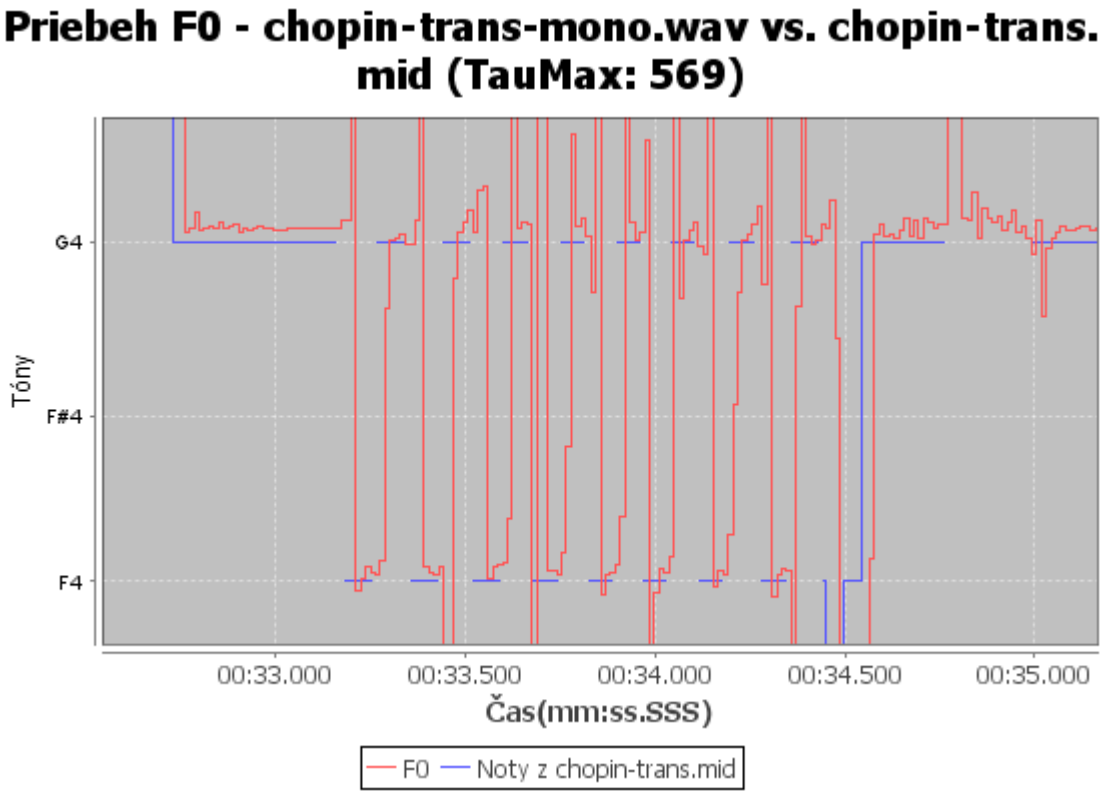
Nástroj Rock guitar má na východzích hodnotách zadefinovaný overdrive, vďaka ktorému dosahuje „drsný“, skreslený zvuk, typický pre rockovú hudbu. Chromatická nahrávka

dosiahla na 30 BPM zhodu až 98,042% (Obrázok 31). Takmer všetky chybné odhady boli iba na prechodoch tónov. Pre 160 BPM nahrávku sa situácia opakovala so zhodou 89,478%.

Nokturno vykázalo zhodu 91,953%, pričom priebeh vyzeral veľmi podobne tomu z nahrávky čistej el. gitary – chyby najme na prechodoch tónov a mordenty a trilky zvládnuté s posunom ~30ms. Detail na trilky je na obrázku (Obrázok 32).



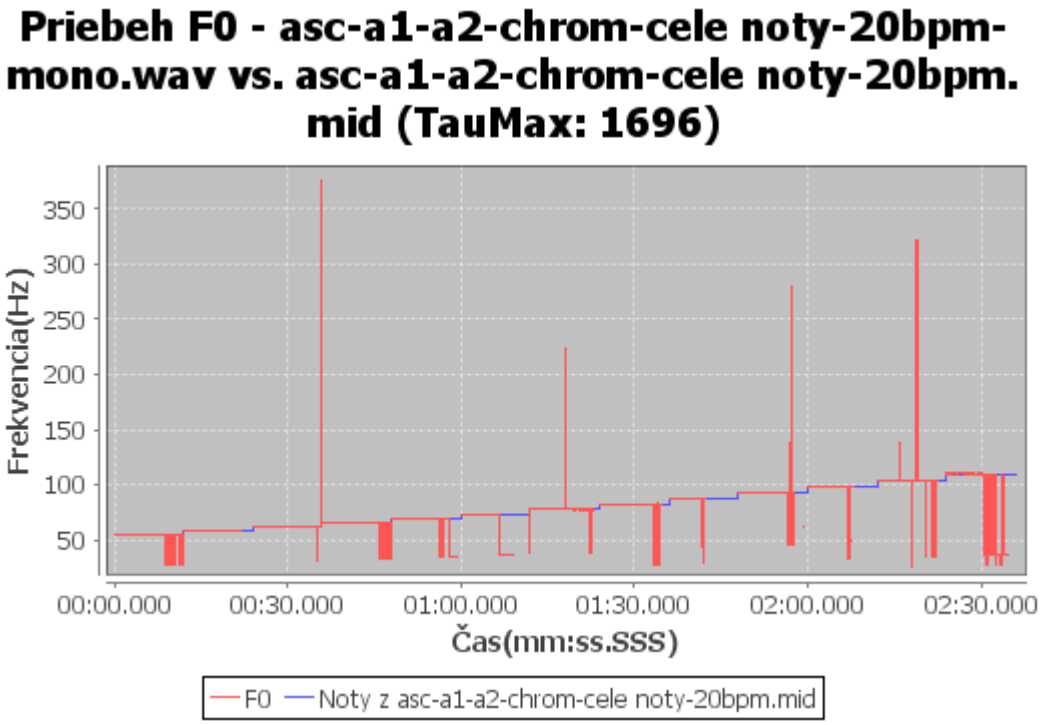
Obrázok 31 Priebeh F_0 na chromatike hranej na el. gitare s overdrive-om



Obrázok 32 Trilky hrané na el. gitare s overdrive-om

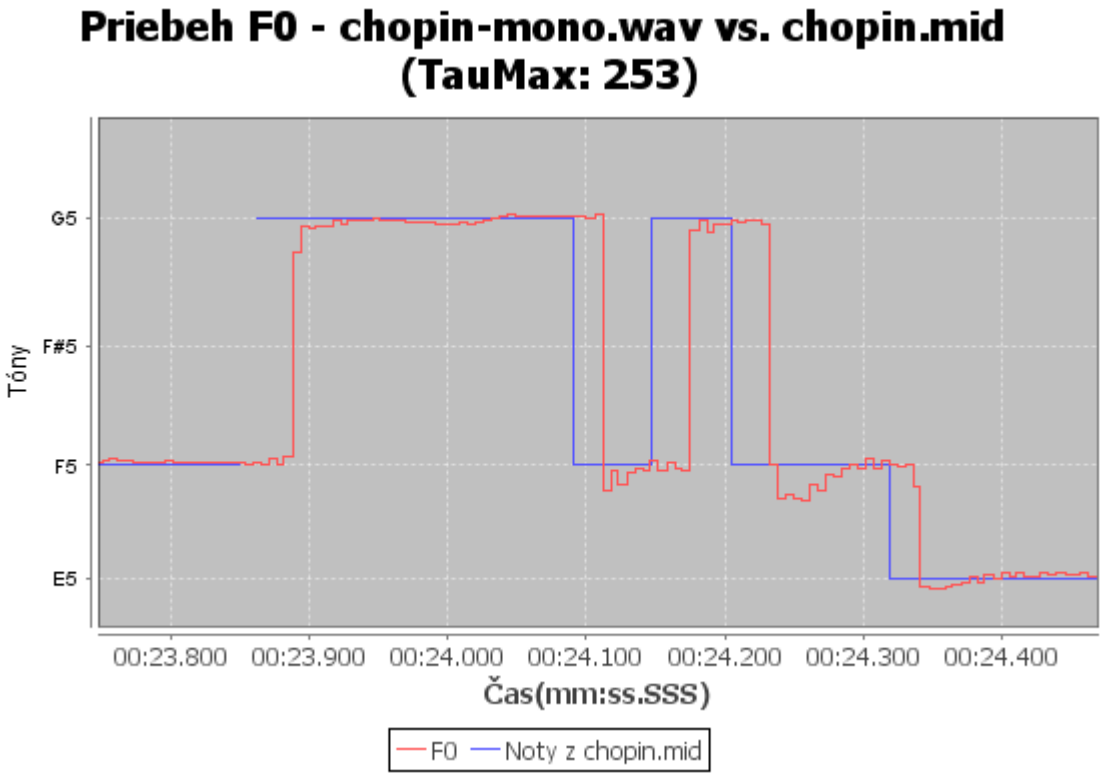
5.2.5 Dodatočné merania

Kvôli tomuto spozorovanému poklesu chýb v prechodoch pri klavírnych chromatikách som ešte vygeneroval jednu nahrávku navyše. V nej bola jedna oktáva (A1-A2), ktorá zhruba odpovedala miestam najväčšieho výskytu týchto chýb. Nastavil som však dĺžku jednotlivých nôt na 12 sekúnd. Takáto dĺžka dáva oveľa väčší priestor jednotlivým notám doznieť. Na základe analýzy tejto nahrávky je možné vidieť, že tieto chyby vznikali skutočne kvôli stretu dvoch F_0 v analyzovanom okne a nie kvôli utlmenému signálu, keďže frekvenčný odhad začal zlyhávať až od zhruba siedmej sekundy hranej noty a vzniká preň skôr tendencia prepadu frekvencií. Väčšina týchto prepadov mala opäť charakter oktávovej chyby. Výsledok analýzy tejto nahrávky je na obrázku (Obrázok 33).



Obrázok 33 Priebeh F_0 pre chromatickú stupnicu A1-A2 v celých notách a 20 BPM na klavíri

Druhé dodatočné meranie, ktoré som spravil, bolo testom, či nastavenie nulovej tolerancie odchýlky finálneho od prvotného odhadu eliminuje efekt posunu F_0 v čase. Nulová tolerancia spôsobí, že algoritmus nehľadá periódu v okolí pôvodného merania ale za finálny odhad prehlási hneď pôvodný. Ako nahrávku som zvolil nokturno hrané na trúbke, tentokrát však s nastavenou 0% odchýlkou. Výsledok (detail mordentu) ukázal, že efekt ostal rovnaký (Obrázok 34). To znamená že na začiatku novej noty je naozaj dominantná frekvencia z dozvuku predchádzajúcej noty, ktorú nájde aj algoritmus. Percentuálna zhoda sa zvýšila len minimálne na 92,807% z pôvodných 92,393%.



Obrázok 34 Mordent hraný na trúbke s nulovou toleranciou odchýlky

6 Diskusia

6.1 Úspešnosť algoritmu

Evalvácia na nahrávkach ukázala, že na úspešnosť algoritmu nemá vplyv dynamika hry. Problémy však môžu v závislosti od nástroja a zvolenej veľkosti okna spôsobovať veľmi krátke noty, ktoré v sú v notových zápisoch zvyčajne zaznamenané ako ozdoby. Pri použití klavíra sa totiž nestihli odhady F_0 ustáliť počas trvania noty na správnej hodnote a tak boli vo výsledku určené chybné tóny. Pre klavírnu hru sa tiež ukázali nedostatky pri detegovaní vysokých tónov, kde je samotná rezonancia nôt príliš rýchlo utlmená, prípadne s ňou silno interferuje hluk mechanických častí klavíra. Bez ohľadu na nastavenia sa tiež ukázalo, že algoritmus často zlyháva pre malý počet odhadov (zvyčajne jeden až dva) v oblasti prechodu jedného tónu do druhého. Vtedy sa fundamentálne frekvencie dvoch tónov stretávajú v jednom okne a algoritmus kvôli ich interferencii často určí chybnú periódu. Táto chyba v prechodoch je pozostatkom z obmedzenia algoritmu pre monofonickú estimáciu. Ďalším z problémov, ktorý ovplyvňoval výsledky evalvácie je fakt, že algoritmus mal tendenciu voliť F_0 predchádzajúceho tónu aj krátky časový úsek (30-60ms) po začatí hrania druhého tónu. Ako sa ukázalo, tento problém neeliminuje ani nastavenie nulovej tolerancie odchýlky (de facto vynechanie posledného kroku – voľby najlepšieho lokálneho odhadu). Algoritmus volí F_0 predchádzajúcej noty jednoducho preto, lebo je v mieste analýzy dominantná – ak je aj nahrávka monofonická, dozvuk prvej noty môže prevážiť nad začiatkom novej. Tento problém mal za následok vizuálne „posunutie“ frekvenčného priebehu v čase voči anotácii. Posledný problém predstavuje oktávová chyba – aj napriek použitiu viacerých krokov znižujúcich mieru chybovosti autokorelácie algoritmus občas zvolí nesprávny násobok periódy. Pre väčšinu ustálených tónov však algoritmus určil F_0 správne. Percentuálne vyhodnotenie výsledkov evalvácie pre tri druhy nahrávok je v Tab. 2.

Tab. 2 Percentuálna zhoda odhadnutých F_0 s očakávanými tónmi z nahrávok

	Klavír	Tlmená trúbka	Čistá el. gitara	El. gitara s overdrive-om
Chromatika (dĺžka noty 0,375 s)	79,493%	87,032%	88,869%	89,478%
Chromatika (dĺžka noty 2 s)	80,017%	94,392%	89,432%	98,042%
Nokturno op.9 č.2	83,932%	92,393%	92,12%	91,953%

Navzdory tomu, že zvolené nokturno je pôvodne klavírna skladba, dopadli klavírne nahrávky v percentuálnej zhode najhoršie a v priemere dopadla najlepšie el. gitara s overdrive-om. Na tom má zásluhu aj fakt, že väčšia hodnota τ_{max} znamená aj väčšie okno W a tým pádom menej bodov merania (porovnávania zhody). Preto ubúda počet odhadov na ustálených tónoch, ktoré sú vo väčšine prípadov správne (naprieč všetkými testovanými nástrojmi), v pomere k chybným odhadom na prechodoch tónov.

Tieto problémy by prekážali v prípade použitia na automatizovanú transkripciu. Pri krátkych notách by totiž výsledkom bola buď nesprávna nota, alebo by sa ňou spôsobená zmena odhadu stratila vo fluktuácii priebehu F_0, prípadne by mohla byť považovaná za chybný odhad. Tak isto pri občasnom výskyte oktávových chýb by bola nutná manuálna úprava človekom. Zlé spracovávanie rýchlo sa utlmujúcich nôt a citlivosť na okolitý hluk pozorovaný pri klavírnych nahrávkach by zase limitovali bezpečný rozsah pre transkripciu. Samotné chyby na prechodoch tónov veľký problém nepredstavujú, keďže ich trvanie bolo väčšinou v rámci dvoch chybných odhadov. Takáto krátka fluktuácia by sa pri prepise nemusela brať v úvahu, čo ale na druhú stranu ešte viac sťažuje detekciu krátkych nôt a ozdôb. Efekt posunu taktiež nepredstavuje veľké riziko. Pri prepise do nôt je skôr dôležitejšia korektná dĺžka trvania než presný začiatok tónu, aby sa mohla správne určiť dĺžka noty. Navyše rôzne nástroje majú rôzny charakter nábehu tónu, ktorý by sa v záujme časovo presnej transkripcie musel brať v úvahu. Keďže sa algoritmus podarilo implementovať dostatočne rýchlo na použitie v reálnom čase, jeho použitie ako pomocného softvéru na ladenie sa zdá byť vhodnejšie, nakoľko toto použitie nepredpokladá výskyt veľmi krátkych tónov.

6.2 Efektívnosť implementácie

Rýchlosť implementácie algoritmu je náročné určiť z viacerých dôvodov. Prvý je prítomnosť JIT kompilácie v Jave a automatickej alokácie pamäte. Dĺžka analýzy sa tak môže meniť v závislosti od toho, či je analýza spustená tzv. „na studeno“ – bez predchádzajúcej analýzy, alebo už pred ňou nejaká analýza prebehla. Najdlhší výpočet teda zvyčajne býva pri analýze na studeno. Druhým je veľký rozsah nastavení, ktoré veľmi razantne menia dĺžku trvania analýzy. Dĺžka narastá so zväčšujúcim sa τ_{max}, zväčšujúcou sa povolenou odchýlkou a v niektorých prípadoch môže narásť aj pri zväčšujúcom sa Thresholde. Keďže algoritmus predčasne končí jednu svoju iteráciu v prípade že nájde vyhovujúceho kandidáta, dĺžka závisí aj od obsahu samotnej nahrávky. Dá sa však povedať, že efektivita implementácie je dostatočná aj pre použitie v reálnom čase, lebo

algoritmu trvala na testovanej konfigurácii analýza nahrávky bežne zhruba jednu pätinu z dĺžky nahrávky. Ako ilustračný príklad uvediem 78 sekundovú chromatiku hranú na gitare. Tú sa za použitia nastavení uvedených v evalvácii podarilo zanalyzovať za ~14 sekúnd na 2 jadrovom CPU Intel Core i3-2310M@2,1GHz so 64 bitovou Javou.

7 Záver

Cieľom tejto práce bolo implementovať a otestovať funkčnosť algoritmu YIN pri detegovaní tónov v monofonických nahrávkach hudobných nástrojov.

Pre porovnanie a kategorizovanie algoritmu som najskôr uviedol prehľad základných metód estimácie F_0 a následne som popísal samotný algoritmus YIN.

Algoritmus som implementoval v jazyku Java, výsledky som zobrazoval za pomoci knižnice JFreeChart a nahrávky som vygeneroval z MIDI notácie pomocou VST nástrojov, čo umožnilo presnú anotáciu.

Algoritmus sa ukázal byť vo väčšine prípadov funkčný, avšak problémy algoritmu zistené pri evalvácii môžu v závislosti od požadovaného časového rozlíšenia predstavovať problém pri automatizovanej transkripcii. Keďže sa ale algoritmus podarilo implementovať dostatočne rýchlo na použitie v reálnom čase, jeho využitie ako pomocného softvéru na ladenie nástrojov sa zdá byť vhodnejšie. Napriek tomu je treba aj pri tomto použití rátať s možnou oktávovou chybou.

Zoznam bibliografických odkazov

1. SCHWARTZ, David A.; PURVES, Dale. Pitch is determined by naturally occurring periodic sounds. *Hearing research*, 2004, 194.1: 31-46. Dostupné na internete: http://www.sciencedirect.com/science/article/pii/S037859550400139X
2. MATHER, George. *Foundations of perception*. Taylor & Francis, 2006. Dostupné na internete: http://books.google.sk/books?id=LYA9faq3lt4C&pg=PA125&redir_esc=y#v=onepage&q&f=false
3. DE CHEVEIGNÉ, Alain; KAWAHARA, Hideki. YIN, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, 2002, 111.4: 1917-1930. Dostupné na internete: http://audition.ens.fr/adc/pdf/2002_JASA_YIN.pdf
4. STEMPLE, Joseph C.; GLAZE, Leslie E.; GERDEMAN, Bernice K. *Clinical voice pathology: Theory and management*. Cengage Learning, 2000. Dostupné na internete: http://www.google.sk/books?hl=sk&lr=&id=y2UboKGPCNIC&oi=fnd&pg=PR9&dq=Stemple,+J.+C.,+Glaze,+L.+E.,+Gerdeman-Klaben,+B.,+Clinical+Voice+Pathology,+Theory+and+Management&ots=0Z27CfxqHf&sig=XBBL5FZhZgWBKMi1vJ9aIT62GHQ&redir_esc=y#v=onepage&q&f=false
5. MCLEOD, Philip. Fast, accurate pitch detection tools for music analysis. *Academisch proefschrift, University of Otago. Department of Computer Science*, 2009. Dostupné na internete: http://miracle.otago.ac.nz/tartini/papers/Philip_McLeod_PhD.pdf
6. VERTELETSKAYA, Ekaterina; SAKHNOV, Kirill; SIMAK, B. Pitch detection algorithms and voiced/unvoiced classification for noisy speech. In: *Systems, Signals and Image Processing, 2009. IWSSIP 2009. 16th International Conference on*. IEEE, 2009. p. 1-5. Dostupné na internete: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=05367778
7. GERHARD, David. *Pitch extraction and fundamental frequency: History and current techniques*. Regina: Department of Computer Science, University of Regina, 2003. Dostupné na internete: http://audio-fingerprint.googlecode.com/svn-history/r62/trunk/referencias/2003-06.pdf

8. BACHU, R. G., et al. Separation of Voiced and Unvoiced using Zero crossing rate and Energy of the Speech Signal. In: *American Society for Engineering Education (ASEE) Zone Conference Proceedings*. 2008. p. 1-7. Dostupné na internete: http://spoken-number-recognition.googlecode.com/svn/trunk/docs/End%20point%20detection/ASEE12008_0044_paper.pdf
9. Wikimedia Commons, Zero crossing.svg. Dostupné na internete: http://en.wikipedia.org/wiki/File:Zero_crossing.svg
10. NORTON, Michael Peter; KARCZUB, Denis G. *Fundamentals of noise and vibration analysis for engineers*. Cambridge university press, 2003. Dostupné online: http://books.google.sk/books?hl=sk&id=jDeRCSqtev4C&q=power#v=snippet&q=power%20cepstrum&f=false
11. AKAY, Metin. *Biomedical signal processing*. Academic Press, 2012. Dostupné na internete: http://www.google.sk/books?hl=sk&lr=&id=wr8BjI7tBv4C&oi=fnd&pg=PP1&dq=akay+cepstrum&ots=FtP_0OFnc_&sig=RJjGoKfiEykToTi2WnPGmwFNv8k&redir_esc=y#v=onepage&q=akay%20cepstrum&f=false ň
12. OpenJDK 7u60, ArrayList.java. Dostupné na internete: http://hg.openjdk.java.net/jdk7u/jdk7u60-dev/jdk/file/1c1b897a12ed/src/share/classes/java/util/ArrayList.java
13. Standard MIDI-File Format Spec. 1.1. Dostupné na internete: http://www.music.mcgill.ca/~ich/classes/mumt306/midiformat.pdf
14. The Java Language Specification, Jave SE 7 Edition. Dostupné na internete: http://docs.oracle.com/javase/specs/jls/se7/html/jls-15.html#jls-15.22.1
15. International Music Score Library Project, Nocturnes, Op.9 (Chopin, Frédéric). Dostupné na internete: http://imslp.org/wiki/Nocturnes,_Op.9_%28Chopin,_Fr%C3%A9d%C3%A9ric%29

Prílohy

Príloha A: Bakalárska práca v elektronickej podobe (na CD médiu v koreňovom adresári súbor Bakalárska práca.pdf)

Príloha B: Používateľská príručka aplikácie (na CD médiu v koreňovom adresári súbor Príručka.pdf)

Príloha C: Skompilovaná aplikácia (na CD médiu v adresári Aplikácia (spustiteľný súbor YIN_-_pitch_detection_algorithm.jar))

Príloha D: Zdrojové kódy aplikácie vo forme NetBeans projektu (na CD médiu v adresári Zdrojové kódy)

Príloha E: Dokumentácia zdrojových kódov vygenerovaná nástrojom Javadoc (na CD médiu v adresári Javadoc)

Príloha F: Hudobné nahrávky vo formátoch WAV, SMF a SIB (na CD médiu v adresári Hudobné nahrávky)