VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE TÓNŮ Z AUDIO SIGNÁLU METODOU ZPRACOVÁNÍ SIGNÁLU

DETECTION OF TONES FROM AUDIO SIGNAL BY SIGNAL PROCESSING METHODS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **LUKÁŠ KRISTEK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. KAREL VESELÝ, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

157933

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Kristek Lukáš**
Program:	Informační technologie
Název:	**Detekce tónů z audio signálu metodou zpracování signálu**
Kategorie:	Zpracování signálů
Akademický rok:	2023/24

Zadání:

1. seznamte se s problematikou detekce tónu v audio signálu
2. implementujte detektor tónů pomocí metody zpracování signálu s vhodným časovým oknem
3. experimentujte s komponentami a hyperparametry: pre-emfáze, Hammingovo okno, DFT vs. Konstantní Q-transformace, okénková funkce, audio signál
4. proveďte objektivní hodnocení přesnosti systému metodou F1-score na vhodných testovacích datech

Literatura:

- Beauchamp, J.: Analysis, synthesis, and perception of musical sounds the sound of music. New York: Springer, 2007, ISBN 0-387-32496-8.
- Jakub Hyrák: Přesná detekce základního tónu hudebních nástrojů. Bakalářská práce, FIT VUT, 2013.
- Nodžák Petr: Automatické rozpoznání akordů pomocí hlubokých neuronových sítí. Diplomová práce, FIT VUT, 2020.
- Petr Marciniak: Ve stopách Leoše Janáčka - převod řeči na hudbu. Bakalářská práce, FIT VUT, 2010.

Při obhajobě semestrální části projektu je požadováno:
1.-2.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Veselý Karel, Ing., Ph.D.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2023
Termín pro odevzdání:	31.7.2024
Datum schválení:	9.11.2023

Abstrakt

Tato bakalářská práce se zabývá detekcí tónů z audio signálu pomocí metod zpracování signálu a vyhodnocením jejich úspěšnosti F1 metrikou. Použité metody zahrnují Fourierovu transformaci (FT) a Konstantní Q-transformaci (CQT). Jako rozšíření práce experimentujeme i s vlastní metodu založenou na detekci přítomnosti alikvotních tónů, která interně používá Fourierovu transformaci. Každá z metod má specifický postup pro finální detekci tónů. Práce se zaměřuje na implementaci těchto postupů a porovnání výsledků jednotlivých metod.

Abstract

This bachelor's thesis focuses on tone detection from audio signals using signal processing methods and evaluating their performance with F1 metric. The methods used include the Fourier transform (FT), Constant Q-transform (CQT). As an extension of the thesis we experiment with our own method based on the presence of harmonic tones, which internally uses the Fourier transform. Each method has a specific procedure for the final decision of a tone detection. The thesis is focused on the implementation of these procedures and the comparative evaluation of the individual methods.

Klíčová slova

Detekce tónů, Fourierova transformace, konstantní Q-transformace, alikvotní tóny, preemfáze, okénkové funkce, F1-score.

Keywords

Tone detection, Fourier transform, constant Q-transform, overtones, pre-emphasis, window functions, F1-score.

Citace

KRISTEK, Lukáš. *Detekce tónů z audio signálu metodou zpracování signálu.* Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Karel Veselý, Ph.D.

Detekce tónů z audio signálu metodou zpracování signálu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Karla Veselého, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Lukáš Kristek
30. července 2024

Poděkování

Tímto bych rád poděkoval vedoucímu mé bakalářské práce panu Ing. Karlovi Veselému, Ph.D. za věnovaný čas a poskytnutí odborné pomoci. Zejména si vážím jeho podrobných zpětných vazeb, návrhů na změny a možnosti realizovat vlastní téma bakalářské práce. Dále také všem, kteří se přičinili k dokončení mé bakalářské práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Abychom mohli zahrát skladbu, jejíž noty neznáme, musíme tyto noty nejprve detekovat. Zkušený muzikant dokáže jednotlivé tóny rozpoznat podle sluchu. Potřebuje k tomu dobrý sluch, spoustu hudební teorie a také zkušenosti.

Detekce tónů z audio signálu metodou zpracování signálu není jednoduchý úkol. Každá zahraná nota v audio nahrávce způsobí přítomnost základní frekvence noty, ale také frekvence alikvotních tónů. Audio může obsahovat několik hraných not zároveň hraných několika různými nástroji a k tomu například bubny a zpěv. Lidské ucho dokáže tyto složky dobře rozlišit a zaměřit se tak například na jeden nástroj. Při zpracovávání audia lze použít nástroje, které umí jednotlivé složky rozdělit, složky signálu ale nejde rozdělit perfektně a obsahují spoustu šumu. Tato práce se zabývá detekcí tónů z čistého audia, které bylo syntetizováno z MIDI souborů pomocí zvukové banky piána.

Cílem mé bakalářské práce bylo experimentovat s metodami zpracování signálů. Především s diskrétní Fourierovou transformací (konkrétněji STFT) a konstantní Q-transformací. Také použití okénkových funkcí, preemfáze a objektivní ohodnocení výsledků metrikou F_1 score. Po použití těchto metod jsem implementoval i vlastní metodu, která se opírá o přítomnost alikvotních tónů. Alikvotní tóny produkuje většina hudebních nástrojů, dokonce i lidský hlas.

Práce je strukturována do několika kapitol. První kapitola je o charakteristice zvukového signálu. Seznamuje čtenáře s podobou audio signálu. Kapitola se také věnuje základům hudby, s jejich porozuměním lze lépe předpokládat obsah audia a které frekvence mohou být v audiu přítomny. Druhá kapitola se věnuje metodám používaným pro zpracování signálů. Jedná se o metody Fourierovy transformace a konstantní Q-transformace. Každá z těchto metod je popsána, poté je pomocí ní provedena implementace detekce tónů a výsledek je vyhodnocen metrikou F_1 score. Třetí kapitola se zabývá implementací, vytvořením vlastní metody k detekci, jednoduchou aplikací a výpočtem metriky F_1 score.

Kapitola 2

Charakteristika zvukového signálu

2.1 Zvuk

Zvuk je mechanické vlnění, které se šíří prostřednictvím média. Přenosové médium může mít různé skupenství, například vzduch, voda nebo pevný vodič. Lidské ucho vnímá zvuky v rozsahu přibližně od 20 Hz do 20 000 Hz. S věkem se horní hranice tohoto rozsahu obvykle snižuje.

Převod frekvence na vlnovou délku závisí na rychlosti zvuku v daném médiu. Vlnová délka, označovaná jako λ, je dána vzorcem:

$$\lambda = \frac{v}{f} \tag{2.1}$$

kde:

- v: rychlost zvuku,
- f: frekvence.

Pro zvukové vlny šířící se vzduchem platí, že vyšší teplota znamená vyšší rychlost zvuku. Pro suchý vzduch o pokojové teplotě 20°C je rychlost zvuku přibližně 343 m/s. Za těchto podmínek lze rozsah 20 – 20 000 Hz přepočítat na vlnovou délku 17,15 – 0,0171 m.

2.2 Reprezentace audia

Reprezentace audia je jedním ze základních konceptů v digitálním zpracování zvuku, který se zabývá převodem analogových zvukových signálů na digitální formát a jejich následným zpracováním. Tato kapitola se zaměřuje na základní principy reprezentace audia, včetně digitálního a analogového zvuku, vzorkování, kvantizace a formátů zvukových souborů.

2.2.1 Analogové a digitální audio

Analogový zvuk je kontinuální signál, který má nekonečně mnoho hodnot v čase. Příkladem analogového zvuku může být zvuková vlna zachycená mikrofonem nebo záznam na gramofonové desce. Na rozdíl od toho je digitální zvuk reprezentován diskrétními hodnotami, které jsou získávány vzorkováním a kvantizováním.

Digitální audio je reprezentováno pomocí číselných hodnot, pravidelně zachycených na vzorkovací frekvenci a hodnoty reprezentují amplitudu signálu v jednotlivých vzorcích. Analogové audio by se dalo reprezentovat pomocí kontinuální funkce, ale tato metoda je vhodná

pouze pro pravidelné signály, jako je sinusová vlna. Pro audio zachycené v přirozeném prostředí, ve kterém se nachází šum, je vhodné audio převést na digitální.

Převod analogového signálu na digitální provádí analogově-digitální převodník (ADC) a pro převod analogového signálu na digitální formát se používá digitálně-analogový převodník (DAC). Tyto převodníky jsou součástí mnoha zařízení zpracovávajících zvuk, včetně počítačů, mobilních telefonů, audio zařízení a dalších.

Digitální audio slouží pro přenos a zpracování zvuku, zatímco analogový zvuk je ten, který slyšíme nebo který přenášíme vodičem.

2.2.2 Vzorkování

Vzorkování a kvantizace jsou dva základní procesy při digitalizaci analogového zvuku Princip vzorkování spočívá v pravidelném měření hodnot analogového signálu v časových intervalech, známých jako vzorky. Tento proces převádí kontinuální analogový signál na diskrétní digitální formát.

Frekvence vzorkování, vyjádřená v hertzích, určuje, jak často je signál vzorkován za sekundu. Jako standardní frekvence vzorkování pro kvalitní zvukové nahrávky se používá 44.1 kHz nebo 48 kHz, což znamená, že jsou prováděny 44 100 nebo 48 000 vzorků za sekundu. Každý vzorek je kvantizován na číselnou hodnotu.

Nyquistův-Shannonův teorém stanoví, že nejvyšší frekvence obsažená v analogovém signálu (nazývaná Nyquistova frekvence) musí být nejméně dvojnásobkem frekvence vzorkování, aby mohla být analogová data správně rekonstruována z digitální podoby. Pokud je nejvyšší frekvence signálu 20 kHz, je třeba vzorkovat signál s frekvencí nejméně 40 kHz, aby nedošlo ke ztrátě informací.

Vzorkování může být prováděno různými zařízeními, jako jsou již zmíněné analogově-digitální převodníky. ADC převádí kontinuální analogový signál na diskrétní číselné hodnoty, které jsou reprezentovány binárně. Tyto hodnoty jsou pak uloženy jako digitální zvukový soubor, ve kterém je audio lehce přenositelné a lze jej dále analyzovat diskrétními metodami.

Vzorkování je základním krokem při digitalizaci zvuku a je důležité pro zachování kvality a přesnosti původního analogového signálu.

2.2.3 Kvantizace

Vzorkování může být provedeno s různými rozlišeními, což ovlivňuje přesnost reprezentace analogového signálu v digitálním formátu. Tento proces převodu vzorků se nazývá *kvantizace* Rozlišení kvantizace je obvykle udáváno v bitech na vzorek. Nejpoužívanější velikosti vzorků jsou 16-bit, 24-bit a 32-bit. Čím vyšší je rozlišení, tím více hodnot může být reprezentováno a tím přesnější je digitální reprezentace signálu.

Kombinace vzorkovací frekvence a rozlišení určuje kvalitu digitální reprezentace původního signálu. Pro analýzu signálu a pro zpětnou rekonstrukci analogového signálu je potřeba použít dostatečnou kvalitu.

Použití příliš vysoké kvality není vždy potřebné, protože rozdíl v audiosignálu přestává být rozpoznatelný a pouze zatěžuje prostředky.

2.2.4 Formáty zvukových souborů

Existuje mnoho různých formátů zvukových souborů používaných k ukládání digitálního audia. Mezi nejběžnější formáty patří WAV, MP3, AAC, FLAC a OGG. Tyto formáty se liší ve formě komprese, kvalitě zvuku a podpoře funkcí jako jsou metadata a vícekanálový zvuk.

2.3 Noty a frekvence

V hudební tvorbě se standardně používá klaviatura klavíru pro sázení nástrojů a syntezátorů v čase. Každá klávesa jednoznačně definuje notu a tím i základní frekvenci, kterou bude daný nástroj znít.

Standardní klaviatura obsahuje 88 kláves. Klávesy jsou seřazeny podle frekvence od nejnižší po nejvyšší. Noty jsou seskupeny do oktáv. Každá oktáva obsahuje 12 not: C, $C^\sharp$, D, $D^\sharp$, E, F, $F^\sharp$, G, $G^\sharp$, A, $A^\sharp$, B. Jako referenční bod, například pro ladění, se používá nota A ze 4. oktávy (dále jako A_4).

Aby se dosáhlo zmíněných 88 kláves, oktávy se musí opakovat. Začíná se indexem 0 a pokračuje až po oktávu s indexem 8. První použitou notou není C_0, ale A_0 s frekvencí **27.5 Hz**. Nejvyšší je C_8 s frekvencí **4186 Hz**.

Důvodem rozřazení do oktáv je skutečnost, že noty znějí podobně, i když jsou z jiné oktávy. Je to dáno jejich frekvencí. Každá oktáva obsahuje noty s dvojnásobnou frekvencí oproti oktávě předchozí. Například nota A_5 má frekvenci 880 Hz, A_4 440 Hz, A_3 220 Hz, atd. Toto platí pro všechny noty, viz tabulka 2.1.

Nota	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
C		32.70	65.41	130.81	261.63	523.25	1046.50	2093.00	4186.01
$C^\sharp$/$D^\flat$		34.65	69.30	138.59	277.18	554.37	1108.73	2217.46	
D		36.71	73.42	146.83	293.66	587.33	1174.66	2349.32	
$D^\sharp$/$E^\flat$		38.89	77.78	155.56	311.13	622.25	1244.51	2489.02	
E		41.20	82.41	164.81	329.63	659.25	1318.51	2637.02	
F		43.65	87.31	174.61	349.23	698.46	1396.91	2793.83	
$F^\sharp$/$G^\flat$		46.25	92.50	185.00	369.99	739.99	1479.98	2959.96	
G		49.00	98.00	196.00	392.00	783.99	1567.98	3135.96	
$G^\sharp$/$A^\flat$		51.91	103.83	207.65	415.30	830.61	1661.22	3322.44	
A	27.50	55.00	110.00	220.00	440.00	880.00	1760.00	3520.00	
$A^\sharp$/$B^\flat$	29.14	58.27	116.54	233.08	466.16	932.33	1864.66	3729.31	
B	30.87	61.74	123.47	246.94	493.88	987.77	1975.53	3951.07	

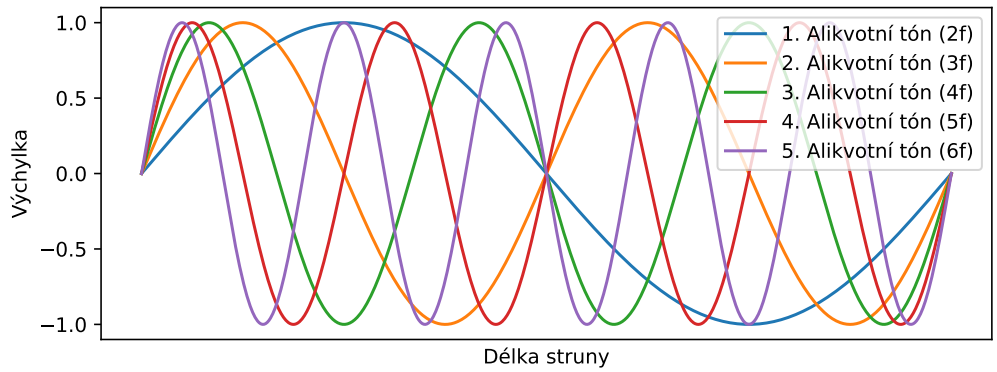
Tabulka 2.1: Frekvence používaných not, rozřazeno do oktávách

MIDI soubory používají jeden bajt k mapování tónů. To umožňuje identifikaci až 128 různých tónů. Podrobnější tabulka s možnými tóny je uvedena v článku [1]. Tóny, které se nenacházejí na standardní klaviatuře, se používají zřídka, a jejich detekcí jsem se nezabýval.

2.4 Alikvotní tóny

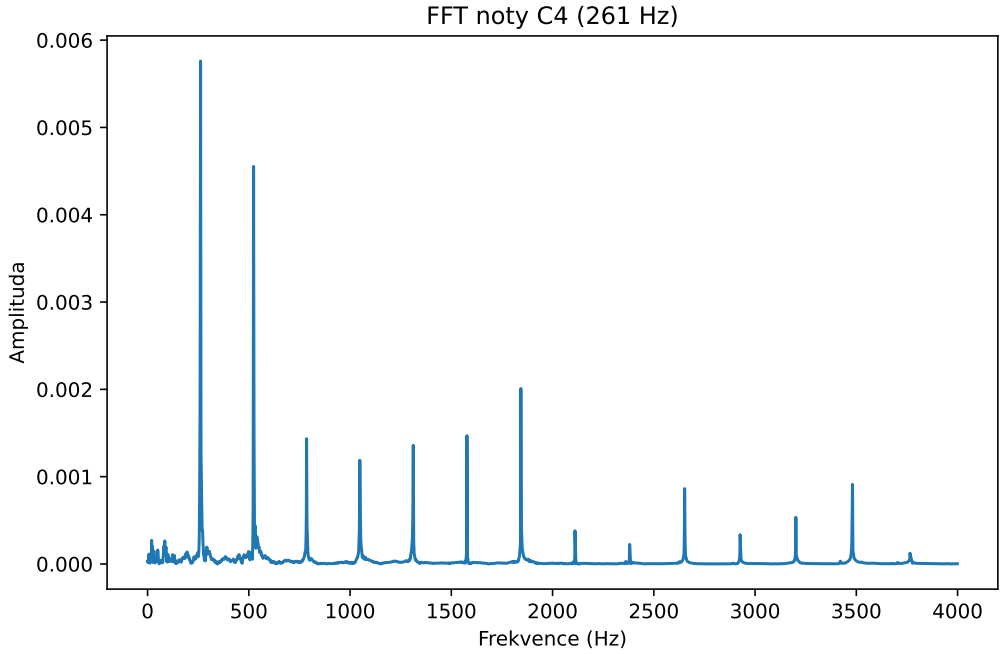
Každý tón má svou základní frekvenci, ale zároveň vznikají i *alikvotní tóny*, které jsou násobky základní frekvence. Intenzita těchto frekvencí se liší mezi jednotlivými nástroji. Díky alikvotním tónům má každý nástroj svůj unikátní zvuk. Podle sluchu jsme schopni rozpoznat, jestli byla nota zahrána na kytaru, piano, housle, nebo jiný hudební nástroj a to i přesto, že na nástrojích zahrajeme stejnou základní frekvenci, tedy stejnou notu.

Alikvotní tóny vznikají například při kmitání struny. Když se struna rozechvěje, nevibruje pouze v celku (což by produkovalo pouze základní tón), ale také v různých částech. Po celé délce struny vznikají různé uzly (body, které zůstávají relativně nehybné) a maxima (body maximálního pohybu). Základní tón odpovídá vibraci celé délky struny mezi jejími konci. První harmonický (první alikvotní) tón odpovídá vibraci struny v polovině její délky, druhý harmonický tón ve třetině atd. Tyto různé módy kmitání generují vyšší frekvence, které se označují jako alikvotní tóny, viz obrázek 2.1.



Obrázek 2.1: Vizualizace alikvotních tónů - vibrující struna

Na obrázku 2.2 je zobrazena Fourierova transformace krátké nahrávky noty C_4 zahrané na piano. Základní frekvence této noty je přibližně 261 Hz. Intenzita vyšších alikvotních tónů obecně klesá. Konkrétní amplituda ostatních frekvencí závisí na charakteristikách nástroje a stylu úderu do struny. Přítomnost alikvotních tónů je viditelná na obrázku 2.2.



Obrázek 2.2: FFT audia obsahující zahranou notu C_4

Alikvotní tóny nevznikají pouze u strunných nástrojů, ale téměř u všech hudebních nástrojů. U dechových nástrojů vznikají prostřednictvím vibrací vzduchového sloupce uvnitř nástroje. I lidský hlas je bohatý na alikvotní tóny, které vznikají harmonickými frekvencemi vibrací hlasivek.

Alikvotní tóny jsou tedy základem v hudbě. Jejich přítomnost může komplikovat detekci not při klasické analýze. Při použití metod jako Fourierova transformace nejsou noty vždy jednoznačně viditelné. Z obrázku 2.2, kde hraje pouze jedna nota, je vidět mnoho frekvencí, které by neměly být detekovány jako samostatné noty. U složitějších audio signálů s více notami je obtížné rozlišit základní frekvence od alikvotních tónů. Například při hraní oktávy, jako jsou noty C_4 a C_5, může být nota C_5 překryta alikvotními tóny noty C_4. Jedním z možných řešení může být ignorování noty C_5, protože její přítomnost přidává pouze barvu notě C_4. Lze tedy říci, že každá nota má svoji důležitost, která vychází z hudební teorie 2.6.

2.5 Ladění

Ladění not je proces, při kterém je každé notě přiřazena její frekvence. Pro vznik harmonie je důležitý poměr hodnot mezi notami. O harmonii podrobněji pojednává kapitola 2.6.3. Nejvýznamnější harmonické poměry základních frekvencí dvou not zahrnují: oktávu (2:1), kvintu (3:2), kvartu (4:3)...

Problém ladění spočívá v tom, že není možné naladit všechny noty tak, aby byly přesně v harmonickém poměru. Existují dva hlavní přístupy, jak tento problém řešit:

- Naladit všechny noty rovnoměrně a tím ztratit přesný harmonický poměr.
- Udržet přesný harmonický poměr u not v rámci jedné tóniny a přijmout velkou nepřesnost mezi ostatními notami.

Existuje několik typů ladění, z nichž každé má své vlastní charakteristiky a aplikace. Rovnoměrně temperované ladění je nejběžnější, protože díky rovnoměrnému kroku mezi notami je nejuniverzálnější. Dalším významným typem je Pythagorejské ladění.

2.5.1 Rovnoměrně temperované ladění

Rovnoměrně temperované ladění je nejčastěji používaný systém ladění not v hudbě. Je založen na rozdělení oktávy na 12 rovnoměrných intervalů, z nichž každý odpovídá polotónu. Tento systém umožňuje snadnou modulaci mezi různými tóninami a hraní ve všech klíčích bez nutnosti přeladění nástroje.

Frekvence jednotlivých not v rovnoměrně temperovaném ladění se vypočítávají podle vzorce, který rozděluje oktávu na 12 polotónů. Výpočet zahrnuje použití umělého čísla, známého jako *temperovací krok*.

Vzorec pro výpočet frekvence f jednotlivých not v rovnoměrně temperovaném ladění:

$$f = f_0 \times 2^{\frac{n}{12}} \tag{2.2}$$

kde:

- f_0: frekvence referenční noty, nejčastěji A_4 s frekvencí 440 Hz,
- n: pořadové číslo noty v temperované stupnici. Například 0 pro referenční notu A_4, 1 pro notu o půltón vyšší, -1 pro notu o půltón nižší,
- 12 je počet položek v temperované stupnici, tedy počet not v oktávě.

Rovnoměrně temperované ladění umožňuje hudebníkům snadno hrát v různých klíčích a transponovat hudbu mezi různými nástroji bez nutnosti změny ladění. Tento systém je využíván ve všech moderních klávesových nástrojích, jako je klavír a elektronické klávesy, stejně jako v dalších nástrojích s pevně stanovenými frekvencemi not, jako jsou kytara nebo housle.

2.5.2 Pythagorejské ladění

Pythagorejské ladění je starověký systém ladění not, který vychází z pythagorejské matematiky a starověké hudební teorie. Tento systém definuje intervaly mezi jednotlivými notami na základě poměru frekvencí, který vychází z harmonických vztahů mezi tóny.

Základem pythagorejského ladění je rozdělení oktávy na 12 intervalů, přičemž každý interval odpovídá různým poměrům frekvencí. Tyto poměry jsou odvozeny z harmonického řádu, což jsou poměry frekvencí založené na prvních několika harmonických tónech. Typickým poměrem v pythagorejském ladění je 3:2. Vyhodnocení not si lze představit jako kruh, který začíná frekvencí f a končí frekvencí $2f$, přičemž noty vznikají postupným násobením původní hodnoty f. Po sobě následující noty jsou ve velmi dobré harmonii. Tyto dvě noty mají základní frekvenci f a $1.5f$. Jejich alikvotní tóny spolu perfektně souzní na frekvenci $3f$.

Pythagorejské ladění vytváří čisté a harmonické intervaly mezi notami, což umožňuje hraní akordů s bohatým a jasným zvukem. Nicméně, tento systém má také své nevýhody, zejména kvůli nedokonalým intervalovým vztahům, které mohou vést k disonanci a neharmonickým akordům při hraní v různých tóninách.

Pythagorejské ladění se obvykle používá v historických interpretacích starověké hudby a ve specifických hudebních stylech, které chtějí dosáhnout čistoty a autentičnosti starověkého zvuku. V moderní hudbě je však tento systém ladění často nahrazen rovnoměrně temperovaným laděním, které poskytuje větší flexibilitu a možnosti modulace.

2.6 Hudební teorie

Většina umělců se při vytváření kompozice řídí hudební teorií. Hudební teorie neurčuje neporušitelná pravidla, ale spíše popisuje, které noty v daném kontextu dává smysl použít.

Pro tématiku detekce tónů z audia lze hudební teorii použít pro vysvětlení existence některých frekvencí (harmonie). Dalším použitím je omezení rozsahu hledaných not (tóniny).

Tato kapitola se zaměří na představení základních hudebních pojmů a jejich vztahu k procesu detekce not. Pochopení hudební teorie přinese náhled do možné struktury hudebních nahrávek a poskytne tak možnost vytvořit efektivnější algoritmy pro detekci tónů.

2.6.1 Tón

Tón je základní stavební blok hudby. Je definován jako zvuk s určitou frekvencí a výškou, viz kapitola 2.3. Ve většině světa jsou tóny označovány písmeny A, B, C, D, E, F, G. U nás se používá *Německé ladění*, které nahrazuje písmeno B písmenem H.

Těchto sedm písmen definuje sedm základních tónů. Zbylých pět tónů tvoří tzv. *půltóny*, které se nachází mezi některými dvojicemi základních tónů. Půltóny se označují pomocí zvýšení ($\sharp$) nebo snížení ($\flat$). Například mezi tónem C a D je půltón, který má dvě možné označení: C$\sharp$ (Cis), nebo D$\flat$ (Des). Tyto dvě označení jsou ekvivalentní a jejich použití závisí na kontextu skladby.

V této práci často používám termíny nota a tón. Termín *nota* používám, když odkazuji na notaci, například výsledky detekce. Termínem *tón* většinou označuji frekvenci dané noty.

2.6.2 Tónina

Tónina určuje, které tóny a akordy budou v určité skladbě považovány za hlavní a jaký bude celkový hudební charakter této skladby.

Existuje celkem 30 tónin, pár z nich je ekvivalentních. Unikátních tónin je 26. Pro každý tón je jedna durová a jedna mollová tónina. Každá tónina vychází z jednoho tónu, podle kterého je i pojmenovaná.

Existuje jednoduchý algoritmus, kterým lze vyhodnotit všechny tóny patřící do tóniny. Začne se na výchozím tónu a každý další tón je ve vzdálenosti dvou kláves (celý tón), nebo jedné klávesy (půltón). Posloupnost tónů a půltónů závisí na tom, zda je tónina durová nebo mollová:

- `dur`: tón, tón, půltón, tón, tón, tón, půltón
- `moll`: tón, půltón, tón, tón, půltón, tón, tón

S informací, ve které tónině nahrávka je, lze omezit počet hledaných tónů z 12 na 7.

2.6.3 Harmonie

Skladby se většinou skládají z melodie a doprovodu. Doprovod má většinou pomalejší průběh, skládá se z nižších tónů a bývá složen z několika not, které dohromady tvoří akordy. Melodie skladby se obecně hraje ve vyšších frekvencích, má rychlejší průběh a často je tvořena jednou notou v jednom čase.

Tuto strukturu lze zjednodušit tak, že skladba obsahuje pouze dvě noty v jednom čase: jednu pro doprovod a druhou pro melodii. Každou z těchto dvou základních not lze rozšířit o další noty, čímž se základní notě přidá barva. Tomuto rozšíření se říká harmonie.

Harmonie pro lidské ucho zní příjemně, protože tyto harmonické obraty mají něco společného. Každá nota produkuje alikvotní tóny 2.4 a harmonické obraty se skládají z not, jejichž alikvotní tóny se na nějakém násobku potkávají.

Například akord C se skládá ze tří not: C, E/E$^\flat$ a G. Tóny C a G spolu souzní již na 2. alikvotním tónu noty C a 1. alikvotním tónu noty G. To, zda je v akordu E nebo E$^\flat$, určuje, zda je akord durový nebo mollový. Od toho se odvíjí vztah k tónům C a G. Mollové kombinace not zní smutně, durové kombinace naopak zní vesele.

Hudebníkovi stačí znalost tóniny a základní noty a pomocí hudební teorie dokáže notu rozšířit do harmonického obratu. Existuje spousta možností, jak harmonický obrat vytvořit, žádná z nich nenaruší původní myšlenku skladby, pouze přidá barvu.

Při detekci je tedy nejdůležitější detekovat tyto (dvě) základní noty.

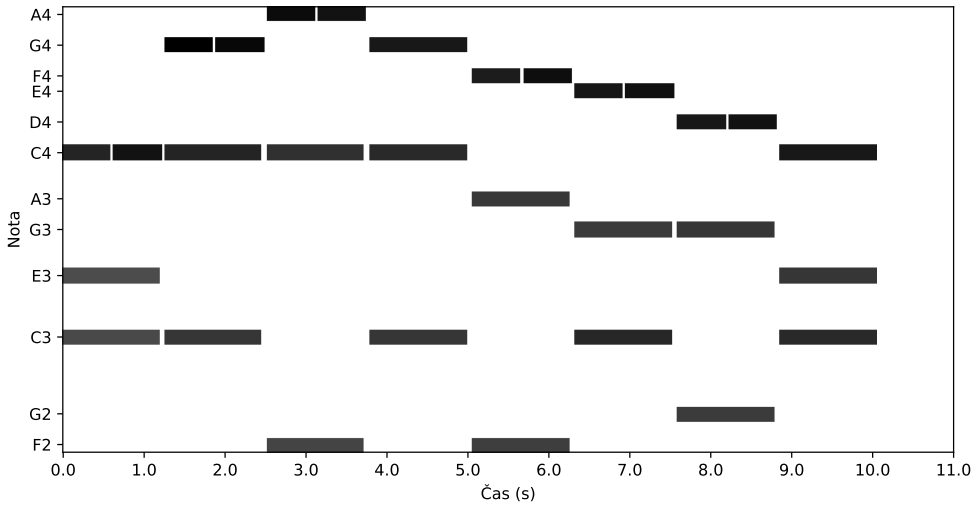
Kapitola 3

Metody analýzy zvukového signálu

Tato kapitola se zabývá popisem a analýzou audia pomocí Fourierovy transformace a konstantní Q-transformace. Každá z metod je nejdříve popsána a následně je provedena implementace pro detekci tónů těmito metodami.

Použití metod je nejdříve provedeno základním použitím funkcí z knihovny librosa. Dokumentace hnihovny librosa [2], použití knihovny [3]. Následují optimalizace a postupy, které jsou potřeba pro převod výsledků do formátu detekovaných not. Nakonec jsou výsledky ohodnoceny pomocí metriky F_1 *score*. Výsledky všech implementovaných metod jsou podrobněji porovnány v závěrečné kapitole 4.3.

Průběžné výsledky jsou zobrazovány na ukázkovém audiu. Jako ukázkové audio jsem použil prvních pár vteřin známé skladby *Twinkle Twinkle Little Star* od *Wolfganga Amadea Mozarta*, viz obrázek 3.1. Audio vychází z MIDI souboru, který byl syntetizován s použitím zvukové banky piana: *Essential Keys-sforzando-v9.6* [4].



Obrázek 3.1: Vizualizace MIDI souboru: Twinkle Twinkle Little Star

Některými z použitých principů se zabývá blog [5], který přibližuje tématiku detekce not.

3.1 Fourierova transformace

Při tvorbě této kapitoly jsem použil zdroje [6] a [7] popisující základy Fourierovy transformace.

Fourierova transformace je základem analýzy dat v několika různých oborech. Slouží k dekompozici složitých signálů na obsah jednotlivých frekvencí. Vstupem je spojitý nebo navzorkovaný signál a vyhodnocuje se podobnost tohoto signálu k obyčejné sinusovce a kosinusovce počítaných frekvencí. Tímto pro každou počítanou frekvenci vznikne výsledek o dvou složkách, reprezentovaný komplexním číslem. Jedna složka, počítaná z kosinusovky, se označuje jako reálná část. Část počítaná ze sinusovky se označuje jako imaginární. Spojením obou složek vznikne komplexní šroubovice. Výpočet Fourierovy transformace pro diskrétní čas:

$$X_k = \sum_{n=0}^{N-1} x[n] e^{-j \cdot 2\pi \frac{kn}{N}} \tag{3.1}$$

kde:

- X_k: k-tý frekvenční koeficient výsledného spektra, jako komplexní číslo,
- N: celkový počet vzorků v čase,
- $x[n]$: n-tý vzorek původního časového signálu,
- k: index frekvenční složky, který nabývá hodnot od 0 do $N-1$,
- n: index vzorku v čase, který také nabývá hodnot od 0 do $N-1$,
- j: imaginární jednotka, kde $i^2 = -1$,
- $\frac{2\pi}{N}$: základní úhlová frekvence.

Výraz $e^{-i\frac{2\pi}{N}kn}$ představuje komplexní exponenciálu, kterou lze podle Eulerovy formule zapsat jako:

$$X_k = \cos\left(\frac{2\pi}{N}kn\right) - i\sin\left(\frac{2\pi}{N}kn\right) \tag{3.2}$$

Z výpočtu lze jednoznačně vidět vznik reálné a imaginární složky. Pro rekonstrukci původního signálu je potřeba vědět obě. Často se zobrazují do komplexní roviny, kde reálná část reprezentuje osu x a imaginární osu y. Přítomnost hledané frekvence určuje amplituda $|X_k|$, v komplexní rovině ji lze vypočítat jako:

$$|X_k| = \sqrt{\Re(X_k)^2 + \Im(X_k)^2} \tag{3.3}$$

Amplituda je tedy vzdálenost od středu. Nabývá větších hodnot, když se původní signál podobá komplexní exponenciále počítané frekvence. Samotná amplituda k rekonstrukci původního signálu nestačí, proto se používá v kombinaci s fází, reprezentovanou jako úhel θ:

$$\theta(X_k) = \tan^{-1}\left(\frac{\Im(X_k)}{\Re(X_k)}\right) \tag{3.4}$$

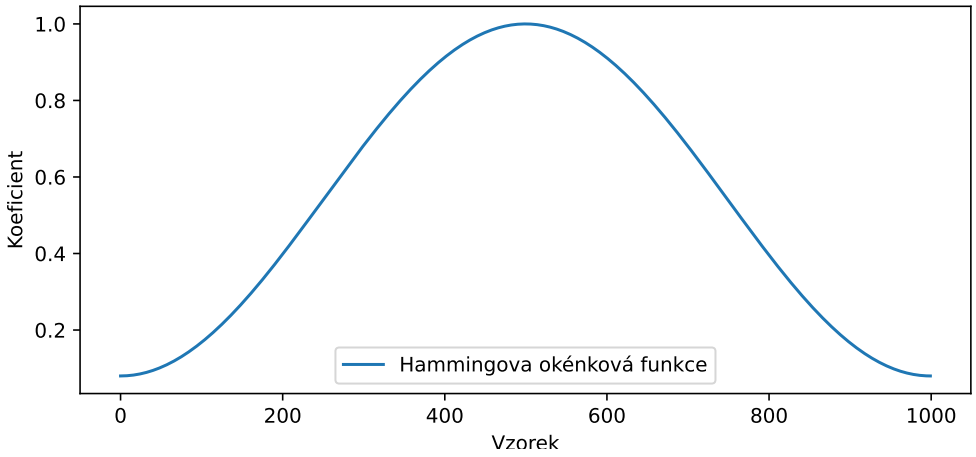
Fáze reprezentuje úhel, neboli posunutí, ve kterém má frekvence největší podobnost, neboli amplitudu, s původním signálem. Použití reprezentace amplitudou a fází se používá hlavně díky jejich intuitivní reprezentaci a jednoduchosti vizualizace amplitudy. Reprezentace komplexním číslem je přímým výstupem Fourierovy transformace a používá se, když není převod potřebný.

3.1.1 Okénové funkce

Delší audio nahrávky se často analyzují po částech. Zvolí se velikost okna, které se bude analyzovat. Kdyby okno přesahovalo mimo audio, chybějící hodnoty lze nahradit nulami. Nuly na výsledek nemají vliv, simulují zmenšení okna.

Audio se postupně analyzuje s konstantním krokem, v knihovnách pro výpočty Fourierovy transformace se často nazývá *hop length*. O hodnotu kroku se okno postupně posouvá až po konec audia. Je vhodné zvolit velikost kroku menší, než velikost okna, například knihovna *librosa* [2] používá jako výchozí hodnotu čtvrtinu délky okna.

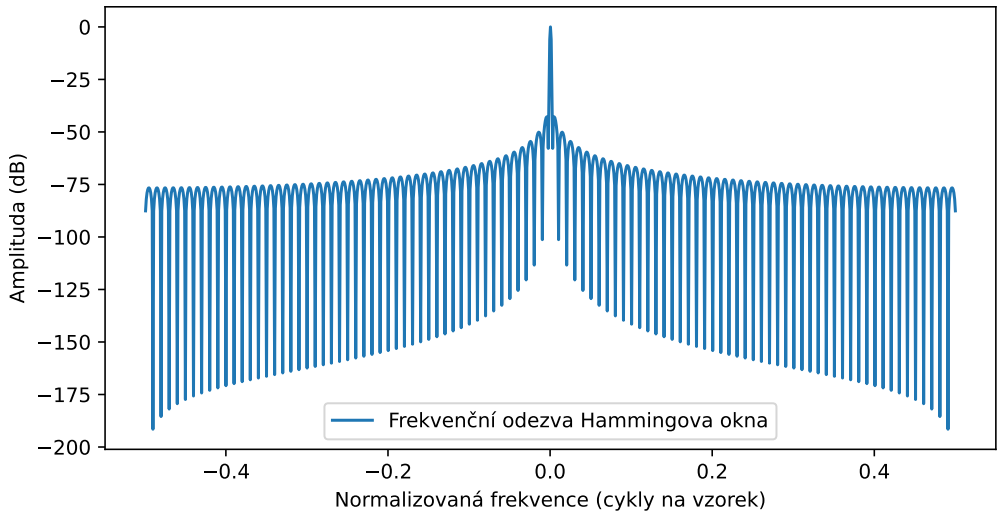
Při periodizaci audia do oken, na krajích oken vznikne skok, který může analýzu audia ovlivnit. Tento efekt se anglicky nazývá *spectral leakage*. Efekt lze potlačit ustředěním okrajů. Za tímto účelem existují okénkové funkce. Funkce se natáhne přes velikost okna a každá hodnota se danou funkcí vynásobí. Existují různé okénkové funkce a lze použít i vlastní. Každá funkce má svoji vlastní charakteristiku. Mezi známé okénkové funkce patří například Hammingovo okno 3.2.



Obrázek 3.2: Koeficienty Hammingova okna (nataženého přes 1000 vzorků)

Svůj název má i prostý výběr prvků vstupního signálu, nazývá se *obdélníková funkce* a na vzorky se při ní žádné váhovací koeficienty neaplikují.

Každá okénková funkce se váže s její frekvenční odezvou. Frekvenční odezva okna se získá aplikací Fourierovy transformace na dané okno v časové oblasti. Tato transformace ukazuje, jak okno ovlivňuje různé frekvenční složky signálu. Na obrázku 3.3 je zobrazena frekvenční odezva Hammingova okna.



Obrázek 3.3: Frekvenční odezva Hammingova okna

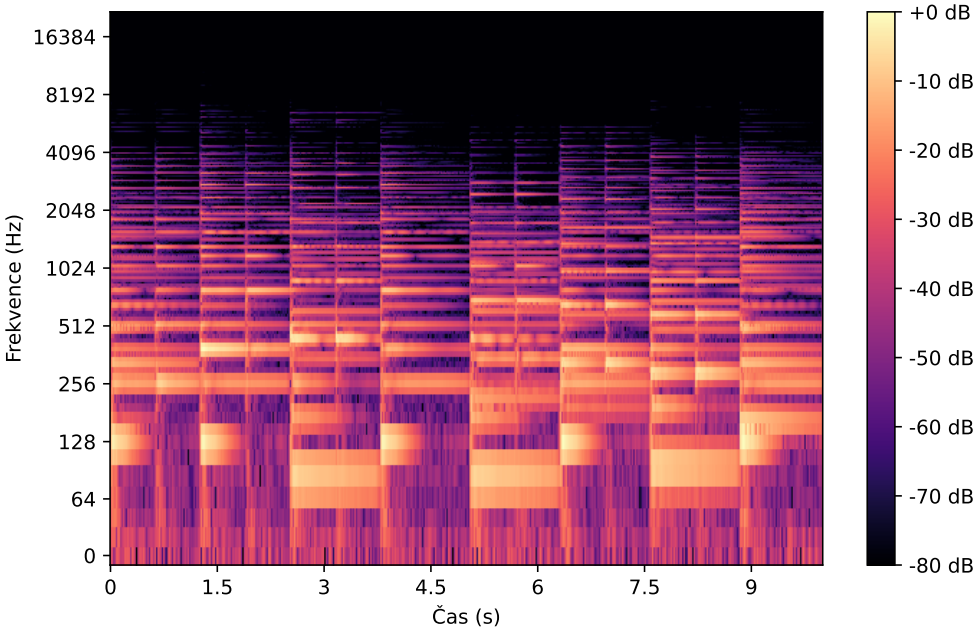
Na obrázku 3.3 jsou vidět tzv. *laloky*. Lalok nacházející se u nulové frekvence je hlavní, jeho spektrální výkon je nejvyšší. Pro Hammingovo okno je hlavní lalok širší než u některých jiných oken, což znamená nižší frekvenční rozlišení. Boční laloky představují nechtěné frekvenční složky, které se objevují kvůli použití okna. Intenzita frekvencí bočních laloků je ale velmi nízká.

3.2 Detekce tónů pomocí Fourierovy transformace

3.2.1 Použití základních funkcí

Knihovna librosa [2] poskytuje spoustu užitečných funkcí pro analýzu audia. Jednou z funkcí je *Short-time Fourier transform (STFT)*, která počítá diskrétní Fourierovu transformaci v krátkých překrývajících se oknech. Výstupem funkce je tedy řada transformací v čase, každá z malé části audia. Základní analýzu audia lze provést následujícím způsobem:

```
y, sr = librosa.load(audio, sr=None, duration=10)
stft = librosa.stft(y)
stft_db = librosa.amplitude_to_db(np.abs(stft), ref=np.max)
librosa.display.specshow(stft_db, sr=sr, x_axis='time', y_axis='log')
```



Obrázek 3.4: STFT spektrogram skladby Twinkle Twinkle Little Star

Svislá osa je v grafu 3.4 zobrazena logaritmicky, protože frekvence not mají také logaritmický rozestup. Z toho vyplývá, že nízké noty mají mezi sebou mnohem menší frekvenční rozestup a tím pádem je i složitější jejich rozlišení. Funkce na analýzu audia, které vycházejí z Fourierovy transformace rozdělují výpočet do oken, které mají danou velikost pro celou analýzu. Když chceme zvýšit frekvenční rozlišení, zmenšíme tím časové rozlišení. Když chceme naopak větší časové rozlišení, přijdeme o frekvenční rozlišení. Jedná se tedy o kompromis mezi frekvenčním a časovým rozlišením. Tento problém řeší metoda *konstantní Q-transformace*, která v průběhu výpočtu mění rozměry oken, pro malé frekvence je okno široké a rozlišuje frekvence přesněji s menší časovou přesností a u vyšších frekvencí je okno užší s lepším časovým rozlišením, viz kapitola 3.3.

3.2.2 Rozlišení a frekvenční koše

Frekvenční rozlišení je závislé na dvou parametrech. Vzorkovací frekvencí a délkou okna. Vzorkovací frekvence jednoduše zvyšuje frekvenční rozlišení. U délky okna je potřeba zvážit kompromis mezi frekvenčním rozlišením a rozlišením v čase.

Na základě frekvenčního rozlišení se frekvence rozdělují do *frekvenčních košů*. Anglické označení je *frequency bins*. Celý frekvenční rozsah se rozdělí do frekvenčních košů, kde každý koš reprezentuje jednu frekvenci. Frekvenční koše jsou rozptýleny se skokem frekvenčního rozlišení. Frekvenční rozsah, který reprezentují, je tedy konstantní. V histogramu každý frekvenční koš reprezentuje jednu hodnotu ve svislé ose.

Výpočet frekvenčního rozlišení Δf:

$$\Delta f = \frac{f_s}{N} \tag{3.5}$$

kde:

- Δf: frekvenční rozlišení,
- f_s: vzorkovací frekvence,
- N: velikost okna, neboli počet vzorků v okně.

Například při vzorkovací frekvenci (f_s) 44.1 kHz a velikosti okna (N) 1024 vzorků se frekvenční rozlišení (Δf) vypočítá jako:

$$\Delta f = \frac{f_s}{N} = \frac{44100 \text{ Hz}}{1024} \approx 43.07 \text{ Hz}$$

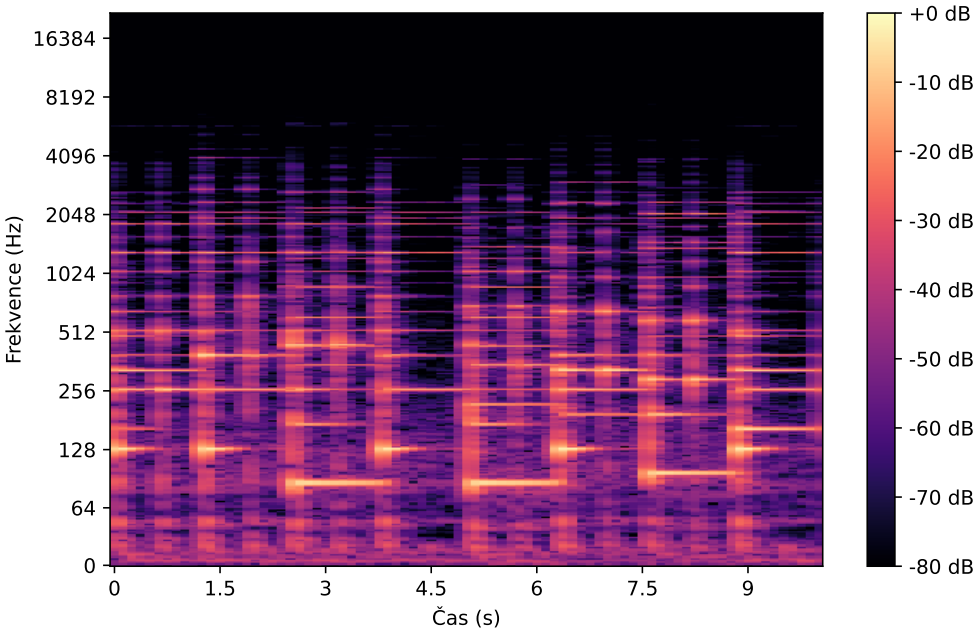
První koš $X[0]$ bude odpovídat frekvenci 0 Hz, druhý koš $X[1]$ 43.07 Hz…

V knihovně librosa nelze nastavit přímo frekvenční rozlišení, ale velikost okna, která nepřímo upraví frekvenční rozlišení. Výchozí hodnota velikosti okna je 2048 vzorků. U audia se nejčastěji používají vzorkovací frekvence 44,1 kHz a 48 kHz. V následujících výpočtech budu vycházet z vzorkovací frekvence 44,1 kHz. Z těchto dvou hodnot dle vzorečku vychází frekvenční rozlišení okolo 22 Hz. Mezi dvěma nejmenšími hodnotami, A_0 a $A_0^\sharp$, je rozdíl zhruba 1.64 Hz. Tyto noty by nebylo možné s tímto rozlišením rozpoznat. Za předpokladu, že by pro rozlišení not stačilo frekvenční rozlišení 2 Hz. Velikost okna je potřeba nastavit na 22,050 vzorků. Toto nastavení zmenší rozlišení v čase na 0,5 s.

Použití tohoto nastavení lze ve funkci librosa.stft docílit změnou parametru *n_fft*, který nastavuje počet vzorků v okně. Zvětšení okna se váže na velikost kroku. Kdyby se velikost kroku nezmenšila, výsledkem by byl téměř stejný počet oken v časové ose s velkým překrytím. Velikost kroku lze změnit parametrem *hop_length*. Některé knihovny mění krok automaticky v závislosti na velikost okna.

Výsledek po nastavení parametrů je vidět na obrázku 3.5. Parametry `n_fft` a `hop_length` jsem nastavil následovně:

```
n_fft = 22050
hop_len = n_fft//4
stft = librosa.stft(y, n_fft=n_fft, hop_length=hop_len)
```



Obrázek 3.5: Nastavení většího frekvenčního rozlišení

3.2.3 Převod Frekvencí na Tóny

Jednotlivé frekvenční koše u Fourierovy transformace neodpovídají přímo tónům. Je potřeba tyto koše převést na reprezentaci tónů. Jedna z možných metod je vzít frekvence jednotlivých tónů, k nim vymyslet funkci a frekvenční koše touto funkcí přepočítat do reprezentace v tónech.

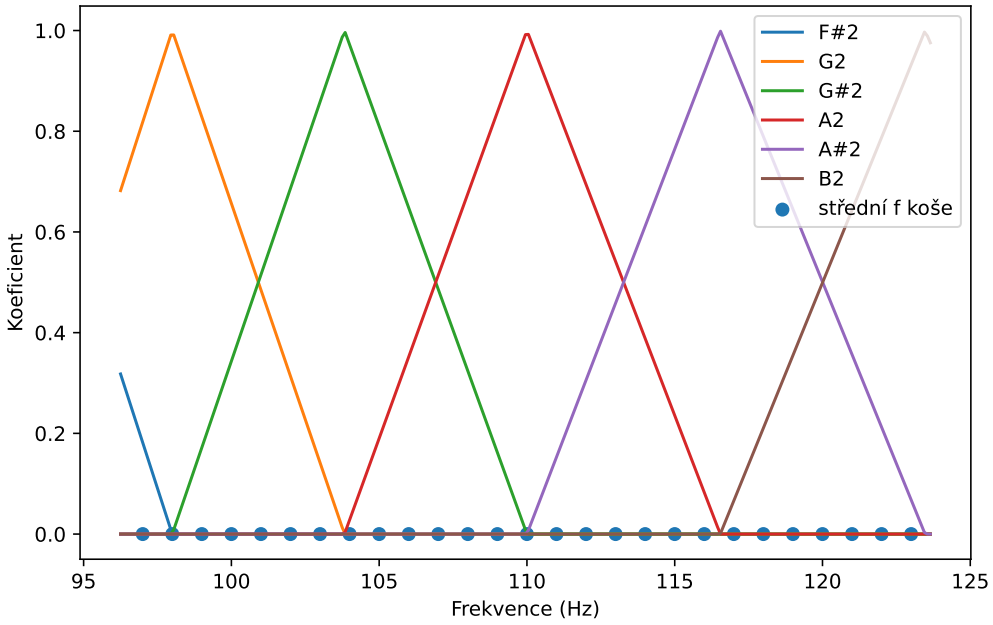
Zvolil jsem funkci ve tvaru trojúhelníku. Své maximum má v právě počítané frekvenci tónu a lineárně klesá k frekvencím vedlejších tónů. Kdyby byla funkce strmější, prošlo by jí méně košů a výsledek by mohl být zavádějící. Pomalejší průběh funkce by naopak příliš vyhladil a sousední tóny by splývaly.

Vzoreček pro výpočet trojúhelníkové funkce:

$$f(f_i, x) = \begin{cases} \frac{x - f_i \cdot 2^{-1/12}}{f_i - f_i \cdot 2^{-1/12}} & \text{pro } f_i \cdot 2^{-1/12} \leq x \leq f_i, \\ \frac{f_i \cdot 2^{1/12} - x}{f_i \cdot 2^{1/12} - f_i} & \text{pro } f_i < x < f_i \cdot 2^{1/12}, \\ 0 & \text{jinak .} \end{cases} \tag{3.6}$$

- f_i: frekvence počítaného tónu.

Z logaritmického kroku mezi tóny vyplývá, že levá strana trojúhelníku je oproti pravé strmější. Průběh funkce pro pár vybraných tónů je vidět na obrázku 3.6.



Obrázek 3.6: Průběh trojúhelníkové funkce

Na vodorovné ose jsou tečkami zobrazeny místa frekvenčních košů. Lze vidět, že maximum funkce není zarovnáno s pozicemi košů. S rostoucí frekvencí se zvětšuje i velikost trojúhelníků. Pro nízké tóny jsou trojúhelníky velice malé, mohou zahrnovat i jen jeden frekvenční koš. Při malém počtu košů hraje roli i zarovnání košů. když bude koš daleko od maxima trojúhelníkové funkce, vedlejší tóny bude těžké rozlišit.

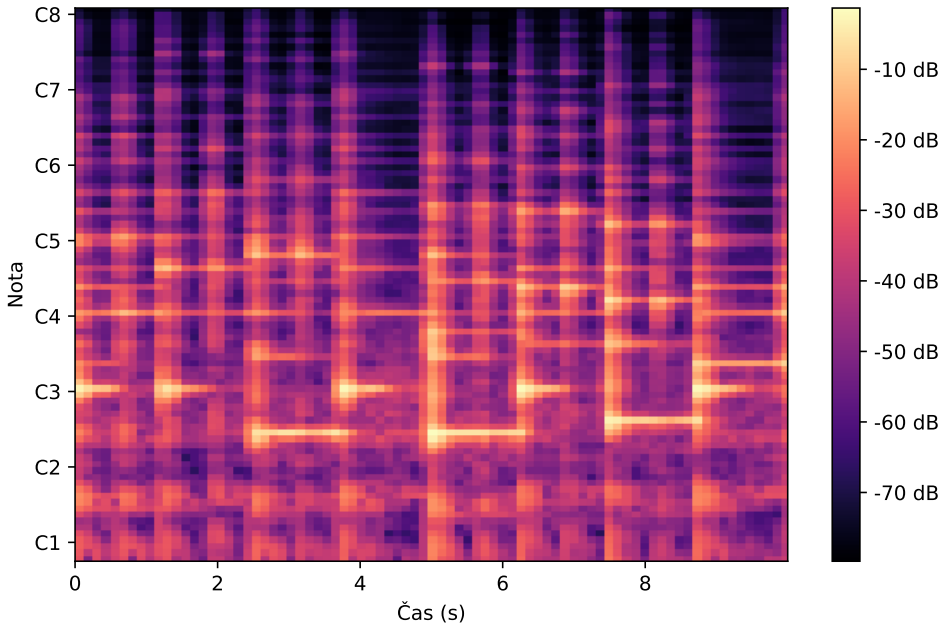
S rostoucí velikostí trojúhelníků u vyšších frekvencí funkce zachycuje více košů. Aby vyšší frekvence nedominovaly, je potřeba normalizace. Převedení frekvencí na tón tedy vypadá tak, že se pro každý koš vezme koeficient vypočítaný trojúhelníkovou funkcí, koeficient se vynásobí na decibely převedenou amplitudou. Pro každý tón se vyhodnotí suma těchto násobků. Výsledná hodnota se normalizuje vydělením součtem koeficientů. Součet váhových koeficientů je ve výsledku 1 viz vzorec:

$$\text{amplituda\_tónu} = \frac{\sum_i w(f_i) \cdot A(f_i)}{\sum_i w(f_i)} \tag{3.7}$$

kde:

- $A(f_i)$: amplituda na frekvenci f_i,
- $A(f_i)$: vypočítaná váha pro frekvenci f_i.

Po provedení STFT ukázkového audia, převedení frekvencí na tónů trojúhelníkovou metodou a normalizaci získáme hodnoty, reprezentující jednotlivé tóny. Jejich hodnoty v čase jsou vidět na obrázku 3.7.



Obrázek 3.7: STFT převedeno na hodnoty tónů

Ze spektrogramu je již viditelná přítomnost některých not. Velkým problémem je, že nízké tóny se přelévají do sousedních tónů a tím dělají rozeznání tónů od vedlejších složitější a zároveň přebíjí tóny vyšší. Přelití do vedlejších tónů nezpůsobil převod trojúhelníkovou metodou, ale fakt, že nižší tóny mají mezi sebou menší frekvenční krok. Proto Fourierova transformace vyhodnotí vedlejší frekvence jako přítomné, protože jsou si velmi podobné.

Jev by mohlo potlačit například použití preemfáze. Ztlumila by nižší frekvence a tím kompenzovala velkou koncentraci nalezených tónů v nízkých frekvencích. Jejich rozptyl by zůstal stejný, ale amplitudy by se zmenšily a nepřekrývaly by tolik vyšší frekvence.

3.2.4 Preemfáze

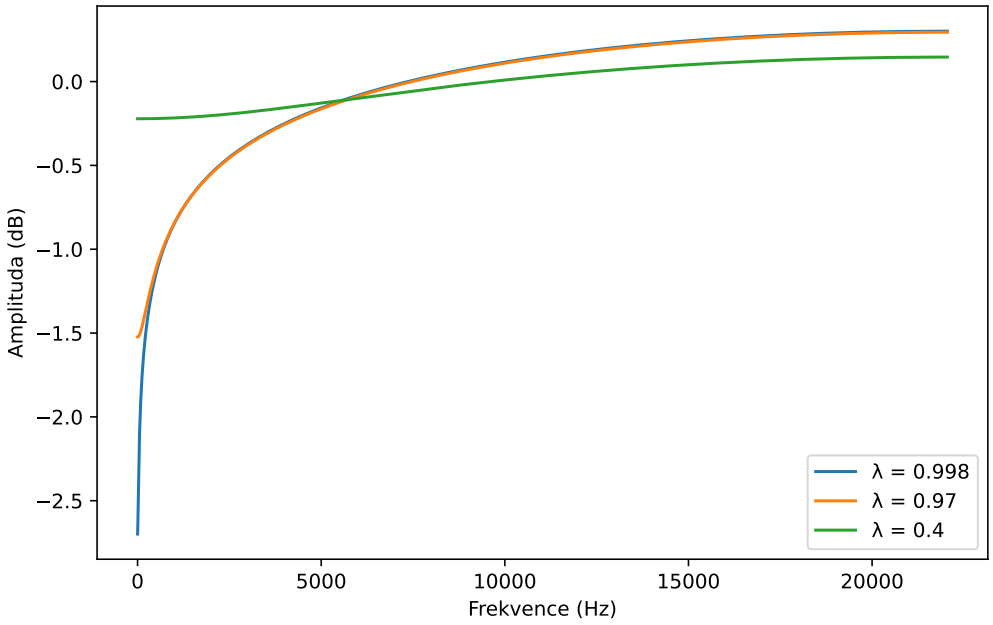
Preemfáze je filtr, který tlumí nízké frekvence a zvýrazňuje tak frekvence vyšší. Aplikuje se na vstupní signál ještě před analýzou audia. Výstup filtru (přenosovou charakteristiku), lze vypočítat vzorcem (převzán z [8]):

$$y_n = x_n - \alpha y_{n-1} \tag{3.8}$$

kde:

- x_n: vstupní signál,
- α: koeficient freemfáze, obvykle se hodnota pohybuje mezi 0,9 a 1,0,
- y_{n-1}: výstupní signál po aplikaci preemfáze na předchozí vzorek n - 1.

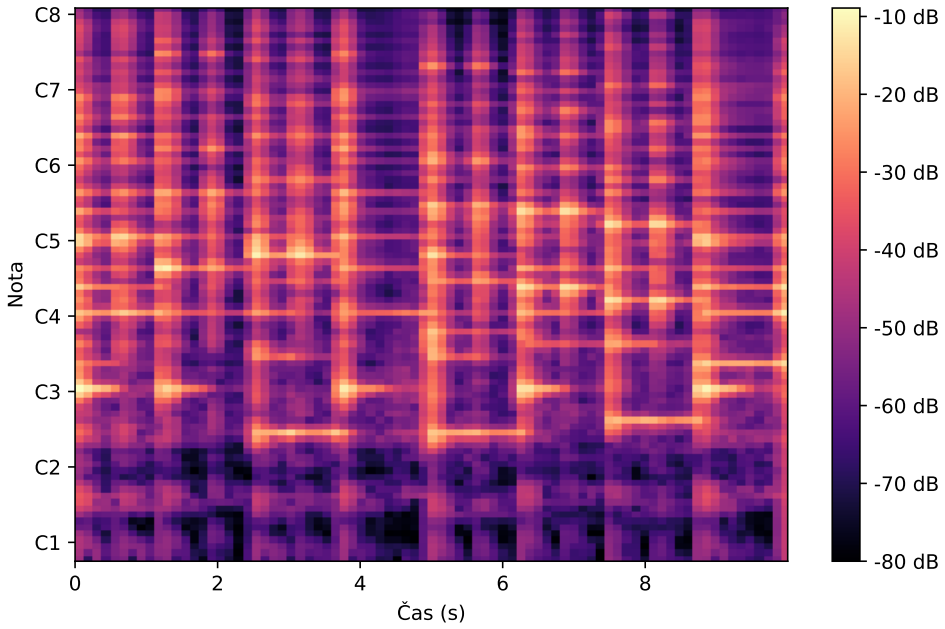
Zobrazení frekvenční charakteristiky pro různé koeficienty je vidět na obrázku 3.8.



Obrázek 3.8: Frekvenční charakteristika preemfázového filtru

Knihovna librosa nabízí implementaci preemfázového filtru: *librosa.effects.preemphasis*. Funkce má nastavený výchozí koeficient filtru α je 0.97, lze je upravit parametrem *coef*. Použil jsem koeficient *coef* 0.998 a udělal jsem znovu analýzu audia, její výsledek je vidět na obrázku 3.9.

```
1  y_preem = librosa.effects.preemphasis(y, coef=0.998)
```



Obrázek 3.9: Spektrogram po aplikaci preemfáze

Aplikace preemfáze pomohla sjednotit hodnoty tónů z různých oblastí frekvenčního rozsahu. Také vyfiltrovala šum, který byl přítomný v prvních dvou oktávách, přestože z této oblasti žádné noty hrány nebyly.

V této fázi ještě noty nelze jednoznačně vyhodnotit. Data obsahují spoustu náhodných vzorků s velkou hodnotou, které se jeví jako tóny, ale v audiu nejsou obsaženy. Dalším krokem je rozmazat hodnoty a potlačit tím šum tvořící tyto osamocené hodnoty.

3.2.5 Potlačení šumu

Noty obvykle hrají přes několik časových vzorků a jejich hodnota postupně klesá. Hodnoty prosakují i ve svislé ose. Nejen u nízkých frekvencí, ale i u frekvencí vyšších. Rozostření dat pomocí 1D konvoluce ve vodorovné ose rozmaže nepřítomné tóny a zanechá shluky velkých hodnot, reprezentující skutečné tóny.

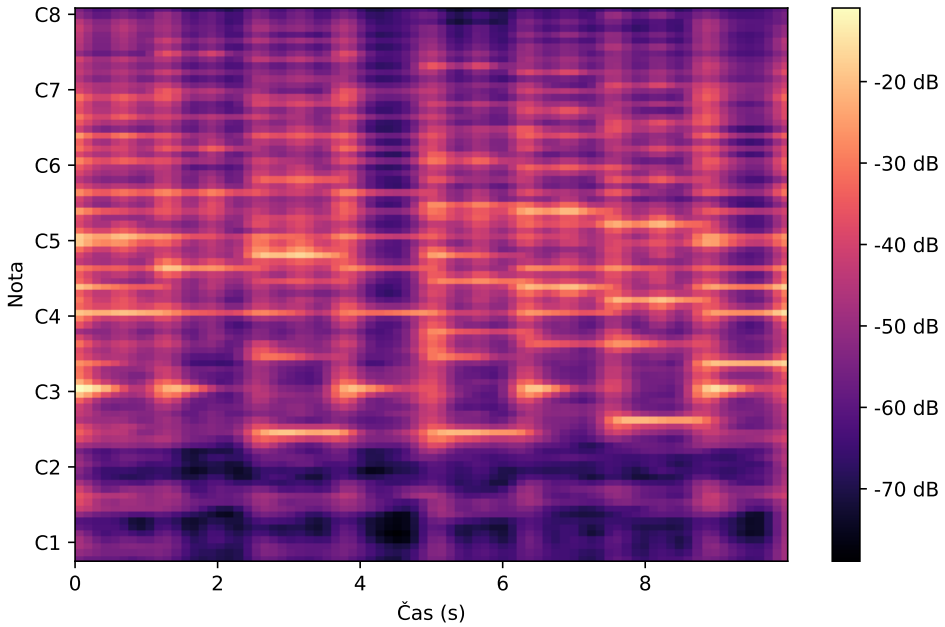
Tvar matice, nebo vektoru, záleží na četnosti dat, neboli parametru *hop_length* a vzorkovací frekvenci. Po chvíli experimentování s různými výškami matic, matice o výšce 1 měla nejlepší výsledky. Pro konvoluci jsem tedy použil vektor, který se aplikuje na vodorovnou osu. Fourierova transformace byla vypočítána s krokem 22050 vzorků a čtvrtinovým posuvem. Audio je na vzorkovací 44,1 kHz. Vzorky jsou tedy vytvořeny s rozmezím 0,125 s. Delší vektor by mohl znamenat přesnější výsledek, ale velmi rychle hrané noty by se mohly začít překrývat.

Zvolil jsem vektor o šířce 5 hodnot, konvoluce tedy bude v rozsahu ±0,25 s:

$$m = \begin{bmatrix} 0.1111 & 0.2222 & 0.3333 & 0.2222 & 0.1111 \end{bmatrix}$$

Součet hodnot vektoru je roven 1. Tím je zachována jednotka vstupních dat. Po aplikaci konvoluce na data dostaneme vyhlazená data, ve kterých jsou potlačené osamocené vysoké hodnoty. Výsledek konvoluce je vidět na obrázku 3.10.

```
data_conv = scipy.signal.convolve2d(data, m, mode="same", boundary="symm")
```



Obrázek 3.10: Spektrogram po aplikaci konvoluce

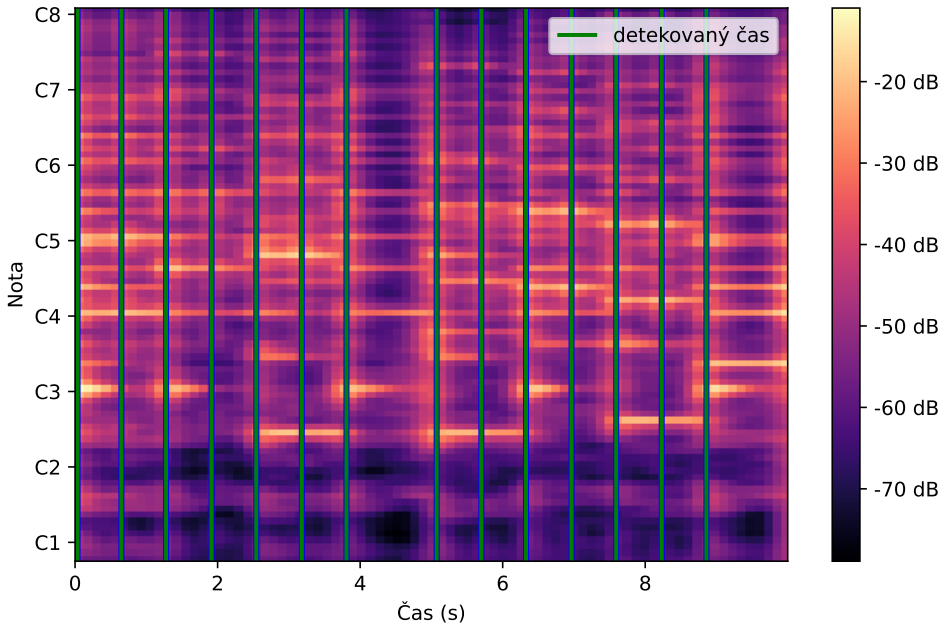
3.2.6 Detekce začátku tónů

Výsledek detekce tónů může být reprezentován několika způsoby. Například pomocí histogramu přítomných not, podobně jako obrázek MIDI souboru 3.1. Lze přidat barevné zobrazení intenzity tónů. U složitějšího audia (už i u ukázkového) je velmi těžké správně určit začátek a především konec noty. Pro hráče na piáno je navíc konec noty nepodstatný, protože většinou drží pedál až po změnu akordu. Při vyhodnocení výstupu jsem se omezil na vyhodnocení času a zahrané noty v tomto čase.

Anglický termín pro detekci začátku tónů je *Onset detection*. Knihovna librosa poskytuje kombinaci funkcí, které dokáží s relativně velkou spolehlivostí detekovat.

Výsledek začátku tónů je vidět na obrázku 3.11. Časy byly detekovány následujícím způsobem:

```
onset_env = librosa.onset.onset_strength(
    y=y_preem, sr=sr, hop_length=512, aggregate=np.median)
detected_times = librosa.onset.onset_detect(
    onset_envelope=onset_env, sr=sr, hop_length=512, units="time")
```



Obrázek 3.11: Detekované začátky tónů

Na obrázku 3.11 jsou detekované časy zvýrazněny svislými čarami. Metoda správně našla všech 14 okamžiků odpovídajících původnímu MIDI souboru.

S připravenými daty a detekovanými časy stačí analyzovat jednotlivé sloupce se vzorky v detekovaných časech.

3.2.7 Analýza v detekovaných časech

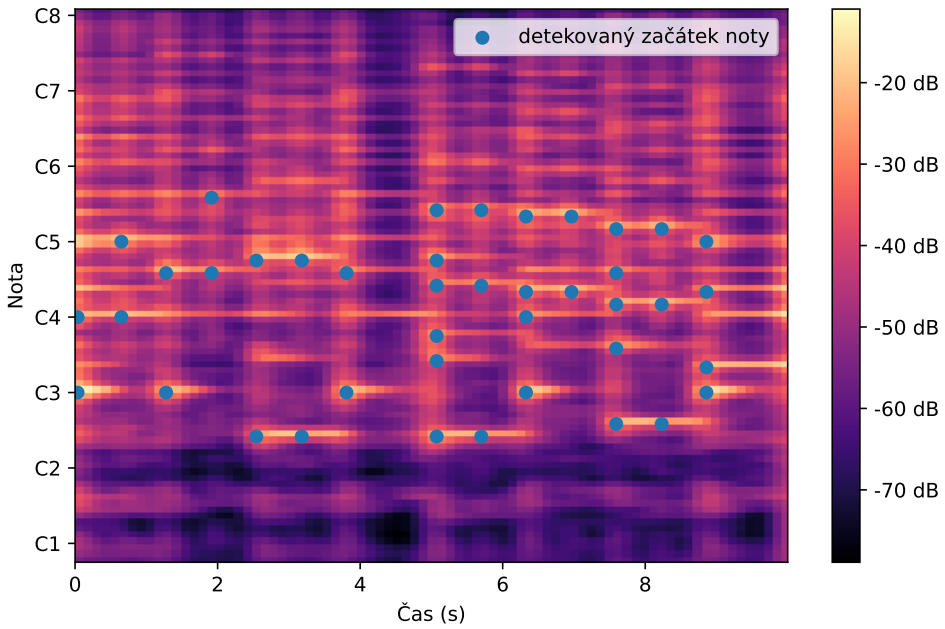
Pro nalezení jedné noty stačí vzít vzorek s nejvyšší hodnotou. Při hledání více not se může pevně definovat hranice, která bude určovat, zda je amplituda dostatečně velká, aby byla notou. Spolehlivější je najít nejvyšší amplitudu, o určitou hodnotu ji zmenšit a takto nastavit hranici dynamicky.

Použití tohoto způsobu většinou identifikuje několik tónů vedle sebe, protože hodnoty vedle sebe prosakují do tónů vedlejších, tento jev posiluje použité vyhlazování (potlačení šumu). Abych detekci vedlejší tónů jako noty odstranil, vytvořil jsem algoritmus, který funguje následovně:

- Vyhodnotí se nejbližší nižší sloupec pro detekovaný čas.
- V tomto sloupci se nalezne maximální hodnota.
- Ze sloupce se vezmou indexy tónů, které mají hodnotu větší, než maximum - 8 db.
- Indexy se seřadí podle amplitudy od největší po nejmenší.
- Přes indexy se iteruje (od toho nejvyšší amplitudou) a z pole se zahodí všechny následující indexy v rozsahu ± 2.

Tímto algoritmem se lokálně detekují jen tóny s nejvyšší amplitudou a tóny v blízkém okolí se přestanou hledat. Podle hudební teorie 2.6 by se noty v takovéto blízkosti neměly vyskytovat, protože spolu nejsou v harmonii.

Konečný výsledek detekce tónů je vidět na obrázku 3.12. Obrázek obsahuje spektrogram, překrytý tečkami, které reprezentují detekované noty.



Obrázek 3.12: Výsledek detekce not

3.2.8 Vyhodnocení výsledků

Vyhodnocení výsledků probíhá metodou F_1 *score*. Protože je audio nahrávka syntetizována přímo z MIDI souboru, můžeme přímo porovnat výsledek detekce s referencí. Podrobnější popis, jak se statistika F_1 score vyhodnocuje, je popsán v kapitole 4.3.

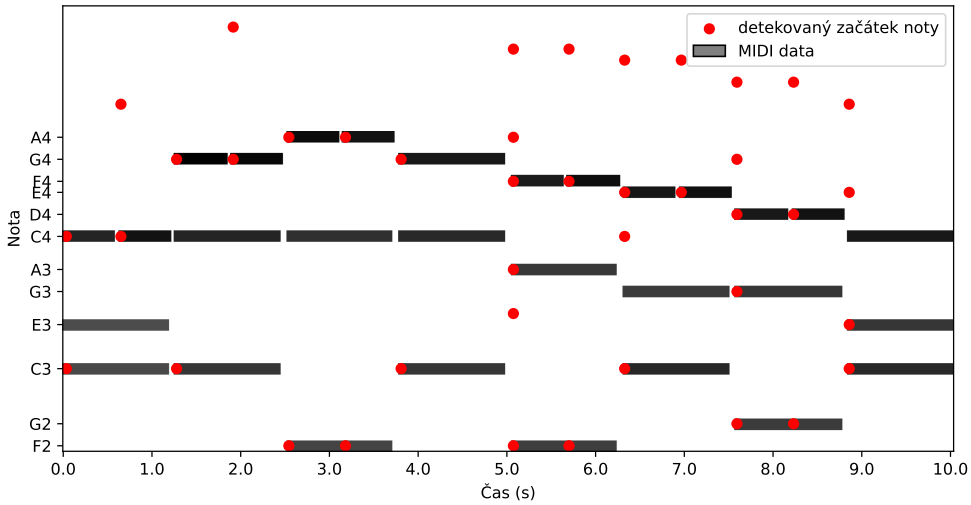
Kategorie	**Počet**
True Positives	24
False Positives	17
False Negatives	6

Tabulka 3.1: Vyhodnocení F_1 score

Ze sečtených kategorií byly vypočítány hodnoty:

- Precision = 0.5854
- Recall = 0.8
- F_1 score = 0.676

Pozice detekovaných not a referenční MIDI soubor je vidět na obrázku 3.13.



Obrázek 3.13: Překrytí výsledku detekce a MIDI dat

Na obrázku lze vidět úspěšnou detekci více než poloviny not. Několik nedetekovaných a spoustu detekovaných navíc. Protože noty detekované navíc se nacházejí vždy oktávu nad nějakým jiným existujícím, jedná se o detekci alikvotních tónů 2.4.

Je možné řešení rozšířit například o potlačení alikvotních tónů, která by po detekci noty potlačila hodnoty všech svých alikotvních tónů (noty by musely detekovat postupně od nejnižší). Ovšem prací s alikvotními tóny se zabývá vlastní řešení 4.1.

3.3 Konstantní Q-transformace

Konstantní Q-transformace, dále jako CQT, je komplexnější technika analýzy signálů. Stejně jako krátkodobá Fourierova transformace převádí signál do spektrogramu, kde vodorovná osa reprezentuje čas a svislá osa frekvenci, nebo přímo tón. Na rozdíl od STFT používá variabilní frekvenční rozlišení, které přizpůsobuje aktuální frekvenci. Díky dynamickému rozlišení se nižší frekvence analyzují s větší frekvenční přesností, ale s menším časovým rozlišením. Frekvenční rozlišení se s rostoucí frekvencí snižuje a časové rozlišení se zvyšuje logaritmicky. Frekvence not se také zvedá logaritmicky. Díky tomu lze správným nastavení frekvenční doménu namapovat na noty. Krok se nejčastěji nastavuje tak, aby každý frekvenční koš odpovídal právě jednomu půltónu. Z tohoto důvodu je CQT spojována s analýzou hudby. Knihovny implementující CQT jsou k tomuto použití přizpůsobeny a obsahují parametry ze světa hudby, jako je nastavení nejnižší a nejvyšší noty (z nich vytvoří frekvenční rozsah), počet detekovaných not (určí počet frekvenčních košů) nebo ladící frekvenci (správné zarovnání frekvenčního rozsahu a košů).

„Q“ v názvu metody reprezentuje tzv. *quality factor*. Tento faktor se používá pro definování šířky frekvenčního pásma ze znalosti střední frekvence. Výpočet kvalitativního faktoru Q se nachází na následující straně.

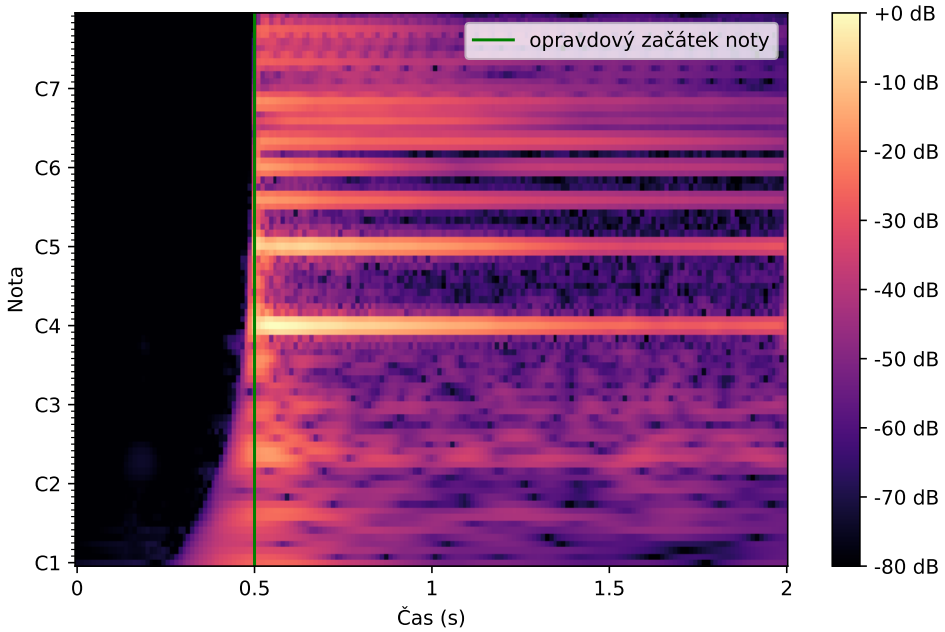
$$Q = \frac{f_c}{\Delta f} \tag{3.9}$$

kde:

- f_c: střední frekvence,
- Δf: frekvenční rozlišení.

Knihovna librosa implementuje CQT tak, že šířka oken se mění, ale *hop_length* se nepřizpůsobuje a zůstává konstantní. Tato implementace zajišťuje, že každý frekvenční koš bude mít v časové ose stejný počet vzorků. Na začátku a konci audia při této implementaci delší okna přesahují za hranice audia. Chybějící hodnoty signálu se vyplní nulami, tzv. *zero-padding*. Toto řešení zajišťuje, že délka výstupního signálu zůstane konzistentní s délkou vstupního signálu.

Při použití CQT se může zdát, že nižší frekvence byly detekovány dříve. Je to dáno větší šířkou okna. Širší okna analýzu příliš neovlivňují, protože ve své implementaci beru v úvahu pouze hodnoty detekované v čase nalezeném *onset* detekcí. V případě rychlého náběhu noty (krátký *attack time*) se může vypočítaná maximální amplituda opozdit. Dopad dynamické velikosti okna je vidět při analýze krátké nahrávky 3.14 (nota zahrána přesně v čase 0,5 s).



Obrázek 3.14: CQT audia obsahující zahranou notu C_4

Stejně jako Fourierova transformace i CQT používá okénkové funkce.

Efektivní implementaci CQT popisuje článek [9]. Detailnější vysvětlení a ukázka je v článku [10].

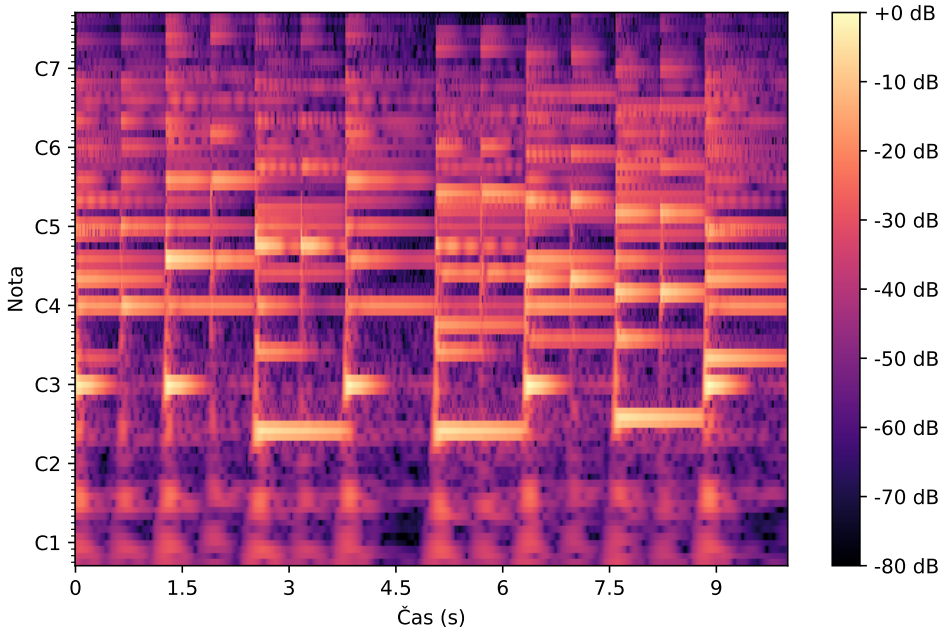
3.4 Detekce tónů pomocí Konstantní Q-transformace

3.4.1 Použití základních funkcí

Knihovna librosa poskytuje implementaci CQT stejnojmennou funkcí *librosa.cqt*. Funkce je přizpůsobena používání k hudební analýze. Ve výchozím stavu je nastavena na vytvoření 88 frekvenčních košů přesně odpovídajícím jednotlivým tónům. Počítá s temperovaným laděním s výchozí hodnotou 440 Hz pro A_4.

Výsledek použití funkce ve výchozím stavu je vidět na obrázku 3.15.

```
C = librosa.amplitude_to_db(np.abs(librosa.cqt(y=y, sr=sr)), ref=np.max)
librosa.display.specshow(C, y_axis='cqt_note', sr=sr, x_axis='time')
```



Obrázek 3.15: Výchozí použití funkce librosa.cqt

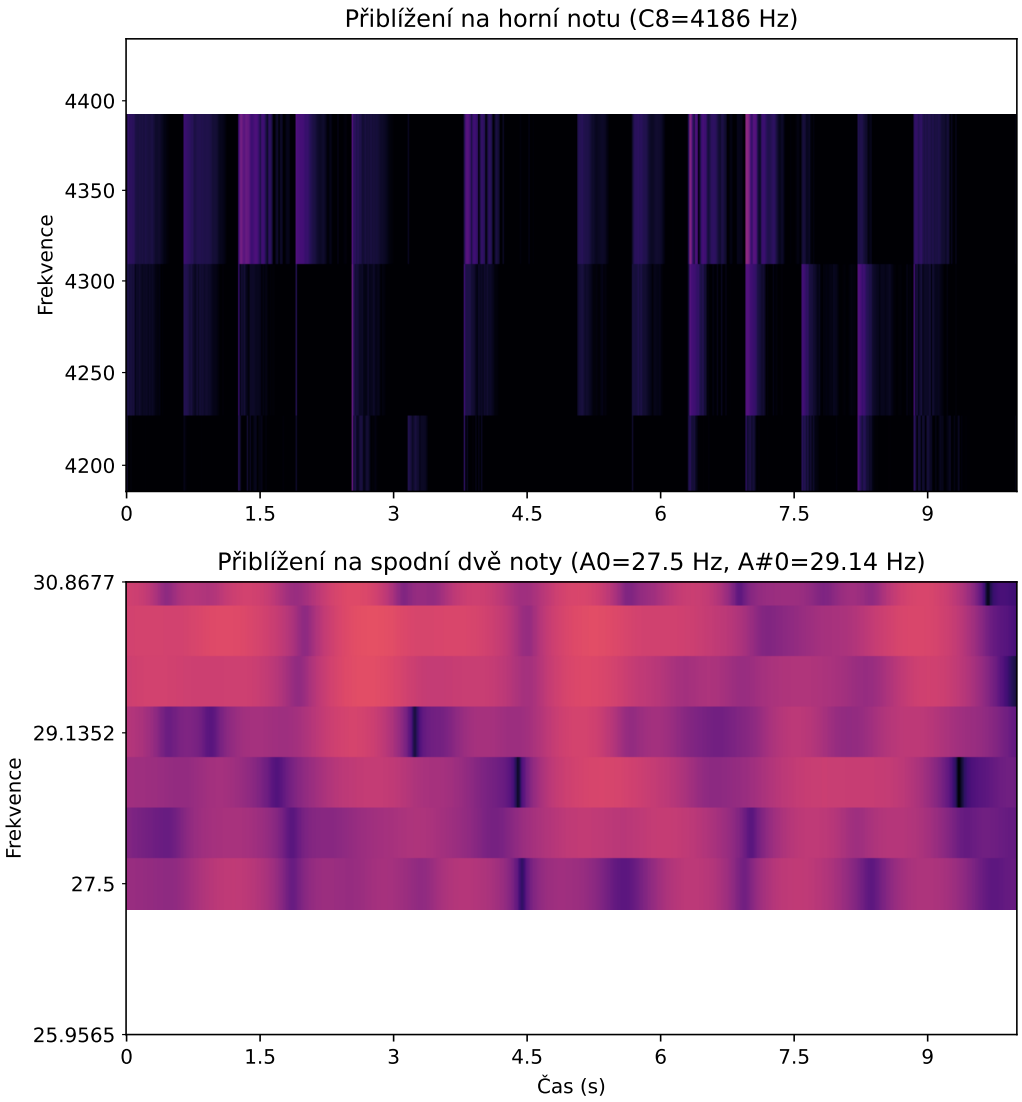
Na obrázku 3.15 je vidět přítomnost tónů, ale hodnoty se značně přelévají do vedlejších tónů.

3.4.2 Zvětšení počtu frekvenčních košů

Frekvenční rozsahy košů se překrývají, proto se mohou mít podobné hodnoty. Lze zvětšit počet košů na půltón. Ideální počet jsou 3 koše. S tímto počtem se u každého půltónu odstraní krajní koše a nechá se jen prostřední. Přítomná nota půjde jednoznačněji identifikovat, nevýhodou je snížení rozlišení v čase.

Zarovnání vzorků

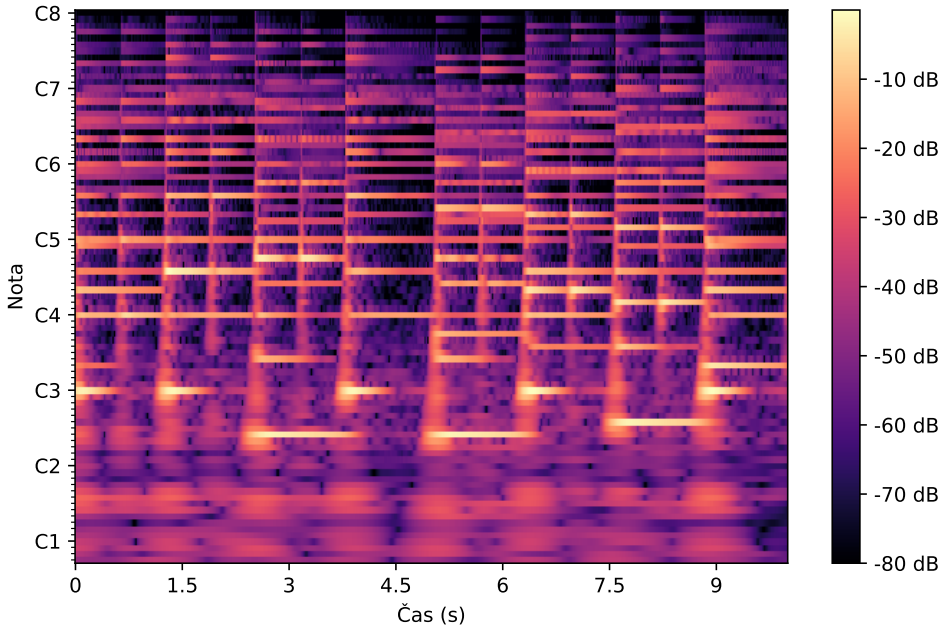
Knihovna librosa rozdělení půltónu do více (tří) vzorků trochu neintuitivně. Základní frekvenci noty nepřidělí prostřednímu ze tří vzorků, ale tomu prvnímu. Vznikne takto jeden chybějící vzorek pro první půltón a jeden vzorek je navíc za nejvyšším půltónem. Tuto skutečnost jsem zjistil vytvořením obrázku 3.16.



Obrázek 3.16: Ukázka zarovnání vzorků v ose y

Na obrázku 3.16 je vidět, že první vzorek má frekvenci A_0 a nad ním jsou dva nové vzorky, namísto toho, aby se z každé strany vytvořil jeden. V mé implementaci chybějící vzorek není využitý, ale je potřeba na něj myslet při indexování. Předpokládané využití indexů `[1::3]` se tedy posune na `[0::3]`. Výsledeka následujícího kódu je vidět na obrázku 3.17.

```
per_note = 3
C = librosa.cqt(y=y, bins_per_octave=12*per_note, n_bins=88*per_note, sr=sr)
C_db = librosa.amplitude_to_db(np.abs(C), ref=np.max)
librosa.display.specshow(C_db[0::3], y_axis='cqt_note', sr=sr, x_axis='time')
```

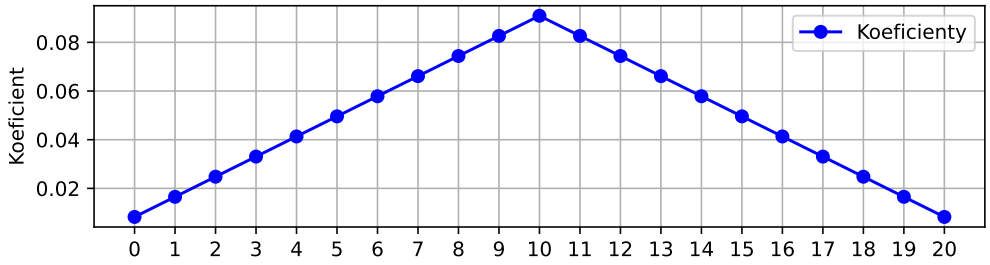


Obrázek 3.17: CQT po zvětšení počtu frekvenčních košů

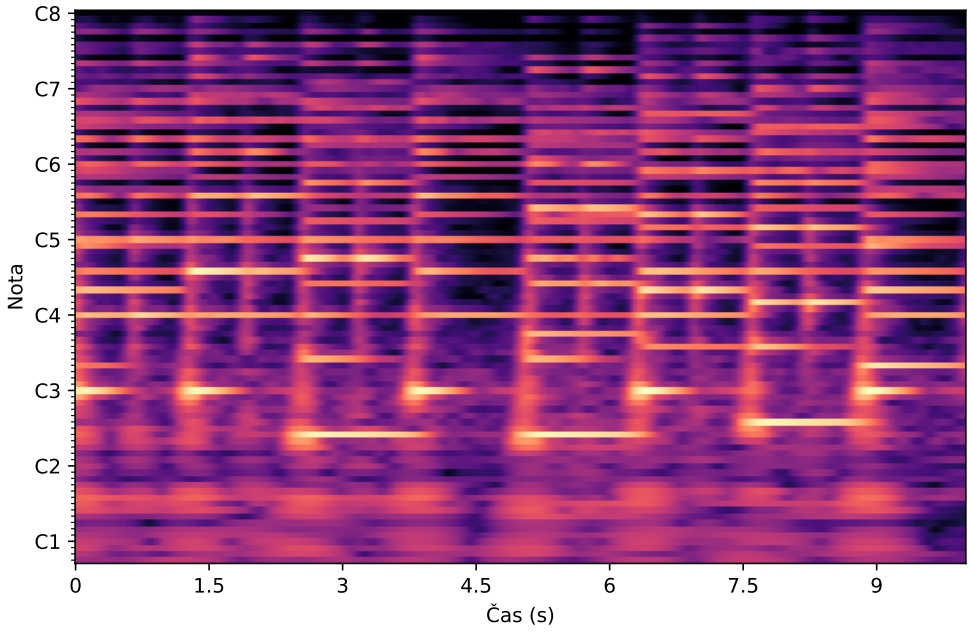
3.4.3 Potlačení šumu v časové ose

Jednotlivé tóny jsou vidět poměrně jednoznačně. Pouze v čase začátku tónu jsou vysoké hodnoty přes většinu spektra. Aby se předešlo detekci některé z těchto vysokých hodnot jako tónu, lze hodnoty rozmazat ve vodorovné ose.

Vzorkovací frekvence audia je 44,1 kHz a *hop_length* 512 vzorků. Vzorky mají ve vodorovné ose kvůli malému kroku husté obsazení a je vhodné použít delší vektor. Vytvořil jsem vektor o velikosti 21 vzorků, bude provádět konvoluci v rozsahu ±0,115 s. Koeficienty od středu ke kraji lineárně klesají a jejich součet je roven 1, viz obrázek 3.18. Aplikace konvoluce je vidět na obrázku 3.19.



Obrázek 3.18: Hodnoty vektoru (délka 21 vzorků)



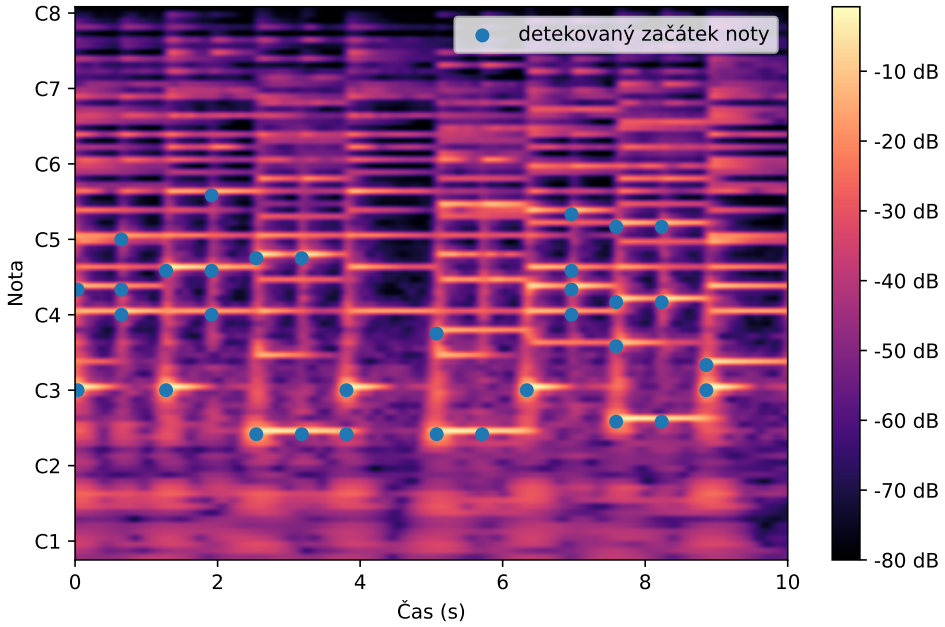
Obrázek 3.19: Spektrogram po rozostření

3.4.4 Detekce tónů

Nad audiem se provede detekce začátku tónů. Tato detekce se provádí přímo nad audiem, proto způsob zpracování na výsledek nemá vliv a výsledek je stejný jako u analýzy Fourierovou transformací 3.2.6. Výsledkem onset detekce jsou indexy vzorků, ve kterých byl začátek tónu detekován. Tento index vzorku se převede na nejbližší sloupec ze spektrogramu a s tímto sloupcem se dále pracuje. Protože jsou hodnoty v časové ose rozmazány, do nich se už částečně promítly hodnoty z okolních sloupců.

K vyhodnocení přítomných not se použije algoritmus, který byl dříve použitý u Fourierovy transformace 3.2.7. Pouze se změní prahová hodnota. Ta opět vychází z maximální hodnoty ve sloupci a propustí noty s hodnotou větší, než $maximum - 10$ dB.

Konečný výsledek detekce tónů je vidět na obrázku 3.20.



Obrázek 3.20: Výsledek detekce not

3.4.5 Vyhodnocení výsledků

Vyhodnocení výsledků metodou F_1 *score* je následovné:

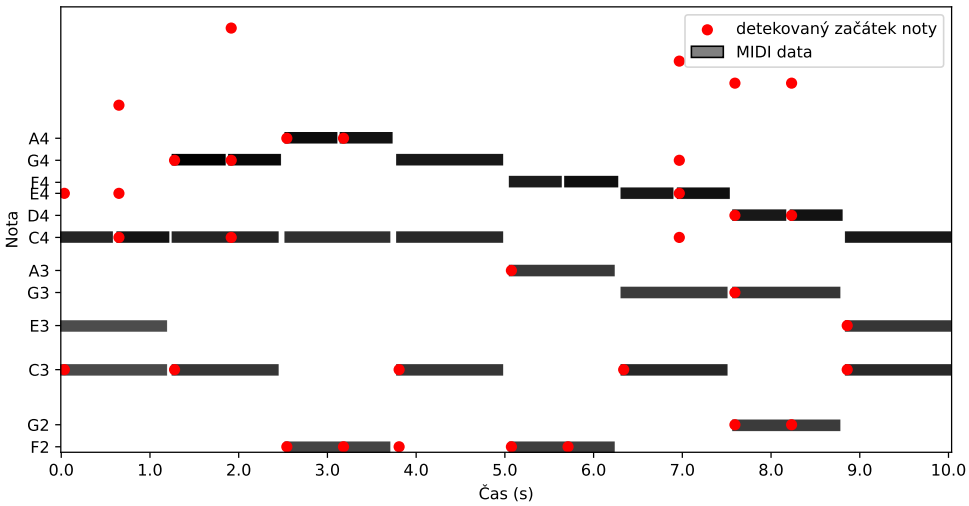
Kategorie	Počet
True Positives	19
False Positives	14
False Negatives	11

Tabulka 3.2: Vyhodnocení F_1 score

Ze sečtených kategorií byly vypočítány hodnoty:

- Precision = 0.5757
- Recall = 0.6333
- F_1 score = 0.6032

Pozice detekovaných not a referenční MIDI soubor je vidět na obrázku 3.21.



Obrázek 3.21: Překrytí výsledku detekce a MIDI dat

Z obrázku je opět vidět nechtěná detekce alikvotních tónů. Dále několik nedetekovaných not a zhruba pět detekcí navíc, způsobené doznínáním noty, která byla zahrána již v předešlém čase.

Kapitola 4

Implementace

Pro detekci tónů jsem vytvořil implementaci pomocí Fourierovy transformace a konstantní Q-transformace, které jsou popsány v kapitolách 3.2 a 3.3. Kromě těchto základních metod jsem vytvořil vlastní implementaci vycházející z přítomnosti alikvotních tónů. Dále jsem vytvořil jednoduchou aplikaci pro detailnější zobrazení výsledků.

Veškerá implementace byla provedena v jazyce python. Soubory určené ke spuštění jsou *app.py* a *Fscore.py*. Implementaci jsem vytvořil v python verzi 3.11 a použil jsem následující knihovny:

- librosa==0.10.1
- matplotlib==3.7.1
- numpy==1.24.2
- pydub==0.25.1
- scipy==1.14.0
- midi2audio==0.1.1
- pretty_midi==0.2.10
- PyQt6==6.7.1
- PyQt6_sip==13.6.0

Pro úplné spuštění je potřeba instalace těchto knihoven s kompatibilní verzí. MIDI soubory jsou ve složce *resources* a jsou syntetizované ve složce *generated*. Při potřebě vlastní syntetizace je potřeba ve zdrojových souborech nastavit cestu ke zvukové bance (soubor s příponou .sf2). Tento soubor kvůli své velké velikosti není přiložen, soubor lze získat například na odkazu[1].

[1] https://sites.google.com/site/soundfonts4u/

4.1 Metoda alikvotních tónů

Všechny strunné, dechové a řada dalších hudebních nástrojů produkují alikvotní tóny. Toho se dá využít i pro detekci tónů základních. A to tak, že se bude hledat přítomnost nejen základní frekvence, ale i pár zvolených alikvotních tónů. Vybrat násobky alikvotních tónů lze různým způsobem. Například liché násobky (3f, 5f...) by lépe detekovali dvě zahrané noty z oktávy, jinak vybrané násobky například kvintu, kvartu, tercii... Já jsem zvažoval první dva alikvotní tóny (2f, 3f). První alikvotní tóny obvykle mají největší amplitudu, a vzdálenější alikvotní tóny by při detekci vyšších tónů mohly porušovat Nyquistův-Shannonův vzorkovací teorém.

Vycházel jsem z principů Fourierovy transformace a CQT. Pro každou detekci základního tónu jsem vytvořil několik komplexních exponenciál. Tyto exponenciály odpovídají základnímu tónu a několika alikvotních tónů. Vypočítaná komplexní čísla jsem převedl na magnitudy, vynásobil mezi sebou a výsledný součin jsem vydělil hodnotou vycházející z velikosti okna a počtu použitých alikvotních tónu. Násobením jsem dosáhl významného zvýraznění existujících not, vydělením se výsledek normalizoval. Výpočet jsem prováděl pro frekvence odpovídajících 88 tónům klaviatury.

Rovnice pro výpočet magnitudy pro frekvenci f:

$$M_f = \left(\prod_{o \in \mathrm{O}} \left(\frac{1}{N} \left| \sum_{n=0}^{N-1} x_n \cdot e^{-j \cdot 2\pi o \frac{f \cdot n}{sr}} \right| \right) \right)^{\frac{1}{|O|}} \tag{4.1}$$

kde:

- f: Právě počítaná frekvence,
- M_f: výsledná magnituda pro frekvenci f,
- n: index vzorku vstupního signálu,
- N: celkový počet vzorků vstupního signálu,
- $x[n]$: n-tý vzorek vstupního signálu,
- j: imaginární jednotka, kde $j^2 = -1$,
- o: hodnota alikvotního tónu z množiny,
- O: množina, reprezentující násobky základní frekvence, které mají být zvažovány, neboli alikvotní tóny. Například {1, 2, 3, 5} –> f, 2f, 3f, 5f,
- |O|: velikost množiny O, určuje počet vytvořených exponenciál,
- sr: vzorkovací frekvence.

4.1.1 Rozbor rovnice

Výpočet koeficientu DFT

$$\sum_{n=0}^{N-1} x_n \cdot e^{-j \cdot 2\pi o \frac{f \cdot n \cdot o}{sr}} \tag{4.2}$$

Tato část rovnice je podobná výpočtu DFT. Exponenty $-j \cdot 2\pi \frac{k \cdot n \cdot o}{N}$ a $-j \cdot 2\pi \frac{f \cdot n \cdot o}{sr}$ jsou ekvivalentní. U standardního výpočtu DFT se používá verze s parametrem k, který reprezentuje index frekvenčního koše. Protože chceme počítat DFT pro konkrétní frekvenci, je lepší použít verzi exponentu vycházející z frekvence f a vzorkovací frekvence sr.

Ve výpočtu přibyl parametr o. Reprezentuje násobek základní frekvence. Například hodnoty $O = \{1, 2, 3, 5\}$ znamenají výpočet magnitudy pro základní frekvenci a dále pro první, druhý a čtvrtý alikvotní tón.

Součin přes alikvotní tóny

$$\prod_{o \in \mathrm{O}} \left| \sum_{n=0}^{N-1} x_n \cdot e^{-j \cdot 2\pi o \frac{f \cdot n \cdot k}{sr}} \right| \tag{4.3}$$

Výsledky z rovnice 4.2 jsou převedeny z komplexních čísel na magnitudy. Přicházíme o informace, jako je fáze, ale dále je pro nás podstatná pouze přítomnost frekvence. Součin prochází přes magnitudy základní frekvence a specifikovaných alikvotních tónů.

Normalizace

$$\left(\prod_{o \in \mathrm{O}} \left(\frac{1}{N} \left| \sum_{n=0}^{N-1} x_n \cdot e^{-j \cdot 2\pi o \frac{f \cdot n}{sr}} \right| \right) \right)^{\frac{1}{|O|}} \tag{4.4}$$

Vypočítal jsem koeficient DFT z oken s různým počtem vzorků a vynásobil několik magnitud mezi sebou. Vznikl dynamický rozsah hodnot a je potřebné tyto hodnoty normalizovat, aby se detekované hodnoty pohybovaly v přibližně podobném rozsahu:

- Každá magnituda se dělí počtem vzorků v okně N,
- Výsledek se odmocní velikostí množiny $|O|$.

Suma se tedy normalizuje dělením a součin odmocněním.

4.1.2 Implementace v jazyce Python

Funkce počítající magnitudy pomocí alikvotních tónů se nachází v přiloženém souboru *core.py* pod názvem *custom_dft*.

Výstupem funkce je vektor o velikost 88 prvků. Každá hodnota reprezentuje magnitudu jedné z not. Vstupní parametry jsou následující:

- y: Navzorkovaný vstupní signál,
- sr: vzorkovací frekvence,
- *undertone*: *bool* hodnota - rozšíření metody, popsané v kapitole 4.1.3,
- *overtones*: vektor násobků základní frekvence (1 by měla být vždy přítomná).

```python
def custom_dft(y, sr: int, undertone=True, overtones=[1, 2, 3, 5]) -> List[float]:

  # Inicializace promennych
  N = len(y)
  magnitudes = np.zeros(len(notes["freqs"]))

  # Aplikace Hammingova okna
  hamming_window = librosa.filters.get_window("hamming", N)
  y = hamming_window * y

  # Cyklus pro kazkou notu (88 not)
  for i, freq in enumerate(notes["freqs"]):

    n = np.arange(N)
    X = np.zeros(len(overtones), dtype=np.complex128)

    # Pro kazdy overtone se vypocita exponenciala
    for j, overtone in enumerate(overtones):
      exponent = -2j * np.pi * overtone * freq * n / sr
      X[j] = np.sum(y * np.exp(exponent)) / N

    # prevedeni komplexniho cisla na magnitudu
    magnitudes_overtone = np.abs(X)

    # soucin magnitud
    magnitude = np.prod(magnitudes_overtone)

    # normalizace
    magnitudes[i] = magnitude ** (1 / len(overtones))

  return magnitudes
```

4.1.3 Zohlednění o oktávu nižšího tónu

Detekce pomocí pouze alikvotních tónů může lehce nalézt i první alikvotní tón (ten o oktávu výše) jako notu. Například zahrání noty A_3 způsobí přítomnost frekvencí 220, 440, 660, 880, ... Hz. Základní forma metody používající alikvotní tóny by pro notu A_4 vyhodnotila vysokou magnitudu, protože se skládá z podobných frekvencí: 440, 880, 1320... Hz. Pro omezení tohoto jevu jsem přidal možnost zohlednit přítomnost tónu o oktávu níže. Vytvoří se ještě jedna další magnituda exponenciály a tentokrát se jí dělí:

$$M_f = \left(\frac{\left| \prod_{o \in \mathrm{O}} \left(\frac{1}{N} \left| \sum_{n=0}^{N-1} x[n] \cdot e^{-j \cdot 2\pi \frac{f \cdot n \cdot o}{sr}} \right| \right) \right|}{\frac{1}{N} \left| \sum_{n=0}^{N-1} x[n] \cdot e^{-j \cdot 2\pi \frac{f \cdot n}{sr}} \right|} \right)^{\frac{1}{|O|-1}} \tag{4.5}$$

Princip vzorce zůstává podobný originálu 4.1. Změnou je dělení magnitudou vyhodnocenou z exponenciály se stoupáním o polovině základní frekvence. Této změně byla přizpůsobena i normalizace.

Z předešlého kódu se nahradí 29. řádek následujícím rozšířením:

```
1   if undertone:
2     exponent = -j * np.pi * freq * n / sr
3     X_u = np.sum(y * np.exp(exponent)) / N
4
5     undertone_magnitude = np.abs(X_u)
6     magnitude /= undertone_magnitude
7
8     magnitudes[i] = magnitude / (N ** (1 / (len(overtones)-1)))
9   else:
10    magnitudes[i] = magnitude / (N ** (1 / len(overtones)))
```

4.1.4 Nastavení citlivosti detekce

Pomocí metody se vypočítají hodnoty pro každou notu. Tyto hodnoty nazývám skóre, protože magnitudy jsou již upravené normalizací. Dalším krokem je vyhodnotit, které z těchto hodnot detekujeme jako přítomnou notu. Je potřeba stanovit práh, neboli *threshold*. Tento práh je vytvořený z části maximální nalezené hodnoty. Abych našel ideální práh, implementoval jsem nejdříve metriku f_1 score 4.3 a poté zkoušel detekce not s různým nastavením prahu.

Výsledky testování jsou zobrazeny v následující tabulce:

Metoda	Práh	TP	FP	FN	Precision	Recall	Score
Aliq	0.2	509	1008	245	0.3355	0.6751	0.4483
	0.25	482	805	272	0.3745	0.6393	0.4723
	0.3	460	647	294	0.4155	0.6101	0.4944
	0.35	432	529	322	0.4495	0.5729	0.5038
	0.4	412	438	342	0.4847	0.5464	0.5137
	0.45	395	369	359	0.5170	0.5239	**0.5204**
	0.5	369	319	385	0.5363	0.4894	0.5118
	0.55	353	295	401	0.5448	0.4682	0.5036
	0.6	331	269	423	0.5517	0.4390	0.4889
	0.65	303	237	451	0.5611	0.4019	0.4683
$Aliq_u$	0.05	552	1518	202	0.2667	0.7321	0.3909
	0.1	501	935	253	0.3489	0.6645	0.4575
	0.15	462	697	292	0.3986	0.6127	0.4830
	0.2	432	559	322	0.4359	0.5729	0.4951
	0.25	405	451	349	0.4731	0.5371	**0.5031**
	0.3	381	389	373	0.4948	0.5053	0.5000
	0.35	356	332	398	0.5174	0.4721	0.4938
	0.4	339	293	415	0.5364	0.4496	0.4892
	0.45	324	270	430	0.5455	0.4297	0.4807
	0.5	306	250	448	0.5504	0.4058	0.4672
	0.55	290	227	464	0.5609	0.3846	0.4563
	0.6	270	206	484	0.5672	0.3581	0.4390
	0.65	259	190	495	0.5768	0.3435	0.4306

Tabulka 4.1: Nastavení prahu pro metodu alikvotních tónů

Tabulka zobrazuje výsledky metody alikvotních tónů bez (Aliq) a s ($Aliq_u$) použitím parametru *undertone*. Očekával jsem, že koeficienty se budou lišit. Parametr undertone totiž výrazně zvětšuje hodnoty skóre, protože se dělení magnitudou, která je u přítomných not malá.

Nejlepší ohodnocení má metoda Aliq okolo prahu s koeficientem 0,45. Tuto hodnotu jsem dále používal. Pro verzi s parametrem undertone jsem používal polovinu této hodnoty, tedy 0,225.

4.1.5 Použití metody

Vytvoření vlastní metody mi dalo volnost experimentovat s velikostí okna, které jsem vytvořil dynamické a frekvenční rozlišení jsem nesledoval, protože exponenciála je vytvořena přímo pro počítanou frekvenci.

Celkový postup detekce tónů je:

Načtení audia

Audio se načte a aplikuje se preemfáze.

```
def load_audio(path, preem=0.97):
  y, sr = librosa.load(path, sr=None)
  y = librosa.effects.preemphasis(y, coef=preem)
```

Detekce začátku tónů

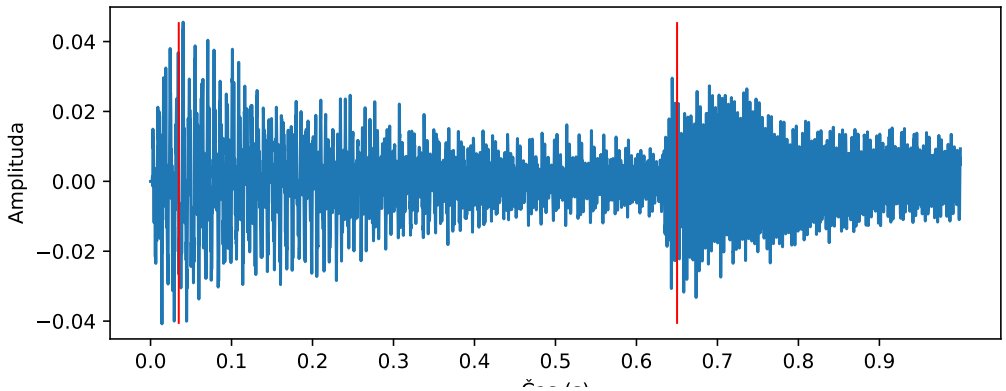
Pomocí knihovny librosa se detekují indexy vzorků, odpovídající času zahrání tónu. Takzvaná *onset detekce*.

```
def detect_samples(audio, sr):
  onset_env = librosa.onset.onset_strength(
    y=audio, sr=sr, hop_length=512, aggregate=np.median
  )
  detections = librosa.onset.onset_detect(
    onset_envelope=onset_env, sr=sr, hop_length=512, units="time"
  )

  return [int(i * sr) for i in detections]
```

Vytvoření oken

Vytvoření oken jsem udělal ze vzorků mezi dvěma detekcemi začátku tónu, nebo detekcí a koncem audia. Jedna úprava byla vhodná a to posunutí okna o náběh noty, anglicky *attack time*. Je to čas, kdy se intenzita signálu zvedá do svého maxima. Výsledek onset detekce překrytý se signálem je vidět na obrázku 4.1.



Obrázek 4.1: Výsledek onset detekce

Posunutím okna doleva o vhodně zvolený čas se okno vytvoří od začátku tónu a odstraní náběh tónu následujícího. Čas náběhu noty se pohybuje okolo 20 ms.

Je-li okno delší než 5 s, omezí se na prvních 5 s.

```
def get_window(y, sr, times, i):
  """
    y : array
      entire audio
    times : array
      indexes of y, where note was detected
    i : int
      which value from array times is processed
  """

  start_index = times[i]
  if start_index > int(sr * cnst["ATTACK_TIME"] / 1000):
    start_index -= int(sr * cnst["ATTACK_TIME"] / 1000)

  end_index = len(y)
  if len(times) > i + 1:
    end_index = times[i + 1] - int(sr * cnst["ATTACK_TIME"] / 1000)

  if end_index - start_index > sr * cnst["MAX_WINDOW_SIZE"]:
    end_index = start_index + sr * cnst["MAX_WINDOW_SIZE"]

  return start_index, end_index
```

Výpočet skóre pro detekci

Vytvořená okna se zpracují vytvořenou funkcí *custom_dft* 4.1. Pro každý detekovaný čas funkce vrátí vektor 88 hodnot, reprezentující skóre pro detekci jednotlivých not.

Vyhodnocení detekovaných tónů

Z vypočítaného vektoru 88 hodnot se vezme maximum. Všechny ostatní hodnoty, které jsou větší, než poměrná část této hodnoty, jsou brány jako přítomné noty.

```
  max = np.argmax(magnitudes)
  above_threshold = np.argwhere(
    magnitudes >= cnst["THRESHOLD"] * magnitudes[max]
    ).flatten()
    detected_notes.append({times[i] / sr: above_threshold.tolist()})
```

Výstupní formát

Výstup detekce je pole, obsahující slovníky s časem a indexy detekovaných not. Při porovnávání s MIDI souborem je potřeba myslet na to, že v MIDI soubor indexuje 128 různých tónů. Většinou se používá základních 88 tónů, které odpovídají standardní klaviatuře. Nejnižší používaná nota na klaviatuře je A_0, v MIDI souboru má index 21 [1].

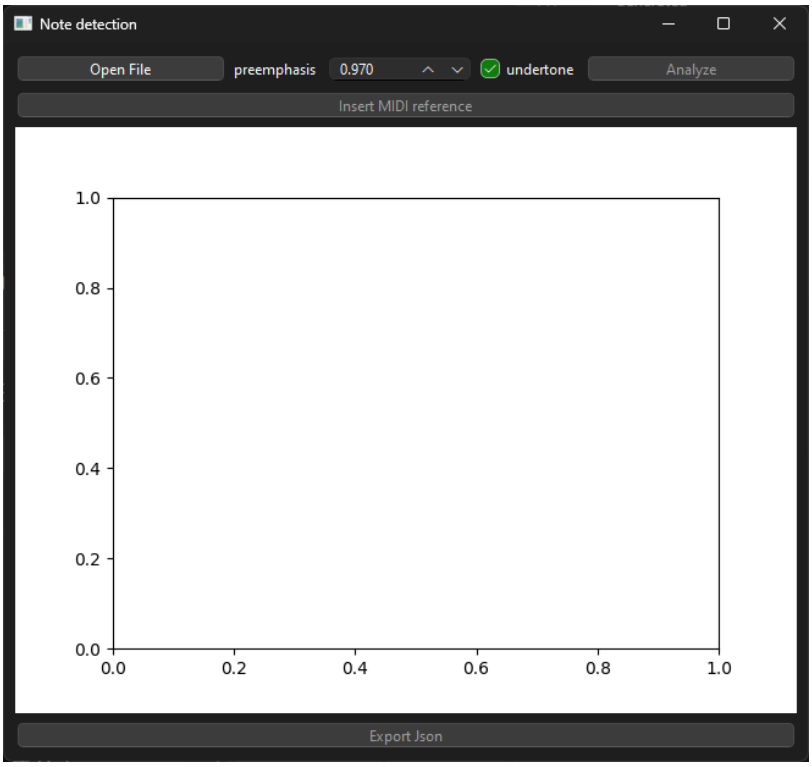
Python notace výstupu:

```
  List[Dict[float, List[int]]]
```

4.2 Grafické rozhraní

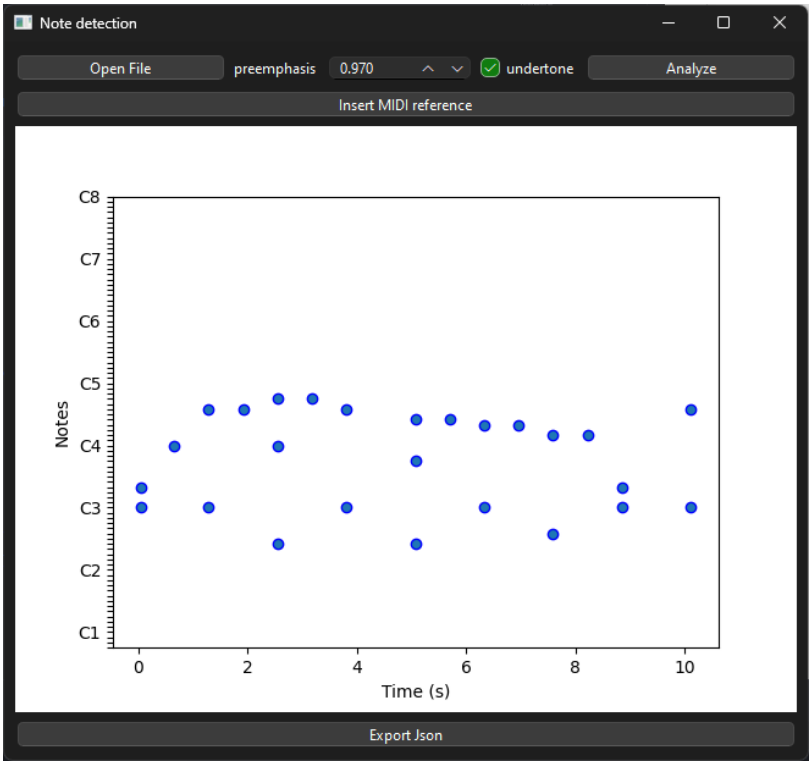
Pro náhled do detekce jsem vytvořil jednoduchou aplikaci. Její účel je vizualizace detekovaných not, použitého okna, porovnání s jinými výsledky (například Fourierovou transformací) a zobrazení midi souboru.

Výchozí stav aplikace je vidět na obrázku 4.2.



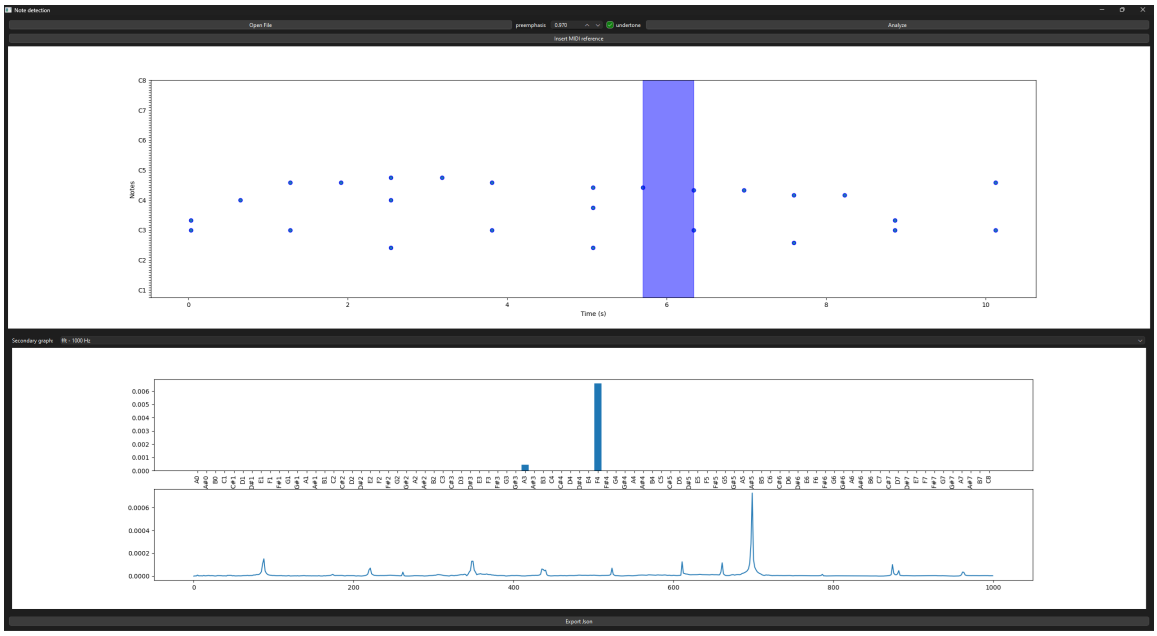
Obrázek 4.2: Aplikace po otevření

Vybere se audio soubor klikem na tlačítko `Open File`. Volitelně se nastaví koeficient preemfáze a použití parametru *undertone*. Po kliknutí `Analyze` se začne audio zpracovávat, po chvíli se aktualizuje graf, viz obrázek 4.3.



Obrázek 4.3: Zobrazení prvního grafu

Graf vizualizuje v čase nalezené začátky not. Mezi kterékoli noty lze kliknout a zobrazí se podrobnější informace, viz obrázek 4.4.



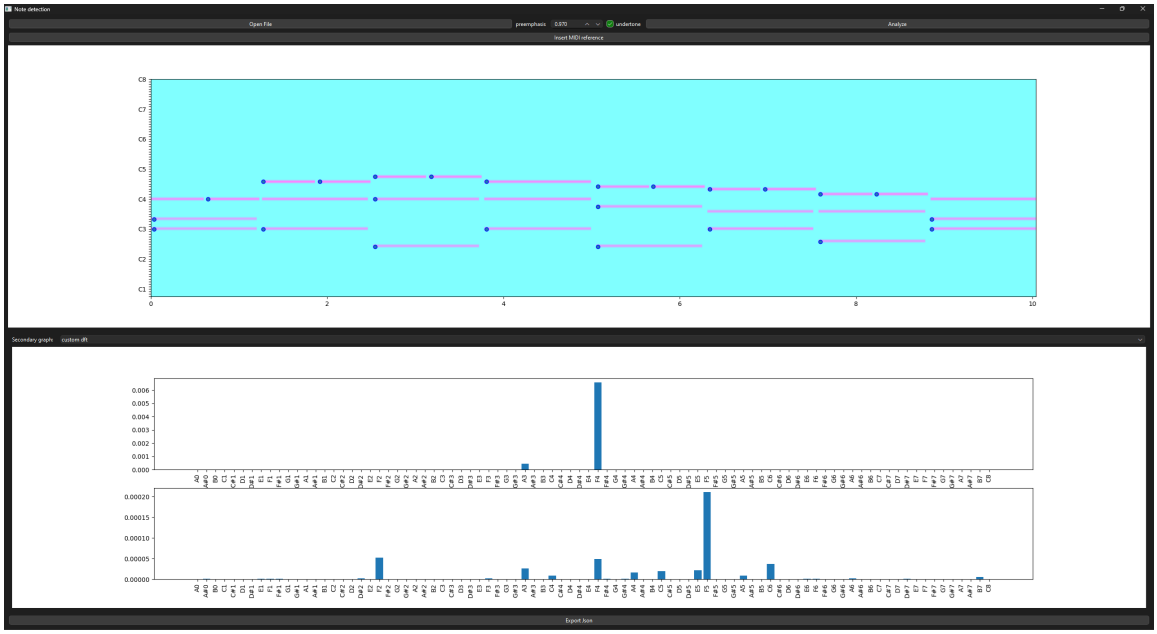
Obrázek 4.4: Detailnější informace

Teď jsou vidět dva panely, v prvním se zvýraznil vybraný segment, kliknutím do jiného místa lze segment přepnout.

Druhý panel se skládá z dvou podgrafů. První podgraf vždy ukazuje hodnoty z detekce alikvotními tóny pro vybraný segment. Obsah druhého grafu lze přepnout pomocí výběru, nacházejícím se mezi grafy. Na výběr je:

- `waveform`: Zobrazení signálu z vybraného segmentu,
- `fft - 1000 Hz`: Fourierova tranformace vybraného segmentu, omezena do 1000 Hz,
- `fft - 9000 Hz`: Fourierova tranformace vybraného segmentu, omezena do 9000 Hz,
- `custom dft`: Výsledek metody *custom_dft*, s parametrem *overtones=[1]*. Z tohoto grafu je vidět, jak velké magnitudy se mezi sebou násobí při výpočtu vrchního grafu,
- `onset times`: Zobrazení celého signálu a vizualizace onset detekce.

Tlačítko `Insert MIDI reference` vyzve uživatele k výběru MIDI souboru a vykreslí jej přes detekované noty. Je tak jednoduše vidět, které noty jsou detekovány správně, viz obrázek 4.5.



Obrázek 4.5: Vložení MIDI reference

První graf podporuje použití kolečka u myši pro přibližování a oddalování.

4.3 F1 score

Pro porovnání úspěšnosti detekčních metod je potřeba metody ohodnotit. Při vyhodnocování detekovaných not metoda F_1 *score* porovná detekované noty s referenčními daty tzv. *ground truth* a vyhodnotí počet výskytů v následujících kategoriích:

- *True Positive* - správně nalezená nota,
- *False Positive* - nota detekována tam, kde být neměla,
- *False Negative* - nota z referenčních dat nebyla detekována.

Z těchto kategorií se vypočítají dvě proměnné: *precision* a *recall*.

Precision je poměr správných detekcí ve všech detekcích systémem. Snižuje se tedy detekováním not navíc:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.6}$$

Recall je poměr správných detekcí ve všech referenčních notách v testovací sadě. Snižuje se tedy nedetekováním not:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.7}$$

Harmonická střední hodnota precision a recall udává hodnotu F_1 score:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.8}$$

4.3.1 Implementace v jazyce python

Funkce počítající F_1 score jsem pojmenoval *calculate_f_score* a nachází se v přiloženém souboru *Fscore.py*. Výstupem funkce je desetinné číslo a vstupy jsou následující:

- *detected_notes* - detekované noty ve složitější struktuře 4.1,
- *ground_truth_notes* - reference, pole hodnot ve formátu [index noty, čas],
- *tolerance* - tolerovaná odlišnost v čase (přesná detekce není možná).

Inicializace dat

Struktura detekovaných not je zbytečně složitá, převede se do stejné struktury, jako referenční data.

```
1  matched_gt_notes = set()
2  true_positives = 0
3  false_positives = 0
4  false_negatives = 0
5
6  # prevod detekovanych not do jednodussi struktury
7  detected_flat = []
8  for note_dict in detected_notes:
9    for start_time, pitches in note_dict.items():
10     for pitch in pitches:
11       detected_flat.append((pitch, start_time))
```

Vyhodnocení kategorií

V cyklu se každá detekovaná nota hledá v referenčních datech.

```
# iteruje se pres detekovane noty
for d_pitch, d_start in detected_flat:
  match_found = False
  # kontroluje se pres referencni data
  for index, (gt_pitch, gt_start) in enumerate(ground_truth_notes):
    # cas a nota odpovidaji -> TP
    if d_pitch+OFFSET == gt_pitch and abs(d_start - gt_start) <= tolerance:
      if index not in matched_gt_notes:
        true_positives += 1
        matched_gt_notes.add(index)
        match_found = True
        break
  # detekovana nota nenalezena v~referencnich datech -> FP
  if not match_found:
    false_positives += 1

  # FN se rovna rozdilu poctu referenci a TP
  false_negatives = len(ground_truth_notes) - len(matched_gt_notes)
```

Výpočet F1 score

Podle vzorců 4.3 se vypočítá precision, recall a F_1 score.

```
precision = (
  true_positives / (true_positives + false_positives)
  if (true_positives + false_positives) > 0
  else 0
)
recall = (
  true_positives / (true_positives + false_negatives)
  if (true_positives + false_negatives) > 0
  else 0
)
f_s-core = (
  2 * (precision * recall) / (precision + recall)
  if (precision + recall) > 0
  else 0
)
```

4.3.2 Testování metod

Vytvořil jsem několik MIDI nahrávek a vzal pár originálních skladeb. Pomocí F_1 score jsem každou nahrávku ohodnotil pro každou z implementovaných detekčních metod: STFT 3.2, CQT 3.3 a vlastní metodou 4.1 (s a bez použití parametru *undertone*).

Některé nahrávky jsou zaměřené na různé charakteristiky pro specifické otestování. Ostatní jsou reálné skladby. U ručně hraných nahrávek lze předpokládat odlišné intenzity not a časová nepřesnost:

Název	**Počet not**	**Délka**	**Hráno ručně**	**Charakteristika**
Bach_BWV846	754	2:29	Ne	–
ttls	90	0:33	Ne	–
MS_1st_mov	144	0:47	Ano	–
low	89	1:15	Ano	Náhodné noty z 0. a 1.oktávy
high	55	0:54	Ano	Náhodné noty ze 7. a 8.oktávy
harmony	97	0:48	Ano	3+ harmonické noty zároveň

Tabulka 4.2: Tabulka MIDI nahrávek

Těchto šest MIDI souborů jsem syntetizoval pomocí zvukové banky piána: *Essential Keys-sforzando-v9.6* [4] a analyzoval všemi implementovanými technikami. *Aliq* je zkratka pro vlastní metodou pomocí alikvotních tónů 4.1, $Aliq_u$ je tatéž metoda s parameterem *undertone=True* 4.1.3.

Název audia	Technika	TP	FP	FN	Precision	Recall	F1-Score
bach_BWV846	STFT	338	500	416	0.4033	0.4483	0.4246
	CQT	228	497	526	0.3145	0.3024	0.3083
	Aliq	395	369	359	0.5170	0.5239	**0.5204**
	Aliq_u	422	495	332	0.4602	0.5597	0.5051
ttls	STFT	73	72	17	0.5034	0.8111	0.6213
	CQT	61	45	29	0.5755	0.6778	0.6224
	Aliq	80	15	10	0.8421	0.8889	**0.8649**
	Aliq_u	79	16	11	0.8316	0.8778	0.8541
MS_1st_mov	STFT	43	69	101	0.3839	0.2986	0.3359
	CQT	20	36	124	0.3571	0.1389	0.2000
	Aliq	50	59	94	0.4587	0.3472	**0.3953**
	Aliq_u	53	84	91	0.3869	0.3681	0.3772
low	STFT	0	104	89	0.0000	0.0000	0.0000
	CQT	0	34	89	0.0000	0.0000	0.0000
	Aliq	18	158	71	0.1023	0.2022	**0.1358**
	Aliq_u	17	206	72	0.0762	0.1910	0.1090
high	STFT	34	169	21	0.1675	0.6182	**0.2636**
	CQT	0	235	55	0.0000	0.0000	0.0000
	Aliq	0	776	55	0.0000	0.0000	0.0000
	Aliq_u	2	835	53	0.0024	0.0364	0.0045
harmony	STFT	65	35	32	0.6500	0.6701	0.6599
	CQT	71	64	26	0.5259	0.7320	0.6121
	Aliq	54	11	43	0.8308	0.5567	**0.6667**
	Aliq_u	55	13	42	0.8088	0.5670	**0.6667**

Tabulka 4.3: Výsledky analýzy nahrávek metodou F_1 score

Z tabulky je vidět:

- Počty FP a FN jsou blízko sebe. Znamená to, že citlivost detekce je dobře nastavena,
- velmi nízké tóny se špatně detekují. Pouze metoda alikvotních tónů nalezla pár not.
- velmi vysoké tóny se špatně detekují. Pouze STFT nalezla pár správných detekcí. Alikvotní tóny se pro vlastní metodu nachází příliš vysoko.

Výsledky metod lze ovlivnit změnou různých parametrů. U STFT například úpravou: koeficientu preemfáze, trojúhelníkové funkce, vektoru pro vyhlazení a dalšími. CQT především změnou počtu frekvenčních košů. Vlastní metodu použitím jiného okna, nebo alikvotních tónů.

Všechny metody nějakým způsobem používají práh, ten je vytvořený z části maximální nalezené hodnoty. Změnou tohoto prahu je možné změnit citlivost detekce a prioritizovat tak jednu z kategorií FP/FN, tím i precision/recall.

Kapitola 5

Závěr

Cílem mé bakalářské práce bylo experimentovat s metodami zpracování signálů. Především s diskrétní Fourierovou transformací (konkrétněji STFT) a konstantní Q-transformací. Také použití okénkových funkcí, preemfáze a objektivní ohodnocení výsledků metrikou F_1 score.

Cíle práce se mi podařilo naplnit. Při experimentování jsem si všiml přítomnosti alikvotních tónů. Po pochopení standardních metod analýzy signálu mne napadlo tyto alikvotní tóny využít a postavit na základě jejich přítomnosti vlastní metodu. Výsledek vlastní metody po otestování hodnotím pozitivně. Ve srovnání s ostatními si metoda vedla lépe. Je ale potřeba myslet na to, že ostatní metody nelze použít v základním stavu a musel jsem okolo nich postavit vlastní logiku, abych tóny detekoval. Přestože jsem se metody snažil dle svého uvážení použít co nejefektivněji, nemusí se jednat o ideální řešení.

Jednoduché GUI umožňuje náhled do pozadí detekce. Uživatel se může podívat, jaké hodnoty v konkrétní čas různé metody vypočítaly a umožňuje přes detekovaná data vizualizovat MIDI data. V tomto typu zobrazení je vidět, kde se chybné detekce nacházejí.

V tématu lze dále pokračovat. Možné rozšíření by mohlo být odchycení doznívajících tónů a zredukovat tak počet detekovaných tónů navíc. Toto rozšíření ale není tak jednoduché, jak by se mohlo zdát. Aktivace jednoho tónu totiž zvedne hodnoty v celém spektru a není zřejmé, jestli doznívající nota byla zahrána znovu, nebo její hodnotu posílil jiný tón.

Uvědomil jsem si, že detekce tónů není jednoduché téma, hraje v něm roli spousta proměnných, vycházející například z typu skladby, nebo charakteristiky hraného nástroje. Z komplexnějšího audia tak není možné tóny detekovat s velkou přesností.

Literatura

[1] Acoustics, I. *MIDI note numbers and center frequencies*. 2023. Dostupné z: https://inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies.

[2] McFee, B., Raffel, C., Liang, D., Ellis, D. P., McVicar, M. et al. Librosa: Audio and music signal analysis in python. In: *Proceedings of the 14th python in science conference*. 2015, sv. 8.

[3] McFee, B., Raffel, C., Liang, D., Ellis, D., McVicar, M. et al. Librosa: Audio and Music Signal Analysis in Python. In: *14th annual Scientific Computing with Python conference*. July 2015. SciPy. DOI: https://doi.org/10.25080/Majora-7b98e3ed-003.

[4] Nebauer, J. *Soundfonts 4U*. 2016. Dostupné z: https://sites.google.com/site/soundfonts4u/.

[5] Tjoa, S. *Music information retrieval*. 2014. Dostupné z: https://musicinformationretrieval.com.

[6] Bevelacqua, P. *Fourier Transform*. 2010. Dostupné z: https://www.thefouriertransform.com.

[7] DISW, S. *What is the Fourier Transform?* 2019. Dostupné z: https://community.sw.siemens.com/s/article/what-is-the-fourier-transform.

[8] Bäckström, T., Räsänen, O., Zewoudie, A., Zarazaga, P. P., Koivusalo, L. et al. *Pre-emphasis*. 2. vyd. 2022. DOI: 10.5281/zenodo.6821775. Dostupné z: https://speechprocessingbook.aalto.fi/Preprocessing/Pre-emphasis.html.

[9] Blankertz, B. *The Constant Q Transform*. Technische Universität Berlin, 2024. Dostupné z: https://doc.ml.tu-berlin.de/bbci/material/publications/Bla_constQ.pdf.

[10] Brown, J. C. Calculation of a Constant Q Spectral Transform. In: *Journal of the Acoustical Society of America*. 1991, sv. 89, č. 1, s. 425–434. Dostupné z: https://www.ee.columbia.edu/~dpwe/papers/Brown91-cqt.pdf.

Příloha A

Obsah přílohy

```
+-- app.py                 : Aplikace ke spusteni
+-- core.py                : Soubor, obsahujici potrebne funkce k analyze (backend)
+-- Fscore.py              : Implementace F1−score a testovani
+-- plot_creator.ipynb     : Notebook, generujici obrazky pouzite v technicke zprave
+-- requirements.txt       : Seznam python knihoven, potrebnych ke spusteni
+-- generated/             : Slozka, do ktere se uklada syntetizovane audio
|   +-- ...
+-- resources/             : Slozka se zdrojovymi MIDI soubory
|   +-- bach_BWV846.mid
|   +-- c4.mid
|   +-- high.mid
|   +-- low.mid
|   +-- MS_1st_mov.mid
|   +-- ttls_short.mid
|   +-- ttls.mid
+-- TZ/                    : Technicka zprava
    +-- xkrist24.tex       : Zdroj technicke prace
    +-- xkrist24.pdf       : vygenerovane pdf
    +-- resources/         : Pouzite obrazky (.pdf)
    |   +-- ...
    +-- screenshots/       : Pouzite snimky z aplikace (.png)
        +-- ...
    +-- ...
```