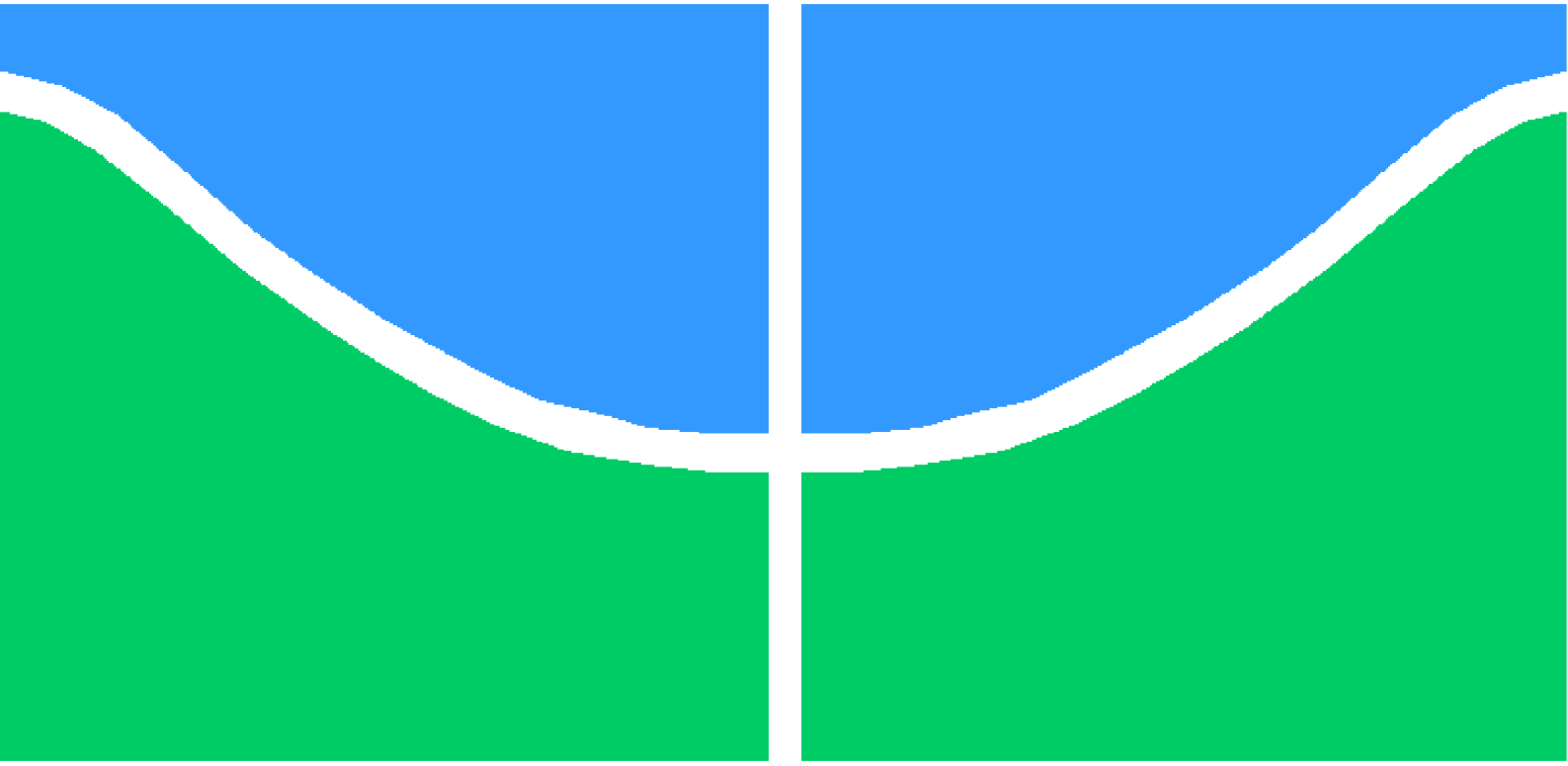

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Solução Computacional para Reconhecimento de Harmonias Musicais

Autor: José Pedro de Santana Neto
Orientador: Dr. Henrique Gomes de Moura

Brasília, DF
2014

José Pedro de Santana Neto

Solução Computacional para Reconhecimento de Harmonias Musicais

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Dr. Henrique Gomes de Moura

Coorientador: Dr. Paulo Roberto Miranda Meirelles

Brasília, DF

2014

José Pedro de Santana Neto
Solução Computacional para Reconhecimento de Harmonias Musicais/ José Pedro de Santana Neto. – Brasília, DF, 2014-
134 p. : il. (algumas color.) ; 30 cm.

Orientador: Dr. Henrique Gomes de Moura

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2014.

1. Reconhecimento. 2. Acordes. I. Dr. Henrique Gomes de Moura. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Solução Computacional para Reconhecimento de Harmonias Musicais

CDU 02:141:005.6

José Pedro de Santana Neto

Solução Computacional para Reconhecimento de Harmonias Musicais

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, 18 de novembro de 2014:

Dr. Henrique Gomes de Moura
Orientador

Dr. Fernando William Cruz
Convidado 1

Dr. Cristiano Jacques Miosso Rodrigues Mendes
Convidado 2

Brasília, DF
2014

Esse trabalho é dedicado às pessoas que quebram coisas para verem como são por dentro.

Agradecimentos

Agradeço primeiramente a Deus, inteligência criadora suprema, por permitir-me nesse mundo, vivendo, aprendendo e contemplando a beleza da natureza primordial de todas as coisas.

A minha amada e querida mãe Francisca, pela paciência, compreensão, tolerância, conselhos, carinho, dedicação, afeto, amizade, silêncio, sorrisos e um intenso amor. Meu primeiro aprendizado na vida mais puro e original de amor foi através dela. Isso me possibilitou a amar verdadeiramente o que faço e ter uma visão de vida mais profunda. Meus sinceros e eternos agradecimentos.

A meu pai Luciano, mesmo não estando presente mais, me inspirou a escolha da minha formação e me ensinou a olhar o mundo com meus próprios olhos.

A meu irmão João, companheiro e amigo de sempre. Seus conselhos e seu exemplo têm me ensinado muito a ser uma pessoa melhor.

A meu padrinho Inácio, meu segundo pai, por seus profundos conselhos sobre a vida e um exemplo para mim de homem honrado e correto.

A meu tio Antônio, por ser o padrinho da minha mãe e assumir o papel de avô na minha vida.

A minha querida tia Naíde, por ter assumido papel de avó na minha vida e ter tido um olhar único sobre minha vida.

A minha madrinha Nevinha, minhas tias Titia, Tia Marli, Tia Gracinha e Tia Bá. O amor delas é indescritível com palavras. A toda minha família pelo apoio, confiança e compressão.

A meus amigos-irmãos Thiago e Leandro, pelo companheirismo indescritível de muitos anos e apoio de sempre.

A minha amiga Marina Shinzato, pelas longas conversas e por ter sido meu ombro forte ao longo desse trabalho.

A minha amiga Anaely, pelo apoio compreensão e inspiração.

A minha amiga Ana Luisa, pelas longas conversas e incrível amizade.

A meus professores de música Boggie e Gedeão por todo conhecimento e inspiração musical.

A meu orientador professor Henrique Moura, pelo exemplo, inspiração, amizade, conselhos, apoio, confiança e investimento de longas conversas. Esse trabalho necessaria-

mente foi fruto de uma orientação em excelência.

A meu co-orientador professor Paulo Meirelles, pelo exemplo e ensinamentos valiosos e práticos sobre o mundo do software e a vida.

Aos professores Hilmer, Milene, Maria de Fátima, Cristiano e Fernando pelos valiosos ensinamentos e exemplos de profissionais-cientistas.

A equipe do LAPPIS pelo suporte e aprendizado na produção de softwares de qualidade.

A professora Suzete e a equipe do MídiaLab por todo aprendizado.

Aos meus amigos da faculdade e companheiros de disciplinas Carlos, Álvaro, Fagner, Eduardo, Wilton, João, Daniel, Matheus, Kleber, Hebert, André Guedes, David, Yeltsin, Wilker, Thaiane, Tomaz, Maxwell, Luiz Oliveira e André Mateus, pela compreensão, apoio e motivação.

Aos meus restantes amigos Luiz Matos, Fábio Costa, Daniel Bucher, Renan, Chico, Leônidas, Lucas, Nayron, Thiago Ribeiro, Marcos Ramos, Cleiton, Marcos Ronaldo, José Alisson, José Alberto, Vilmey, Yan, Igor Josafá, Guilherme Fay, Sérgio, Lucas Kanashiro, Charles Oliveira, Rodrigo, Álex, Jefferson, Alexandre, Matheus Souza, Ana Luiza e outros que esqueci de citar, pelo apoio e zueira de sempre.

E as pessoas que passaram na minha vida e influenciaram de alguma forma nesse trabalho. Meus agradecimentos.

"A vida não é uma sonata que para
realizar sua beleza tem de ser tocada até o fim,
ao contrário, a vida é um álbum de minissonatas.
Cada momento de beleza vivido e amado,
por efêmero que seja, é uma experiência completa
que está destinada à eternidade.
Um único momento de beleza e amor
justifica a vida inteira."
(Rubem Alves)

Resumo

Atualmente a música está num patamar único no que diz respeito a várias abordagens de se contemplar e se executar e, com isso, a tecnologia vem cada vez mais se tornando uma abordagem de interação com os processos musicais. Um dos exemplos de tecnologia são sistemas automáticos de transcrição de música que auxiliam o músico, substituindo por vezes de maneira significativa partituras, tablaturas e cifras. Esse presente trabalho tem como objetivo desenvolver uma solução computacional para reconhecimento de harmonias musicais. Para tal fim focou-se na implementação da análise espectral da amostra de áudio, classificação em notas musicais, classificação em acordes com suportes a inversões, transição rítmica e reconhecimento dos padrões harmônicos ao longo do tempo. O desenvolvimento da solução se deu sobre uma perspectiva transdisciplinar (teoria da complexidade) com o auxílio da metodologia científica, utilizando a linguagem de programação Scilab para implementação. De fundamentos teóricos foram utilizados conceitos físicos do som, teoria musical, processamento de sinais e redes neurais artificiais. O desenvolvimento da solução permitiu a detecção de acordes em tríades maiores, menores, aumentados e diminutos numa amostra de áudio.

Palavras-chaves: reconhecimento. acordes. música. processamento. sinais. redes. neurais. harmonia.

Abstract

Currently the song is a single level with regard to various approaches to behold and run and, therefore, the technology is increasingly becoming an interaction approach with the musical processes. One of the technology examples are automatic music transcription systems that help the musician, replacing sometimes significantly scores, tabs and chords. This present study aims to develop a computational solution for recognition of musical harmonies. For this purpose focused on the implementation of spectral analysis of the audio sample, classification of musical notes, chord classification with support inversion, recognition of rhythmic and harmonic transition patterns over time. The development of the solution took on a transdisciplinary perspective (complexity theory) with the help of scientific methodology, using Scilab programming language for implementation. Of theoretical foundations were used physical concepts of sound, music theory, signal processing and artificial neural networks. The development of the solution allowed the detection of chords in major triads, minor, augmented and diminished in an audio sample.

Key-words: recognition. chords. music. processing. signals. networks. neural.

Lista de ilustrações

Figura 1 – Função da Equação 2.3 28
Figura 2 – Função da Equação 2.4 29
Figura 3 – Distribuição das frequências nas notas musicais em Hz 31
Figura 4 – Modelo de um neurônio 36
Figura 5 – Modelo arquitetural da PNN 37
Figura 6 – Diagrama de Fluxo de Dados 42
Figura 7 – Modelo de Ciclo Adaptado 52
Figura 8 – Teclado ilustrativo para execução dos acordes 69
Figura 9 – Processo ilustrativo da execução dos experimentos 69
Figura 10 – Gráfico da resposta em frequência para a gravação do acorde CM . . . 70
Figura 11 – Gráfico de sugestão de notas para a gravação do acorde CM 70
Figura 12 – Gráficos de sugestão de acordes a gravação do acorde CM 71
Figura 13 – Gráfico da resposta em frequência para a gravação do acorde Dm . . . 72
Figura 14 – Gráfico de sugestão de notas para a gravação do acorde Dm 72
Figura 15 – Gráficos de sugestão de acordes a gravação do acorde Dm 73
Figura 16 – Gráfico da resposta em frequência para a gravação do acorde $Ddim$ . . 74
Figura 17 – Gráfico de sugestão de notas para a gravação do acorde $Ddim$ 74
Figura 18 – Gráficos de sugestão de acordes a gravação do acorde $Ddim$ 75
Figura 19 – Gráfico da resposta em frequência para a gravação do acorde $Daum$ . . 76
Figura 20 – Gráfico de sugestão de notas para a gravação do acorde $Daum$ 76
Figura 21 – Gráficos de sugestão de acordes a gravação do acorde $Daum$ 77

Lista de tabelas

Tabela 1 – Tabela de resultados dado os acordes tocados com inversões 79

Lista de abreviaturas e siglas

aum	Acorde aumentado
dim	Acorde diminuto
M	Acorde maior
m	Acorde menor
A	Lá
B	Si
C	Dó
D	Ré
E	Mi
F	Fá
G	Sol
#	Sustenido
b	Bemol
Hz	Hertz
db	Decibéis
.wav	Formato de arquivo WAVE

Sumário

1 Introdução 23
1.1 Contexto 23
1.2 Problemática 23
1.3 Objetivos 25
1.4 Organização do Trabalho 25
2 Fundamentos Teóricos 27
2.1 Conceitos Físicos do Som 27
2.2 Conceitos Musicais 29
2.3 Conceitos de Processamento de Sinais 33
2.4 Conceitos de Redes Neurais Artificiais 34
3 Desenvolvimento da Solução 39
3.1 Metodologia 39
3.2 Técnicas Utilizadas para Desenvolvimento do Sistema-Solução 42
3.2.1 Procedimento 1: Separar Janelas de 1 Segundo em 5 Partes 43
3.2.2 Procedimento 2: Aplicar Janelas de Blackman 43
3.2.3 Procedimento 3: Calcular o Espectro de Frequências 45
3.2.4 Procedimento 4: Adquirir Energias das Notas 46
3.2.5 Procedimento 5: Binarizar Energia das Notas 47
3.2.6 Procedimento 6: Extrair Baixos 47
3.2.7 Procedimento 7: Extrair Tonalidade 48
3.2.8 Procedimento 8: Extrair Acordes Fundamentais 49
3.2.9 Procedimento 9: Extrair Acorde Recorrente das Partes 50
3.2.10 Procedimento 10: Extrair Acordes com Inversões 51
3.3 Linha de Ciclos de Desenvolvimento 52
3.3.1 Estrutura do Ciclo 52
3.3.2 Ciclo 1 53
3.3.3 Ciclo 2 54
3.3.4 Ciclo 3 54
3.3.5 Ciclo 4 56
3.3.6 Ciclo 5 57
3.3.7 Ciclo 6 58
3.3.8 Ciclo 7 59
3.3.9 Ciclo 8 60
3.3.10 Ciclo 9 61
3.3.11 Ciclo 10 62
3.3.12 Ciclo 11 64

3.3.13 Ciclo 12 . . . 65
4 Resultados . . . 67
4.1 Resposta em Frequência e Sugestões de Notas e Acordes . . . 67
4.1.1 Pré-condições dos Experimentos . . . 68
4.1.2 Experimento 1 - Acorde CM . . . 70
4.1.3 Experimento 2 - Acorde Dm . . . 72
4.1.4 Experimento 3 - Acorde $Ddim$ . . . 74
4.1.5 Experimento 4 - Acorde $Daum$ . . . 76
4.1.6 Tabela de resultados dos acordes tocados . . . 78
4.2 Detecção de Transições Rítmicas . . . 78
4.3 Implementação da Transformada Wavelets . . . 78
4.4 Transcrição de Notas ao Longo do Tempo . . . 78
4.5 Extração da Tonalidade do Áudio . . . 78
4.6 Transcrição Automática de Acordes ao Longo do Tempo . . . 78
5 Considerações Finais . . . 81
5.1 Evoluções Futuras . . . 81

Referências . . . 83

Apêndices 85
APÊNDICE A Primeiro Apêndice . . . 87
A.1 Módulo Principal . . . 87
A.2 Módulo de Monoficação do Sinal de Áudio . . . 88
A.3 Módulo da Transformada de Fourier . . . 88
A.4 Módulo de Equalização do Sinal . . . 90
A.5 Módulo de Correlação de Notas . . . 90
A.6 Módulo de Correlação . . . 90
A.7 Módulo de Correlação de Acordes . . . 91
A.8 Módulo de Interpretação de Correlação dos Acordes . . . 92
A.9 Módulo de Alocação de Constantes para Notas . . . 95
A.10 Módulo de Alocação de Constantes para Acordes . . . 111
A.11 Módulo de Testes em Amostras . . . 122
A.12 Cronograma para Próximas Atividades . . . 134

1 Introdução

1.1 Contexto

Atualmente a música está num patamar único no que diz respeito a várias abordagens de se contemplar e se executar. A tecnologia vem cada vez mais se tornando uma abordagem de interação com os processos musicais (THÉBERGE, 1997). Desde sintetizadores eletrônicos até afinadores programados em software, a música vem acompanhando o desenvolvimento técnico-científico.

Um bom exemplo de impacto direto da tecnologia sobre a música é o software Auto-Tune (TYRANGIEL, 2009). O software é um editor de áudio em tempo real criado pela empresa Antares Audio Technologies (ANTARES, 2014) para afinar instrumentos e vozes. Muitos cantores e artistas usam desse software para poder executar as músicas com mais afinação em apresentações e gravações. Exemplos de artistas que usam são Rihanna, Justin Bieber, Demi Lovato, Bruno Mars, Kelly Clarkson e Lady Gaga. De fato milhares de pessoas são impactadas pelo resultado do trabalho desse software.

Outro exemplo de software impactante na música é o afinador Tuner-gStrings (COHORTOR.ORG, 2014). Ele é um aplicativo da plataforma para dispositivos móveis Android que permite a afinação de quaisquer instrumentos musicais. Na loja virtual Google Play ele está com 4,6 de 5 estrelas em 155.957 avaliações e o número de instalações entre 10.000.000 e 50.000.000 no mundo inteiro [1].

Ambos softwares apresentados são ferramentas de suporte para o músico poder executar corretamente as músicas e facilitar muito trabalho que seria de natureza manual. O presente trabalho tem como foco apresentar uma solução computacional de uma possível ferramenta de suporte ao músico.

1.2 Problemática

Os músicos em geral sempre necessitaram do conhecimento de informações sobre as músicas com o intuito de serem melhor executadas. Informações do tipo de compasso, tom, harmonia, escalas utilizadas, andamento, expressões e variações de tempo. Especificamente obter a noção de harmonia e tom das músicas é de grande valor no que diz respeito a instrumentos melódicos e jazzistas (MONSON, 2009).

Normalmente as informações de harmonia e tom são inferidas na partitura e tablaturas por regras simples como a primeira nota que começa e termina a música ou como

[1] Dados levantados: https://play.google.com/store/apps/details?id=org.cohortor.gstrings

o primeiro acorde que começar e terminar. Também são utilizados noções de escalas e acidentes para inferir que tais notas realmente pertencem a um determinado tom.

Essas técnicas são efetivas se no caso houver partituras, tablaturas ou cifras.Também há a possibilidade, mas somente para quem tem um ouvido bastante treinado, de ouvir melodias e harmonias e poder extrair informações de acordes e tom. Poucas pessoas possuem essa habilidade de discernir notas, tons e harmonia apenas ouvindo o som.

Em vista desse contexto, sistemas automáticos de transcrição de música (KLAPURI, 2004) são perfeitamente adequados a atender as necessidades de extração de informações relevantes numa dada faixa de áudio.

No que se trata diretamente sobre harmonias e acordes, existem poucos estudos publicados sobre um sistema de reconhecimento. Um dos poucos publicados é baseado num método utilizado em tecnologias 3G CDMA para dispositivos móveis (BARBANCHO, 2010). É bastante interessante a correlação que o estudo faz de notas tocadas com clientes CDMA's. Entretanto a técnica CDMA de cancelamento de interferência paralelo (PIC) possui limitações quando se aplica a notas musicais devido a natureza não ortogonal das mesmas (em clientes CDMA são ortogonais entre si, já para notas musicais há o problema dos seus respectivos harmonônicos, que não são ortogonais entre si).

No ponto de vista também da aprendizagem musical há uma motivação para a criação de um reconhecedor de harmonias, dado que muitos iniciantes não sabem os acordes corretos para cada posição do instrumento, como também, os ouvidos são pouco treinados. Um sistema capaz de auxiliar no reconhecimento das harmonias seria de grande ajuda para o aprendiz abstrair os padrões musicais.

Em visto do que foi exposto, um reconhecimento de harmonias musicais facilitaria a atuação do músico por informar acordes e notas. Isso é muito bom pois substituiria parcialmente o uso das partituras, tablaturas e cifras além de funcionar como um ótimo guia para solistas e improvisadores (principalmente jazzistas).

O reconhecimento de harmonias musicais no ponto de vista computacional é inerentemente complexo. Primeiramente deve-se achar uma forma de representação das amostras de áudio em termos de frequências, a ferramenta para esse tipo de atividade deverá fazer uma transformação da informação que está no domínio temporal para o domínio frequencial. Em seguida é preciso identificar notas musicais em meio ao espectro de frequências calculado, focando cobrir com acurácia relações de bandas de frequência com as notas em si. Tendo convertido o espectro de frequência em notas musicais é preciso definir a classificação de acordes dado um embasamento teórico-musical definido, mapeando as combinações possíveis de tríades com os respectivos acordes. E, por fim, é preciso consolidar uma solução para reconhecimento de acordes ao longo de uma música e, para tanto, deverá ser contemplado uma implementação que identifica transições rítmicas e

reconhecimento de harmonias ao decorrer da música.

1.3 Objetivos

O presente trabalho tem como objetivo principal desenvolver uma solução computacional para reconhecimento de harmonias musicais.

Como objetivos específicos têm-se implementação das soluções em:

- desenvolver solução computacional para reconhecimento de acordes e suas inversões;
- desenvolver solução computacional para reconhecimento de acordes ao longo do tempo;
- desenvolver solução computacional para extração do tom da música.

1.4 Organização do Trabalho

Esse trabalho está organizado em capítulos. O capítulo 2 apresenta a metodologia de desenvolvimento da solução. O capítulo 3 apresenta um arcabouço de fundamentos teóricos físicos, musicais, de processamento de sinais e de redes neurais. O capítulo 4 apresenta o desenvolvimento da solução. O capítulo 5 apresenta resultados parciais de experimentos feitos com a solução computacional proposta. O capítulo 6 apresenta as conclusões e evoluções futuras. Segue no final referências bibliográficas e apêndice com o código da solução.

2 Fundamentos Teóricos

Nesse presente capítulo será introduzido conceitos teóricos para a definição de axiomas e ferramentas do contexto. Será abordado primeiramente conceitos físicos no que tange a natureza do som como propagação, formação e dinâmica. Após será exposto fundamentos sobre notas musicais e harmonia. Enfim, então, será apresentado teorias computacionais de processamento de sinais e redes neurais de classificação.

2.1 Conceitos Físicos do Som

O som pode ser visto como uma perturbação mecânica nas moléculas do meio, uma frente de compressão variável de perfil mecânico e longitudinal com velocidade e pressão. O meio de propagação do som pode ser de diversas naturezas como por exemplo sólido, líquido e gasoso. Dessa perturbação mecânica entende-se como variação de pressão em relação ao tempo e espaço (SANTOS, 2008). Em vista disso, a equação diferencial que expressa o comportamento do som é a de natureza ondulatória $\mathbf{p = p(x,t)}$:

$$\frac{\partial^2 \mathbf{p}}{\partial x^2} = \frac{1}{c^2} \cdot \frac{\partial^2 \mathbf{p}}{\partial t^2} \tag{2.1}$$

Na qual $\mathbf{p}$ é a pressão, $\mathbf{x}$ é a localização longitudinal, $\mathbf{c}$ é a velocidade do som e $\mathbf{t}$ é a localização temporal. A solução harmônica para a Equação (2.1) é definida por:

$$\mathbf{p}(\mathbf{x}, \mathbf{t}) = \mathbf{A}.exp^{j(wt-kx)} + \mathbf{B}.exp^{j(wt+kx)} \tag{2.2}$$

Em que $\mathbf{k}$ é dado por $\mathbf{w/c}$ e as constantes complexas $\mathbf{A}$ e $\mathbf{B}$ são utilizadas para condições de contorno.

Uma solução simples para a equação de onda 2.1 é a seguinte:

$$\mathbf{p(t)} = \mathbf{1}.cos(2.\pi.440.t) \tag{2.3}$$

Nela a variável **x** foi fixada na origem do sistema cartesiano e o comportamento da onda só está sendo analisado em relação a variável temporal **t**. Há de considerar de suma importância a fixação frequencial **w** em 440 Hz. Em termos musicais essa nota equivale ao Lá de tom puro, ou seja, nota construída artificialmente sem harmônicos somados (diferentemente dos instrumentos reais que possuem os harmônicos). Um exemplo de gráfico gerado por essa função é dado por:

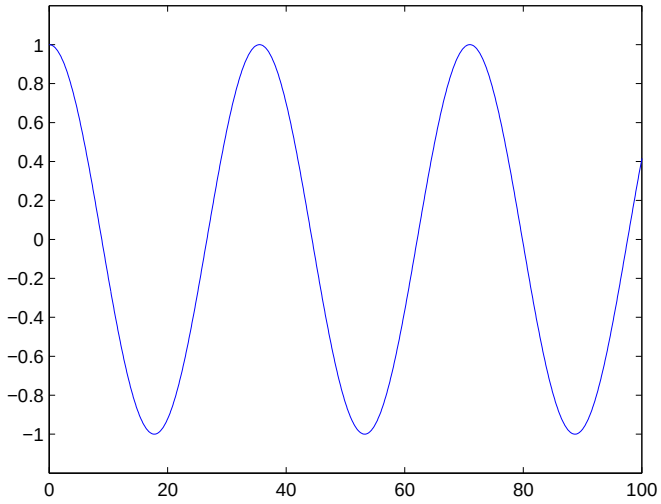

Figura 1 – Função da Equação 2.3

Na forma mais teórica, um acorde, como será apresentado a seguir, é composto de no mínimo 3 ondas sonoras somadas, ou seja, para que seja formado um acorde Am por exemplo a equação total deverá ter essa forma:

$$\mathbf{p(t)} = \mathbf{1}.cos(2.\pi.440.t) + \mathbf{0,5}.cos(2.\pi.523.t) + \mathbf{1}.cos(2.\pi.660.t) \tag{2.4}$$

Pode-se considerar que cada constante multiplicadora das função cosseno determinará a energia de uma onda sonora específica. Nesse caso as ondas sonoras de mais energia são as de 440 Hz e 600 Hz. A onda de menor energia é a de 523 Hz que possui a metade da energia das outras. Esse fato será decisivo para a detecção de acordes. É pertinente comentar que esse acorde montado é um modelo simplificado pois possuem somente tons puros (sem harmônicos somados). No contexto da solução trabalhada será considerado também acordes com harmônicos somados assim como é nos instrumentos reais. Esse fato aumenta a complexidade da solução.

Em termos de representação gráfica segue o resultado:

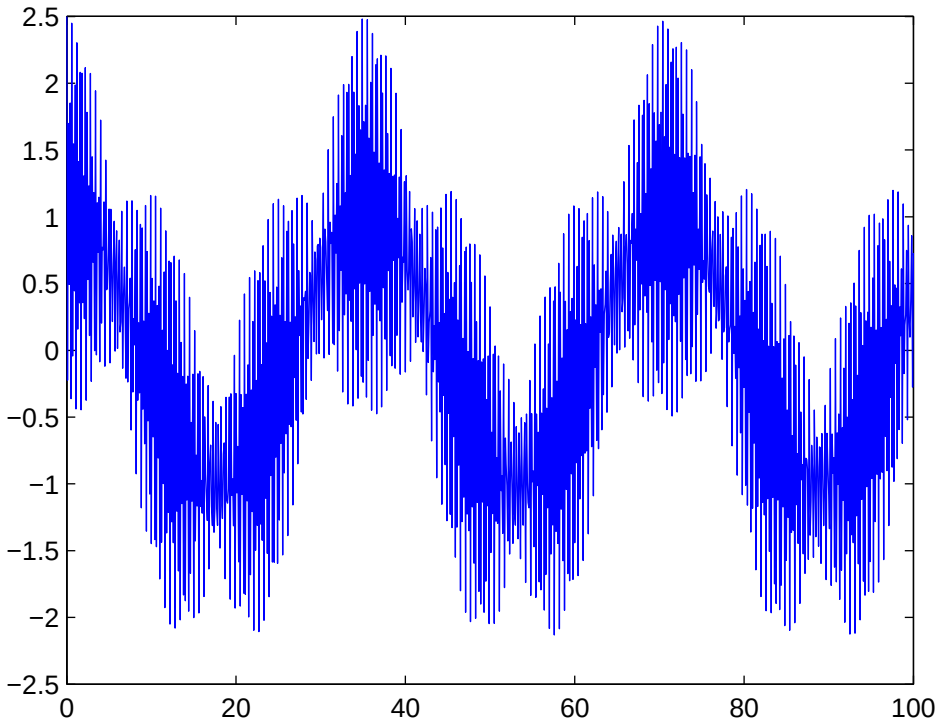

Figura 2 – Função da Equação 2.4

2.2 Conceitos Musicais

A música em si, além de ter em sua essência todas as leis físicas da ondulatória sonora, ela é uma forma de arte no que se refere a apresentação estética e do belo (WÖLFFLIN; JÚNIOR, 2000). Para a construção do belo em formas de som, há desenvolvido durante toda história da humanidade um conjunto de técnicas e metodologias bem apuradas. Nesse aspecto, a música define-se como ciência e pode ser abordada nas áreas de teoria básica, solfejo, ritmo, percepção melódica, dinâmica, harmonia, contraponto, formas musicais, instrumentos musicais, instrumentação, orquestração, arranjo, fisiologia da voz, fonética, psicologia da música, pedagogia musical, história da música, acústica musical, análise musical, composição e regência (MED, 1996).

A estrutura da arte musical em si é baseada na combinação de sons em forma simultânea e sucessiva recorrendo a ordem, equilíbrio e proporção dentro do tempo. Os principais elementos formadores da música podem ser divididos nessas categorias:

- melodia - sons dispostos em ordem sucessiva ao longo do tempo (concepção horizontal da música);

- harmonia - sons dispostos em ordem simultânea ao longo do tempo (concepção vertical da música);
- contraponto - conjunto de melodias e harmonias (concepção híbrida vertical e horizontal da música);
- ritmo - ordem e proporção em que estão dispostos as melodias e as harmonias.

O sons que formam as melodias e as harmonias possuem características principais como:

- altura - frequência das vibrações sonoras. Quanto maior frequência mais agudo o som será;
- duração - tempo de extenção do som ao longo do tempo;
- intensidade - amplitudade ou força das vibrações sonoras, conhecido como volume;
- timbre - combinação das intensidades dos harmônicos que um determinado agente sonoro.

A altura e intensidade do som são as características essenciais para a formulação dos conceitos de notas e acordes. Em altura entende-se como a divisão das frequências em 7 notas musicas - *Dó*, *Ré*, *Mi*, *Fá*, *Sol*, *Lá* e *Si*. Também essas mesmas notas possuem uma correspondente em sequência de letras alfabéticas introduzidas pelo Papa Gregório Grande - *C*, *D*, *E*, *F*, *G*, *A* e *B*. Normalmente essa sequência de letras são usadas para denominar acordes. Entretanto tais divisões de notas não são a menor divisão para o sistema temperado (MED, 1996). A menor divisão de notas se denomina semitom e são configurados pelos acidentes sustenidos (#) ou bemois (♭). Considerando essa divisão semitonal o sistema fica representado nessa sequência de 12 notas musicais (entre uma divisão e outra há a presença de um semitom): *Dó*, *Dó#* ou *Ré♭*, *Ré*, *Ré#* ou *Mi♭*, *Mi*, *Fá*, *Fá#* ou *Sol♭*, *Sol*, *Sol#* ou *Lá♭*, *Lá*, *Lá#* ou *Si♭* e *Si*. Ou seguindo a denominação inglesa: *C*, *C#* ou *D♭*, *D*, *D#* ou *E♭*, *E*, *F*, *F#* ou *G♭*, *G*, *G#* ou *A♭*, *A*, *A#* ou *B♭* e *B*.

Na divisão distributiva das faixas de frequências pelas notas segue um quadro básico (LABGARAGEM, 2014):

Octave → Note ↓	0	1	2	3	4	5	6	7	8	9	10
C	16.352 (−48)	32.703 (−36)	65.406 (−24)	130.81 (−12)	261.63 (±0)	523.25 (+12)	1046.5 (+24)	2093.0 (+36)	4186.0 (+48)	8372.0 (+60)	16744.0 (+72)
C♯/D♭	17.324 (−47)	34.648 (−35)	69.296 (−23)	138.59 (−11)	277.18 (+1)	554.37 (+13)	1108.7 (+25)	2217.5 (+37)	4434.9 (+49)	8869.8 (+61)	17739.7 (+73)
D	18.354 (−46)	36.708 (−34)	73.416 (−22)	146.83 (−10)	293.66 (+2)	587.33 (+14)	1174.7 (+26)	2349.3 (+38)	4698.6 (+50)	9397.3 (+62)	18794.5 (+74)
E♭/D♯	19.445 (−45)	38.891 (−33)	77.782 (−21)	155.56 (−9)	311.13 (+3)	622.25 (+15)	1244.5 (+27)	2489.0 (+39)	4978.0 (+51)	9956.1 (+63)	19912.1 (+75)
E	20.602 (−44)	41.203 (−32)	82.407 (−20)	164.81 (−8)	329.63 (+4)	659.26 (+16)	1318.5 (+28)	2637.0 (+40)	5274.0 (+52)	10548.1 (+64)	21096.2 (+76)
F	21.827 (−43)	43.654 (−31)	87.307 (−19)	174.61 (−7)	349.23 (+5)	698.46 (+17)	1396.9 (+29)	2793.8 (+41)	5587.7 (+53)	11175.3 (+65)	22350.6 (+77)
F♯/G♭	23.125 (−42)	46.249 (−30)	92.499 (−18)	185.00 (−6)	369.99 (+6)	739.99 (+18)	1480.0 (+30)	2960.0 (+42)	5919.9 (+54)	11839.8 (+66)	23679.6 (+78)
G	24.500 (−41)	48.999 (−29)	97.999 (−17)	196.00 (−5)	392.00 (+7)	783.99 (+19)	1568.0 (+31)	3136.0 (+43)	6271.9 (+55)	12543.9 (+67)	25087.7 (+79)
A♭/G♯	25.957 (−40)	51.913 (−28)	103.83 (−16)	207.65 (−4)	415.30 (+8)	830.61 (+20)	1661.2 (+32)	3322.4 (+44)	6644.9 (+56)	13289.8 (+68)	26579.5 (+80)
A	27.500 (−39)	55.000 (−27)	110.00 (−15)	220.00 (−3)	440.00 (+9)	880.00 (+21)	1760.0 (+33)	3520.0 (+45)	7040.0 (+57)	14080.0 (+69)	28160.0 (+81)
B♭/A♯	29.135 (−38)	58.270 (−26)	116.54 (−14)	233.08 (−2)	466.16 (+10)	932.33 (+22)	1864.7 (+34)	3729.3 (+46)	7458.6 (+58)	14917.2 (+70)	29834.5 (+82)
B	30.868 (−37)	61.735 (−25)	123.47 (−13)	246.94 (−1)	493.88 (+11)	987.77 (+23)	1975.5 (+35)	3951.1 (+47)	7902.1 (+59)	15804.3 (+71)	31608.5 (+83)

Figura 3 – Distribuição das frequências nas notas musicais em Hz

Acordes são harmonias formadas por pelo menos uma tríade (três notas)[1] tocadas simultâneamente e são definidos por certas quantidades de semitons entre as notas (MED, 1996). Essa quantidade se denomina intervalo musical.

As 3 notas da tríade são referenciadas como tônica (a nota base do acorde), terça e quinta (MED, 1996). Para acordes maiores (M) a distância entre a tônica e a terça é de 4 semitons (terça maior) e entre a tônica e a quinta é de 7 semitons (quinta justa). Para acordes menores (m) a distância entre a tônica e a terça é de 3 semitons (terça menor) e entre a tônica e a quinta é de 7 semitons (quinta justa). Para acordes aumentados (aum) a distância entre a tônica e a terça é de 4 semitons (terça maior) e entre a tônica e a quinta é de 8 semitons (quinta aumentada). Para acordes diminutos (dim) a distância entre a tônica e a terça é de 3 semitons (terça menor) e entre a tônica e a quinta é de 6 semitons (quinta diminuta).

Exemplos de acordes maiores são:

- dó maior - CM (tríade $Dó$, Mi e Sol);
- lá maior - AM (tríade $Lá$, $Dó\#$ e Mi).

Exemplos de acordes menores são:

- mi menor - Em (tríade Mi, Sol e Si,);
- ré sustenido menor - $D\#m$ (tríade $Ré\#$, $Fá\#$ e $Lá\#$).

Exemplos de acordes aumentados são:

[1] Existem acordes mais complexos com mais de 3 notas porém o escopo desse trabalho só se delimita a tríades.

- sol aumentado - *Gaum* (tríade *Sol*, *Si* e *Ré#*);
- si aumentado - *Baum* (tríade *Si*, *Ré#* e *Sol*).

Exemplos de acordes diminutos são:

- dó sustenido diminuto - *C#dim* (tríade *Dó#*, *Mi* e *Sol*);
- lá sustenido diminuto - *A#dim* (tríade *Lá#*, *Dó#* e *Mi*).

Na teoria dos acordes também há a presença do conceito de inversões. Inverter um acorde consiste em trocar de posição para uma oitava a cima a nota inferior, trocar a nota mais baixa do acorde por uma outra da mesma denominação só que com o dobro de frequência acima. Em tríades há a presença do acorde em seu estado fundamental, a primeira inversão (a terça fica sendo como a nota mais grave) e a segunda inversão (a quinta fica sendo como a nota mais grave).

Segue exemplos de acordes em estado fundamental:

- dó maior - *CM* (tríade *Dó*, *Mi* e *Sol*);
- mi menor - *Em* (tríade *Mi*, *Sol* e *Si*,);
- sol aumentado - *Gaum* (tríade *Sol*, *Si* e *Ré#*);
- lá sustenido diminuto - *A#dim* (tríade *Lá#*, *Dó#* e *Mi*).

Segue exemplos de acordes em primeira inversão:

- dó maior - *CM* (tríade *Mi*, *Sol* e *Dó*);
- mi menor - *Em* (tríade *Sol*, *Si* e *Mi*);
- sol aumentado - *Gaum* (tríade *Si*, *Ré#* e *Sol*);
- lá sustenido diminuto - *A#dim* (tríade *Dó#*, *Mi* e *Lá#*).

Segue exemplos de acordes em segunda inversão:

- dó maior - *CM* (tríade *Sol*, *Dó* e *Mi*);
- mi menor - *Em* (tríade *Si*, *Mi* e *Sol*);
- sol aumentado - *Gaum* (tríade *Ré#*, *Sol* e *Si*);
- lá sustenido diminuto - *A#dim* (tríade *Mi*, *Lá#* e *Dó#*).

2.3 Conceitos de Processamento de Sinais

O conceito de processamento de sinais está inteiramente ligado à natureza do sinal e a aplicação que normalmente se dá é de modificação ou análise. Sinal pode ser entendido como um objeto matemático, normalmente uma função matemática, que descreve o comportamento de um determinado fenômeno da natureza podendo ser, entre outros, físico, químico, biológico e financeiro (OPPENHEIM; WILLSKY; NAWAB, 1983).

Da natureza do sinal abordado, é explícito de que o mesmo é de cunho físico - sinais sonoros. Com a possibilidade de se trabalhar com sinal sonoro, há em processamento de sinais a liberdade de modificar ou extrair informações relevatantes para uma dada aplicação. Nesse contexto o sinal será processado com o intuito de colher informações para serem analisadas de forma a deduzir comportamentos de ondas sonoras.

Os sinais geralmente são funções relacionadas ao tempo. Eles podem ser processados em tempo contínuo (analogicamente) ou em tempo discreto (digitalmente). O escopo desse trabalho está restringido somente ao processamento na forma digital.

Para haver processamento digital é preciso que o sinal seja descrito computacionalmente num hardware. A forma de captação de um fenômeno contínuo para o ambiente computacional se denomina processo de amostragem e quantização (DRUYVESTEYN, 1992).

Amostrar um sinal significa recolher um número determinado de amostras dado um período de tempo, ou seja, haverá uma frequência (taxa de amostragem) **F** associada a um período de tempo **T** que proporcionará um conjundo finito de amostras num intervalo temporal. A relação exposta dar-se-á por:

$$\mathbf{F_s} = \frac{1}{T} \tag{2.5}$$

Segundo o teorema de amostragem Nyquist–Shannon (UNSER, 2000), para sons musicais o mais adequado é que a taxa de amostragem seja o dobro da frequência máxima de audição do ouvido humano - 22.050 Hz, ou seja, a frequência será de 44.100 Hz. Essa grandeza significa que serão captadas 44.100 amostras de áudio a cada segundo.

Quantizar um sinal significa alocar valores digitais para os valores analógicos do eixo da ordenada, que são normalmente valores de tenões elétricas. Essa alocação está ligada diretamente às características do conversor analógico/digital. Nesse caso específico foi usado um conversor de 16 bits para a quantização do sinal. Tal fato permite a presença de 65.536 (2 elevado a 16) valores para representar as subidas e descidas da onda sonora.

O conceito de energia está totalmente ligado a muitas outras áreas e é de extrema importância por ser essencial nos fenômenos naturais (OPPENHEIM; WILLSKY; NAWAB, 1983). O aspecto energético adotado nessa presente solução será em tempo discreto que

é definido como:

$$\mathbf{E} = \sum_{t=t1}^{t2} |x[n]|^2 \tag{2.6}$$

Outro conceito relacionado que é de extrema importância é a lei de conservação de energia representada pelo teorema de Parseval. Esse teorema mostra que a energia do sinal sempre se conserva independentemente da projeção que o sinal foi submetido. Mais especificamente na transformada de fourier discreta o teorema é descrito como:

$$\sum_{n=0}^{N-1} |x[n]|^2 = \frac{1}{N}. \sum_{k=0}^{N-1} |X[k]|^2 \tag{2.7}$$

Tal qual N é o número total de amostras e $X(k)$ a transformada discreta de fourier.

A transformada de fourier é uma ferramenta muito importante para a realização desse trabalho. Ela permite projetar o sinal em funções de base senoidais, ou seja, é possível ver através dela quais componentes frequenciais de senóides estão presentes no sinal e qual é a energia das mesmas.

A representação da transformada de fourier em frequência discreta (DFT) é dada por:

$$X[k] = \sum_{n=0}^{N-1} x[n].exp^{-j2\pi.\frac{k}{N}.n} \tag{2.8}$$

A representação da transformada inversa de fourier em frequência discreta é dada por:

$$x[n] = \frac{1}{N}. \sum_{k=0}^{N-1} X[k].exp^{j2\pi.\frac{k}{N}.n} \tag{2.9}$$

2.4 Conceitos de Redes Neurais Artificiais

Identificar padrões num sinal nem sempre é um trabalho trivial ou até mesmo determinístico. Normalmente sinais possuem ruídos intratáveis, sua composição é complexa no sentido de haver muitas amostras para análise e, como os sinais são fenômenos naturais, facilmente são vistos como sistemas complexos (MORIN; MATOS, 2007). Determinar uma equação ou um algorítmo fixo e determinístico para classificação e processamento de sinais é bastante limitado e aderente há vários erros.

Além disso, para o reconhecimento de harmonias é preciso usar conceitos de teoria musical no que tange aos acordes e notas para o reconhecimento de padrões presentes no sinal. É preciso então ter alguma forma de representar esse conhecimento musical no ponto de vista computacional.

Diante desse ambiente de incertezas e requisitos de incorporação do conhecimento musical no campo computacional, uma solução que é aderente ao contexto é o uso de redes neurais artificiais. Essa técnica, além de prover as características necessárias para

deixar a solução estável, ela modela o funcionamento neural de organismos vivos. Esse fato é muito interessante visto que surge a possibilidade de usar os mesmos mecanismos (de forma análoga) de reconhecimento de padrões sonoros do cérebro humano num sistema computacional.

Dado um especialista que possui o reconhecimento de padrões dos acordes, basta somente consolidar uma arquitetura de rede neural para receber esse conhecimento empírico de modo que o seu uso seja eficiente para classificação.

Entende-se por rede neural como "um processador maciçamente paralelamente distribuído constituído de unidades de processamento simples, que têm a propensão para armazenar conhecimento experimental e torná-lo disponível para o uso" (S; HAYKIN, 2009).

As redes neurais possuem uma característica essencial no que tange o aprendizado empírico. Sua estrutura oferece suporte para que conhecimentos adquiridos de maneira experimental (via ser humano muitas vezes) possam ser aprendidos e usados. O processo de aprendizagem da rede se chama algorítmo de aprendizado e o mesmo pode ser feito de várias formas como lei de Hebb, algorítmo de *backpropagation*, estratégias de competição e máquina de Boltzmann. Além disso é envolvido nesse processo paradígmas de aprendizado que é como o ambiente vai atuar sobre a rede neural para que ela possa aprender. Exemplos de paradígmas de aprendizado são aprendizado supervisionado, aprendizado por reforço e aprendizado não-supervisionado (ou auto-organizado).

Outra característica essencial de uma rede neural é a representação do conhecimento. Essa característica é referente às informações armazenadas ou a modelos utilizados por uma pessoa ou máquina com o intuito de interpretar, prever e responder de forma coerente ao mundo exterior (S; HAYKIN, 2009). Para tal representação deve-se levantar em conta quais informações serão abstraídas e tornadas explícitas e como a informação será codificada no sistema. Com o intuito de atingir os objetivos de uma boa representação do conhecimento na rede neural há um conjunto de regras sugeridas a se seguir (S; HAYKIN, 2009):

- regra 1 - entradas similares normalmente devem produzir representações similares no interior da rede e devem ser classificadas como pertencentes a mesma categoria;
- regra 2 - itens de classes diferentes devem ser representados de formas diferentes;
- regra 3 - se uma característica é importante deve-se haver um grande número de neurônios envolvidos na representação daquele item de rede;
- regra 4 - informações prévias e invariâncias devem ser incorporadas a rede para que o sistema fique simples e sem trabalho para aprender as mesmas.

Por fim outra característica importante de uma rede neural é a capacidade de generalização. Isso permite com que entradas desconhecidas possam ser classificadas e tratadas de forma coesa e coerente, fazendo com que circunstâncias críticas e imprevisíveis possam ser contornadas sem grandes prejuízos.

A unidade mínima de processamento de uma rede neural é o neurônio artificial. Segue uma representação de um modelo (S; HAYKIN, 2009):

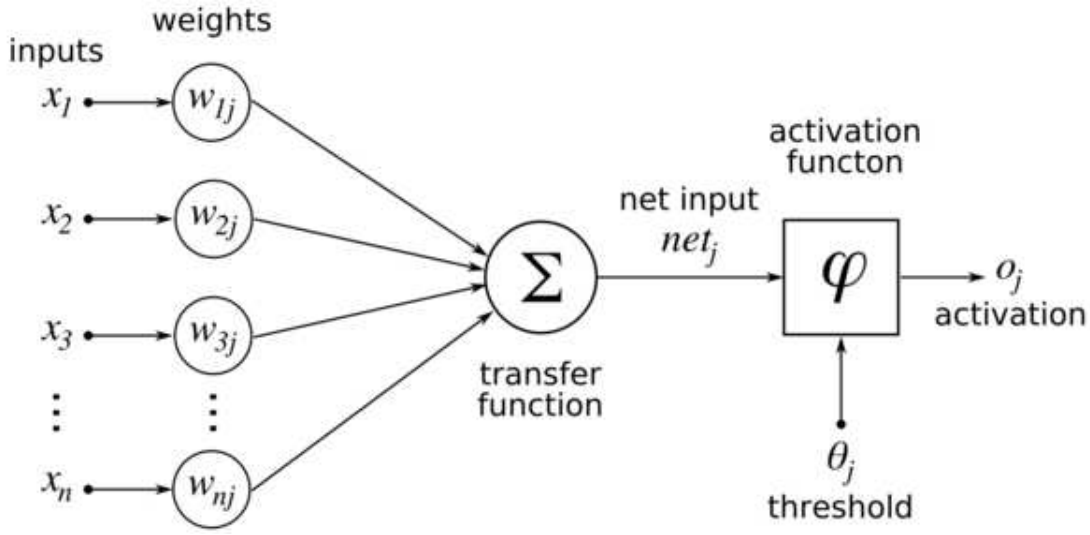

Figura 4 – Modelo de um neurônio

Como é representado na figura os conjuntos de w representam pesos sinápticos para a modulação dos sinais de entrada. Após há um somador para efetuar operações de combinações lineares. Por último há uma função de ativação, mais conhecido como um limiar de ativação para que a resposta possa ser propagada a outros neurônios.

Para o presente problema, foi sugerido o uso da rede neural do tipo PNN (Probabilistic Neural Network). Ela é inerentemente um sistema de classificação bastante simples de aprendizado não-supervisionado, ou seja, novos conhecimentos são adquiridos pela simples inserção de novos neurônios. Segue o modelo arquitetural (JüRGEN, 2014):

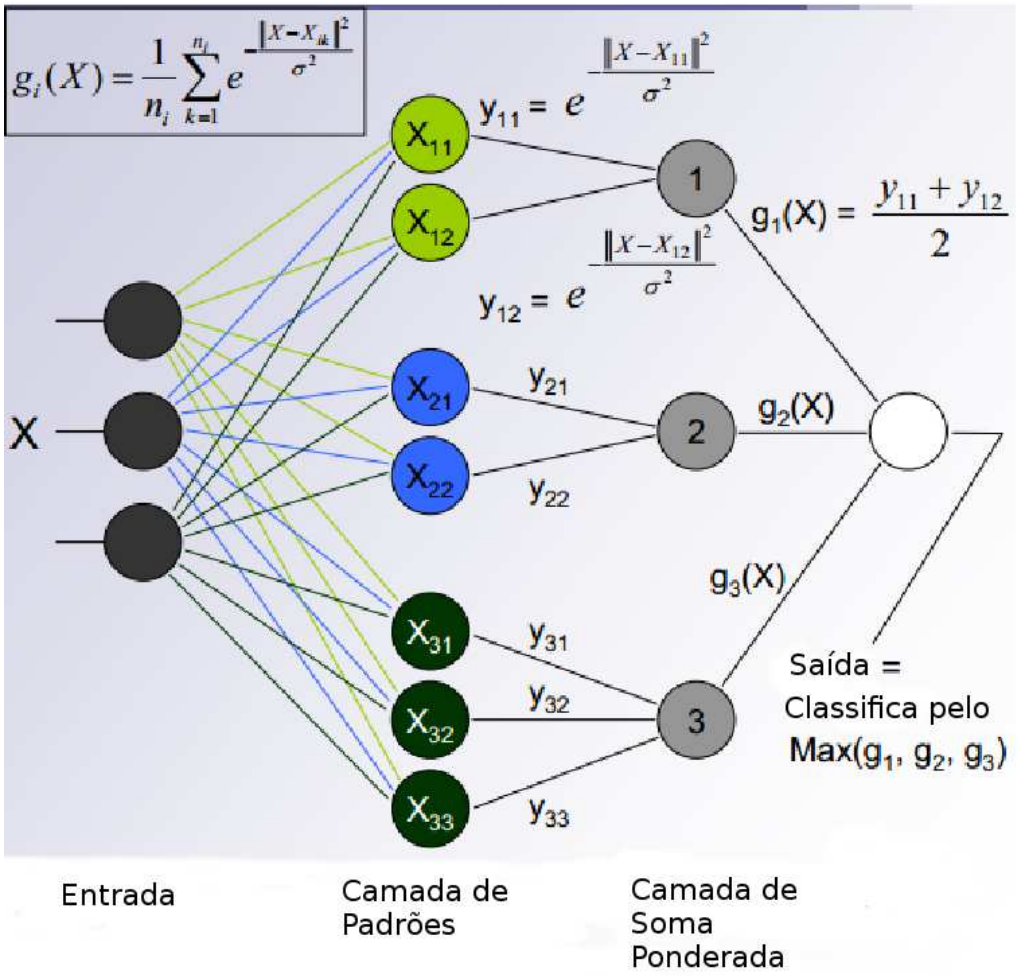

Figura 5 – Modelo arquitetural da PNN

Ela é uma rede de 3 camadas: a primeira é responsável por classificar a probabilidade de um indivíduo ser de uma determinada classe através da distância euclidiana combinada com uma curva gaussiana; a segunda é responsável por somar e fazer uma média simples dessas probabilidades de cada classe; a terceira é responsável por extrair a classe de maior probabilidade, ou seja, pegar o valor máximo dado o conjunto de classes **g**. O valor final de saída é a classificação do sinal de entrada.

3 Desenvolvimento da Solução

3.1 Metodologia

Esse trabalho está dentro de um contexto específico na engenharia de software. Esse contexto se caracteriza num projeto de pesquisa e desenvolvimento no qual até a consolidação da solução pouco se sabia sobre sua natureza (por exemplo arquitetura e viabilidade), o andamento do desenvolvimento ocorreu de forma não-linear e complexa e houve um frequente diálogo entre disciplinas de diferentes domínios - transdiciplinaridade.

No que diz respeito ao desconhecimento da natureza da solução no início do projeto, houve várias questões teóricas de pesquisa que foram passíveis de experimentações e disscusões ao longo do processo de desenvolvimento. Tal fato ocasionou na caracterização da solução ao longo do projeto e sua definição total somente no final.

No que tange a não-linearidade, variou-se muito o desempenho na concretização de soluções. A taxa de produção intelectual comportou-se como não repetível ao longo do tempo. E a complexidade, de acordo com a teoria de Edgar Morin (MORIN; MATOS, 2007), impactou o projeto de forma que a soma dos métodos e tecnologias gerasse resultados imprevisíveis.

O termo transdisciplinaridade é empregado para um patamar de relações entre disciplinas e domínios de conhecimento. Conceitua-se como tal um processo que transcende as disciplinas, que está entre, além e através das disciplinas (RIBEIRO, 2014). Nesse trabalho foram investigados vários domínios de conhecimento como a engenharia, matemática, neurociências, psicologia, música e computação.

Dado o contexto e características do trabalho foi estabelecido um método de desenvolvimento empírico, iterativo e incremental.

Na engenharia de software se destacam dois processos de controle de desenvolvimento: precesso definido e processo empírico. O processo definido é constiuído de um conjunto de sub-processos rigoros nos quais possuem entradas e saídas bem definidas e repetitivas (WEISS et al., 1999). Já o processo empírico é constituído de um conjunto de sub-processos imperfeitamente definidos nos quais as entradas e saídas são imprevisíveis e não repetíveis, características essas presentes no desenvolvimento desse trabalho.

A metodologia empírica de desenvolvimento de software se embasa três fundamentos: precisa ser transparente, visto que o máximo de variáveis devem estar visíveis para os envolvidos no projeto; dado as variáveis expostas a metodologia precisa ser frequentemente inspecionada; feito as inspeções objetivo final é adaptar de acordo com as necessidades.

Esses três fundamentos visam ajustar o processo de desenvolvimento para evitar variações de produção inaceitáveis e maximizar a mesma (DYBÅ; DINGSØYR, 2008).

Para que os procedimentos da metodologia empírica possa ocorrer ela precisa ser de natureza iterativa e incremental. Iterativa e incremental pois terá ciclos curtos de desenvolvimento e a cada ciclo terá incrementos de código. No final de cada ciclo ter-se-à como resultado parâmetros de feedback para a melhoria contínua.

Outra análise do método empírico é o embasamento no ciclo de melhoria de processos e produtos *Plan-Do-Check-Act* (PDCA) (SHEWHART, 1980). Ele é constuído em 4 fases: *Plan* - planejamento do desenvolvimento; *Do* - executar o que foi planejado; *Check* - avaliar o que foi feito; *Act* - propor melhorias para os próximos ciclos.

Por fim, para consolidação da metodologia abordada nesse trabalho, foi utilizado o modelo *Goal-Question-Metric* (GQM) para a estruturação dos ciclos (CALDIERA; ROMBACH, 1994). Esse modelo foi utilizado para orientar o desenvolvimento e ele é composto por 3 etapas sucessivas: *Goal* - objetivo a ser alcançado; *Question* - questões chaves para que o objetivo possa ser alcançado; *Metric* - métricas que vão validar se as questões foram respondidas e objetivo alcançado. Nesse contexto os objetivos do trabalho são os objetivos específicos, as questões são as hipotéses levantadas no início de cada ciclo e as métricas são os resultados finais de cada ciclo.

Com o intuito de atingir os objetivos, serão abordados questões, hipóteses e critérios relacionados às problemáticas abordadas. No que diz respeito as questões, segue a formulação das mesmas:

- Se cada nota é uma frequência de vibração sonora, como analisar o sinal no ponto de vista de frequências? Dessa questão, surge a seguinte hipótese:
  - A transformada de fourier pode construir o espectro de frequências do sinal. O critério para avaliar essa hipótese é:
    - Vetor com os níveis de energia relacionados a cada frequência.
- Como configurar essas informações para localizar as notas musicais?
  - Dado que cada nota musical é um conjunto de frequências, realocar as energias frequenciais da transformada de fourier afim de que cada posição do vetor seje 1 unidedade de frequência (Hz) pode mapear a energia de cada nota. Os critérios para avaliar essa hipótese são:
    - Vetor com os níveis de energia relacionados a cada frequência deve ter o tamanho de 22050 posições.
    - Cada posição do vetor de energia relacionados a cada frequência possui valor de 1 Hz.

- Como adicionar as próximas camadas da rede para determinação dos acordes?
  - Visto que associar as frequências as notas musicas é uma tarefa muito complexa para uma solução determinística, uma rede neural de aprendizado não supervisionado do tipo *ProbabilisticNeuralNetwork* (PNN) pode classificar um conjunto de frequências em sua respectiva nota musical. O critério para avaliar essa hipótese é:
    - Vetor com os níveis de energia relacionados a cada nota.
- Como reconhecer acordes no tempo de tal forma a saber onde eles ocorrem?
  - A transformada de fourier pode construir o espectro de frequências do sinal. O critério para avaliar essa hipótese é:
    - Vetor com os níveis de energia relacionados a cada frequência.
- Como ler o sinal todo e ter a visilibilidade em tempo e frequência?
  - A transformada de fourier pode construir o espectro de frequências do sinal. O critério para avaliar essa hipótese é:
    - Vetor com os níveis de energia relacionados a cada frequência.
- Como ler o sinal todo e ter a visilibilidade em tempo e frequência?
  - A transformada de fourier pode construir o espectro de frequências do sinal. O critério para avaliar essa hipótese é:
    - Vetor com os níveis de energia relacionados a cada frequência.
- Como extrair o tom da música?
  - A transformada de fourier pode construir o espectro de frequências do sinal. O critério para avaliar essa hipótese é:
    - Vetor com os níveis de energia relacionados a cada frequência.
- Como trabalhar com a frequência de aparecimento das notas?
  - A transformada de fourier pode construir o espectro de frequências do sinal. O critério para avaliar essa hipótese é:
    - Vetor com os níveis de energia relacionados a cada frequência.
- Em relação aos acordes transitórios, como corrigir?
  - A transformada de fourier pode construir o espectro de frequências do sinal. O critério para avaliar essa hipótese é:
    - Vetor com os níveis de energia relacionados a cada frequência.

- Como extrair a nota mais grave (baixo) de cada período do tempo?
  - A transformada de fourier pode construir o espectro de frequências do sinal. O critério para avaliar essa hipótese é:
    * Vetor com os níveis de energia relacionados a cada frequência.
- Como extrair a nota mais grave (baixo) de cada período do tempo e incluir os acordes aumentados e invertidos?
  - A transformada de fourier pode construir o espectro de frequências do sinal. O critério para avaliar essa hipótese é:
    * Vetor com os níveis de energia relacionados a cada frequência.

3.2 Técnicas Utilizadas para Desenvolvimento do Sistema-Solução

Nesta secção será descrito o desenvolvimento da solução e as técnicas utilizadas. A explicação da solução se embasará no seguinte diagrama de fluxo de dados:

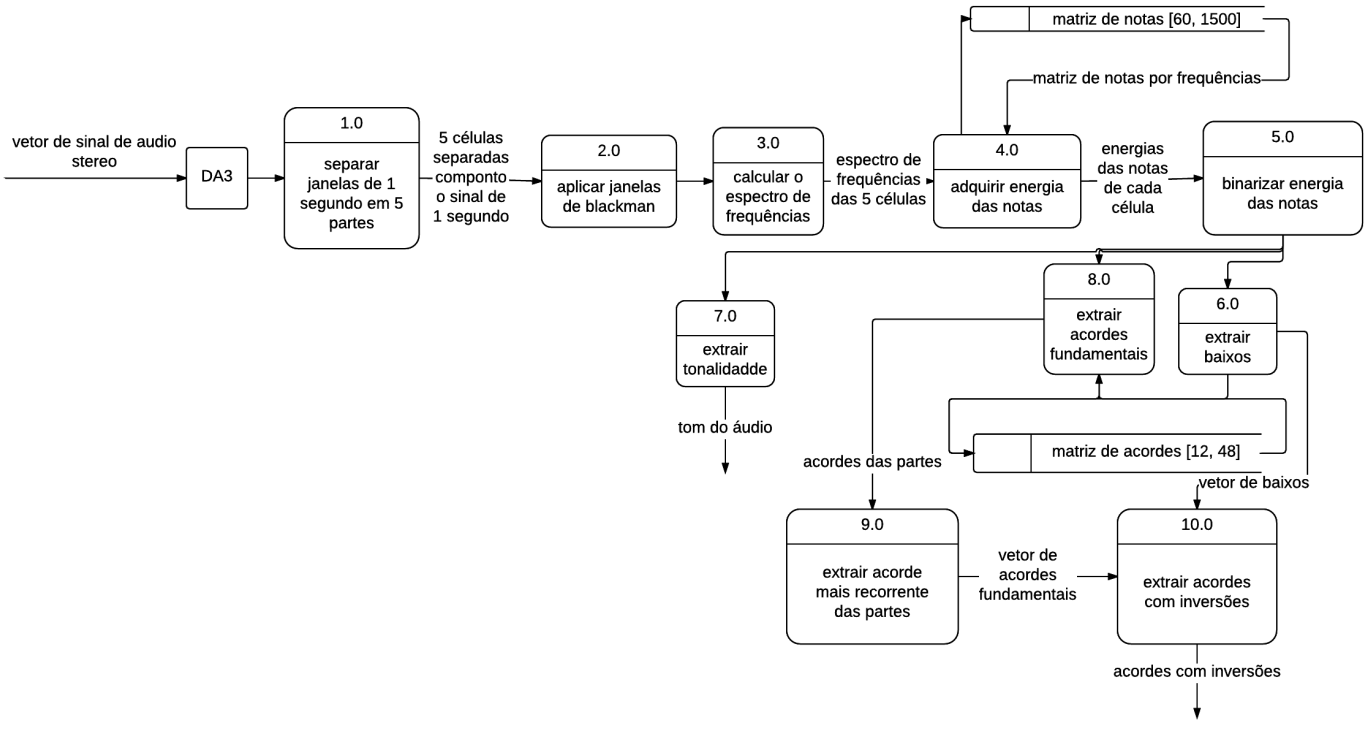

Figura 6 – Diagrama de Fluxo de Dados

A solução começa com a chamada da função DA3. Ela recebe como parâmetro um vetor de audio *stereo*, ou seja, ela carrega uma matriz 2 por N, tal que N é o tamanho do sinal de áudio (número de amostras). Esse sinal de áudio é retornado através da função *wavread*(<*caminho do arquivo*>). O tipo de arquivo lido é do formato-padrão de

áudio .wav. Esse formato de arquivo permite um armazenamento dos dados em blocos em modulação de pulsos PCM (*pulse-code-modulation*). O PCM armazena em arquivo de áudio não-comprimido (sem perdas), ou seja, o processo de amostragem e quantização representa exatamente o que foi descrito na parte de fundamentos teóricos desse trabalho (taxa de amostragem de 44.100 Hz e quantização de 16 bits). Ao final do fluxo há 2 saídas: os acordes ao longo do tempo e o tom do áudio.Nos próximos tópicos serão explicados os comportanmentos de cada uma das caixas de processamento do diagrama de fluxo de dados.

3.2.1 Procedimento 1: Separar Janelas de 1 Segundo em 5 Partes

Após o sinal ser carregado num vetor de audio *stereo*, ele deverá ser transformado num do tipo mono. Sinal mono de áudio é aquele com somente um canal. Isso é necessário para que o processamento não fosse redundante. Não agregaria valor nesse caso processar um sinal de duplo canal sendo que a fonte emissora de ondas sonoras é comum para ambos. Após essa conversão o sinal é repartidos em 5 partes de tamanhos iguais a 1 segundo, porém deslocadas a 0.2 segundos de cada um. Esse processo é para a segmentar áudio no intuito de achar o acorde mais provável num intervalo de tempo de um segundo, fazendo com que acordes de transição ou ruidosos sejam suprimidos. Esse procedimento pode ser conferido a seguir:

```
1  % get total seconds of time to mensure the length of music
2  signal = signal(:,1);
3  time_seconds_total = fix((length(signal)/fs));
4
5  % preparing struct to allocate notes in time
6  set_of_notes_time = {};
7  for set = 1:5
8      notes_time(time_seconds_total, 60) = 0;
9      set_of_notes_time{set} = notes_time;
10 end
```

Basicamente a variável de entrada dessa função é reescrita como uma matriz 1 por N, tal qual N é o número de amostras do sinal. No laço seguinte cria-se um conjunto de 5 células, cada uma comportando uma parte do sinal.

3.2.2 Procedimento 2: Aplicar Janelas de Blackman

Dado que o contexto da solução se deu por $Short\text{-}Time\text{-}Fourier\text{-}Transform$, é preciso minimizar as distorções oriundas dos janelamentos. Para tal foi proposto a multiplicação de cada parte das janelas ao longo do tempo por uma janela de blackman, uma do tipo gaussiana. Segue procedimento em código abaixo:

```
function set_of_windows_signals = build_window_short_fft(signal, time, fs)
    signal = [signal(:)];

    % part A
    time_start_A = round(1+((time-1)*fs));
    time_end_A = round(time*fs);
    signal_time_A = signal(time_start_A:time_end_A);
    signal_time_A = blackman(length(signal_time_A)).*signal_time_A;

    % part B (displacement = + 0.2 seconds)
    time_start_B = round(1+((time-1)*fs+0.2*fs));
    time_end_B = round((time+0.2)*fs);
    if time_start_B < length(signal) && time_end_B <= length(signal)
        signal_time_B = signal(time_start_B:time_end_B);
        signal_time_B = blackman(length(signal_time_B)).*signal_time_B;
    else
        signal_time_B(length(signal)) = 0;
    end

    % part C (displacement = + 0.4 seconds)
    time_start_C = round(1+((time-1)*fs+0.4*fs));
    time_end_C = round((time+0.4)*fs);
    if time_start_C < length(signal) && time_end_C <= length(signal)
        signal_time_C = signal(time_start_C:time_end_C);
        signal_time_C = blackman(length(signal_time_C)).*signal_time_C;
    else
        signal_time_C(length(signal)) = 0;
    end

    % part D (displacement = + 0.6 seconds)
    time_start_D = round(1+((time-1)*fs+0.6*fs));
    time_end_D = round((time+0.6)*fs);
    if time_start_D < length(signal) && time_end_D <= length(signal)
        signal_time_D = signal(time_start_D:time_end_D);
        signal_time_D = blackman(length(signal_time_D)).*signal_time_D;
    else
        signal_time_D(length(signal)) = 0;
    end

    % part E (displacement = + 0.8 seconds)
    time_start_E = round(1+((time-1)*fs+0.8*fs));
    time_end_E = round((time+0.8)*fs);
    if time_start_E < length(signal) && time_end_E <= length(signal)
        signal_time_E = signal(time_start_E:time_end_E);
        signal_time_E = blackman(length(signal_time_E)).*signal_time_E;
    else
        signal_time_E(length(signal)) = 0;
```

```
48      end
49
50      set_of_windows_signals = {};
51      if length(signal_time_A) == length(signal_time_B) && ...
52          length(signal_time_A) == length(signal_time_C) && ...
53           length(signal_time_A) == length(signal_time_D) &&  ...
54            length(signal_time_A) == length(signal_time_E)
55          set_of_windows_signals{1} = signal_time_A;
56          set_of_windows_signals{2} = signal_time_B;
57          set_of_windows_signals{3} = signal_time_C;
58          set_of_windows_signals{4} = signal_time_D;
59          set_of_windows_signals{5} = signal_time_E;
60      else
61          set_of_windows_signals{1} = signal_time_A;
62          set_of_windows_signals{2} = signal_time_A;
63          set_of_windows_signals{3} = signal_time_A;
64          set_of_windows_signals{4} = signal_time_A;
65          set_of_windows_signals{5} = signal_time_A;
66      end
```

3.2.3 Procedimento 3: Calcular o Espectro de Frequências

O passo seguinte é adquirir os espectros de frequências oriundos do cálculo da transformada discreta de fourier. O cálculo será feito para cada uma das 5 partes de janelas. Segue procedimentos para tal:

```
1       % get frequency spectrum
2   function set_of_spectrums = get_frequency_spectrum( ...
3   set_of_windows_signals, sampling)
4
5       % allocate struct to spectrum
6       set_of_spectrums = {};
7       sampling = sampling/21;
8
9       for part_signal_iterator = 1:5
10          % make downsample to put frequency max in 1050 Hz
11          signal = downsample(set_of_windows_signals{ ...
12  part_signal_iterator}, 21);
13          % doing fourier transform
14          frequencies=(0:length(signal)-1)*sampling/length(signal);
15          module_fft = abs(fft(signal));
16          f_round = round(frequencies);
17          frequencies_energy(max(f_round)) = 0;
18          for slot = 2:length(f_round)
19              frequencies_energy(f_round(slot)) = module_fft(slot);
```

```
20          end
21          frequency_spectrum_part = frequencies_energy(1:fix(end/2));
22          set_of_spectrums{part_signal_iterator} = frequency_spectrum_part;
23      end
```

Primeiramente é alocado uma variável para comportar os 5 espectros de frequência, um para cada parte da janela. Depois os sinais passam por uma transformação de subamostragem na qual são eliminadas informações de altas frequências a partir de 1500 Hz. É feito o cálculo do módulo da transformada de frourier e esse mesmo vetor passar por uma reorganização de *slots* de tal forma que cada *slot* comporta-se 1 unidade de Hz.

3.2.4 Procedimento 4: Adquirir Energias das Notas

Nesse procedimento cada espectro de frequência é correlacionado com conjunto de notas musicais dado um conjunto de frequências que nelas estão presentes. Ao final do processo espera-se matrizes de conjuntos de notas para cada uma das 5 partes da janela. Segue os procedimentos em código feitos:

```
1  function set_of_notes_time = get_energy_notes(set_of_spectrums, ...
2  set_of_notes_time, time)
3
4      % load data notes
5      load_notes;
6
7      for set = 1:5
8          respfreq = set_of_spectrums{set};
9          notes_time = set_of_notes_time{set};
10
11         % this case works in one case
12         respfreq = [respfreq zeros(1, length(notes(1,:)) - ...
13 length(respfreq))];
14         for note = 1:60
15             notes_time(time, note) = sum((respfreq.*notes(note,:)).^2);
16         end
17
18         set_of_notes_time{set} = notes_time;
19     end
```

A primeira atividade é carregar a base de dados de notas musicais originando o retorno de um matriz 60 notas por 1500 frequências. Então cada conjunto de notas relacionados às partes de janela serão correlacionados a partir de uma operação de multiplicação e, por fim, é feita a soma dos quadrados dos termos. Ao final uma matriz de notas por tempo é construída para cada uma das 5 partes.

3.2.5 Procedimento 5: Binarizar Energia das Notas

Esse passo compreende o processo de limiarização das energias das notas em 0's ou 1's de tal forma que se possa detectas as notas tocadas (1) ou não (0). Ao final desse processo espera-se conjuntos de energias de notas musicais em somente dois valores - 1 ou 0. Segue o código para esse processo:

```
% binarize set of notes
    for set = 1:5
        notes_time = set_of_notes_time{set};

        for time = 1:time_seconds_total
            for note = 1:60
                if notes_time(time, note) < max(max(notes_time))/180
                    notes_time(time, note) = 0;
                else
                    notes_time(time, note) = 1;
                end
            end
        end

        set_of_notes_time{set} = notes_time;
    end
```

No começo do procedimento é destacado um laço para cada uma das partes das janelas. No meio do procedimento destaca-se com operação de realocação do valor 0 para valores de energia menores que 180% do valor máximo do conjunto de notas e 1 se for caso ao contrário. Por fim cada conjunto de notas são realocados em células.

3.2.6 Procedimento 6: Extrair Baixos

Com o intuito de determinar acordes com inversões e discernir os que são de natureza aumentada, acoplou-se no sistema um componente desenvolvido para a extração das notas mais graves numa dada janela de tempo. Segue o código desenvolvido para essa atividade:

```
function bass_time = get_bass(set_of_notes_time)

    notes_time_A = set_of_notes_time{1};
    notes_time_B = set_of_notes_time{2};
    notes_time_C = set_of_notes_time{3};
    notes_time_D = set_of_notes_time{4};
    notes_time_E = set_of_notes_time{5};

```

```
9       total_seconds = length(notes_time_A(:,1));
10      notes_time(total_seconds, 60) = 0;
11      for time = 1:total_seconds
12          for note = 1:60
13              notes_to_analyse = [notes_time_A(time, note) ...
14               notes_time_B(time, note) ...
15            notes_time_C(time, note) ...
16             notes_time_D(time, note) ...
17              notes_time_E(time, note)];
18              notes_time(time, note) = mode(notes_to_analyse);
19          end
20      end
21
22      bass_time(1:total_seconds) = 0;
23      for time = 1:total_seconds
24          maxs = find(notes_time(time,:)==max(notes_time(time,:)));
25          bass_time(time) = maxs(1);
26      end
27
28      for bass = 1:length(bass_time)
29          bass_time(bass) = mod(bass_time(bass) - 1, 12) + 1;
30      end
31
32  end
```

No procedimento verificamos que os primeiros passos são de atribuição de variáveis em relação as partes das janelas. Depois cada uma dessas partes serão analisadas quanto as notas mais recorrentes com a função **mode**. Dado essa análise os baixos são extraidos com a primeira ocorrência de 1, dado que as notas estão binarizadas.

3.2.7 Procedimento 7: Extrair Tonalidade

A extração de tonalidade é um módulo do sistema que possui como entrada o conjunto de notas binarizadas das partes de janela. A saída é o tom da música tocado baseando-se em acordes fundamentais maiores e menores. Segue a especificação do procedimento:

```
1  function [chord_pitch, chord_pitch_number] = ...
2  get_chord_pitch(notes_time, time_seconds_total, chords)
3
4      dictionary_chords = { 'C', 'Cm', 'Caum', 'Cdim', ...
5       'C#', 'C#m', 'C#aum', 'C#dim', 'D', 'Dm', 'Daum', 'Ddim', ...
6       'Eb', 'Ebm', 'Ebaum', 'Ebdim', 'E', 'Em', 'Eaum', 'Edim', ...
7       'F', 'Fm', 'Faum', 'Fdim', 'F#', 'F#m', 'F#aum', 'F#dim', ...
8       'G', 'Gm', 'Gaum', 'Gdim', 'G#', 'G#m', 'G#aum', 'G#dim', ...
```

```
 9       'A', 'Am', 'Aaum', 'Adim', 'Bb', 'Bbm', 'Bbaum', 'Bbdim', ...
10       'B', 'Bm', 'Baum', 'Bdim' };
11
12      notes_energy_total(60) = 0;
13      for note = 1:60
14          notes_energy_total(note) = sum([notes_time(:,note)]);
15      end
16
17      % discover tone music
18      notes_energy_tone(12) = 0;
19      for note = 1:12
20          notes_energy_tone(note) = notes_energy_total(note) + ...
21           notes_energy_total(note + 12) ...
22              + notes_energy_total(note + 2*12) + ...
23               notes_energy_total(note + 3*12) ...
24                  + notes_energy_total(note + 4*12);
25      end
26
27      % find chord tone
28      load_chords_tone;
29      chords_tone(48) = 0;
30      for chord = 1:48
31          chords_tone(chord) = sum((notes_energy_tone.* ...
32               chords_tone_mask(:, chord)'.^2));
33      end
34
35      chord_pitch_number = find(chords_tone==max(chords_tone));
36      chord_pitch = dictionary_chords{chord_pitch_number};
```

No início desse processo há declaração nominal dos acordes em tipo string. Após o conjunto de notas em relação são somadas, cada uma na sua respectiva frequência, para gerar um vetor que totaliza a soma das frequências tocadas ao longo de todo áudio. O procedimento seguinte, focando extrair um acorde desse vetor de notas ao longo de todo áudio, é utilizado uma correlação do mesmo com uma base dados carregada de notas pelos respectivos acordes. Ao final cada acorde da base de dados terá sua energia correspondente e, ao extrair o máximo das energias, é adquirido o acorde tom da música.

3.2.8 Procedimento 8: Extrair Acordes Fundamentais

A extração de acordes fundamentais é um módulo do sistema que possui como entrada o conjunto de notas binarizadas das partes de janela. A saída é um conjunto de acordes fundamentais ao longo do tempo. Segue a especificação do procedimento:

```
1 function set_of_chords_time = get_set_of_chords_time(set_of_notes_time)
```

```
2       load_chords_tone;
3
4       set_of_chords_time = {};
5       for set = 1:5
6           notes_time = set_of_notes_time{set};
7           time_total = length(notes_time(:,1));
8           chords_time(1:time_total) = 0;
9
10          for time = 1:time_total
11
12              notes_energy_tone(12) = 0;
13              for note = 1:12
14                  notes_energy_tone(note) = notes_time(time, note) + ...
15                  notes_time(time, note + 12) ...
16                      + notes_time(time, note + 2*12) + ...
17                       notes_time(time, note + 3*12) ...
18                          + notes_time(time, note + 4*12);
19              end
20
21              energy_chords(1:48) = 0;
22              for chord = 1:48
23                  energy_chords(chord) = sum((notes_energy_tone.* ...
24                          chords_tone_mask(:, chord)').^2);
25              end
26
27              max_chord = find(energy_chords==max(energy_chords));
28
29              chords_time(time) = max_chord(1);
30          end
31
32          set_of_chords_time{set} = chords_time;
33      end
34  end
```

No início desse processo há o carregamento da base de dados de acordes em relação as notas musicais. Após o conjunto de notas são somadas em relação às respectivas oitavas gerando vetor de somente 12 posições. Esse mesmo vetor é submetido então a um processo de correlação aos acordes derivados da base de dados. Ao final cada acorde da base de dados terá sua energia correspondente e, ao extrair o máximo das energias, é adquirido os acordes fundamentais ao longo do tempo.

3.2.9 Procedimento 9: Extrair Acorde Recorrente das Partes

A extração de acorde recorrente é um módulo do sistema que possui como entrada o conjunto de acordes das partes de janela. A saída são acordes fundamentais ao longo do

tempo com eliminação das 5 partes. Esse módulo é a etapa final de segmentação do sinal de áudio. Segue a especificação do procedimento:

```
function chords = analyse_set_of_chords(set_of_chords)

    set_of_chords_A = set_of_chords{1};
    set_of_chords_B = set_of_chords{2};
    set_of_chords_C = set_of_chords{3};
    set_of_chords_D = set_of_chords{4};
    set_of_chords_E = set_of_chords{5};

    total_seconds = length(set_of_chords_A);
    chords(1:total_seconds) = 0;
    for time = 1:total_seconds
        chords_to_analyse = [set_of_chords_A(time) ...
         set_of_chords_B(time) ...
         set_of_chords_C(time) set_of_chords_D(time) ...
          set_of_chords_E(time)];
        chords(time) = mode(chords_to_analyse);
    end
end
```

No início desse processo há a atribuição de variáveis a cada uma das 5 partes. Após o conjunto das partes em acordes é submetido a função **mode** para extrair o acorde mais recorrente dentro de uma dada faixa de tempo do áudio.

3.2.10 Procedimento 10: Extrair Acordes com Inversões

Da última etapa do sistema, a extração de acordes com inversões tem como finalidade formar acordes invertidos a partir da entrada dos acordes fundamentais e os baixos. Segue as especificações desse procedimento:

```
function chords_with_bass = get_chords_bass(chords_number, bass_time)

 load_dictionary_chords;

 % build chords with bass to translate to dictionary
 chords_with_bass_number = {};
 chord_iterator = 1;
 for chord = 1:48
    for bass = 1:12
        chords_with_bass_number{chord_iterator} = [chord, bass];
        chord_iterator = chord_iterator + 1;
    end
```

```
13  end
14
15  %------------------------------------------------------------------
16  chords_with_bass = {};
17  for time = 1:length(chords_number)
18      for chord = 1:length(chords_with_bass_number)
19         peer_chord = chords_with_bass_number{chord};
20         if peer_chord(1) == chords_number(time) && ...
21             peer_chord(2) == bass_time(time)
22             chords_with_bass{time} = dictionary_chords{chord};
23         end
24      end
25  end
26
27 end
```

No início desse processo há o carregamento de um vetor de acordes, cada um com uma nomeclatura de acorde. Após é construído pares de acordes e baixos para mapear os acordes tocados dentro do vetor de acordes nominais. Ao final do processo as células de acordes e baixos identificados são referenciados dentro do vetor de acordes nominais com inversões, através das células de pares de acordes e baixos.

3.3 Linha de Ciclos de Desenvolvimento

Nesta presente parte do trabalho será descrito os ciclos de desenvolvimento para a construção do sistema-solução. Os detalhes e o código completo podem ser encontrados no repositório *github* [1].

3.3.1 Estrutura do Ciclo

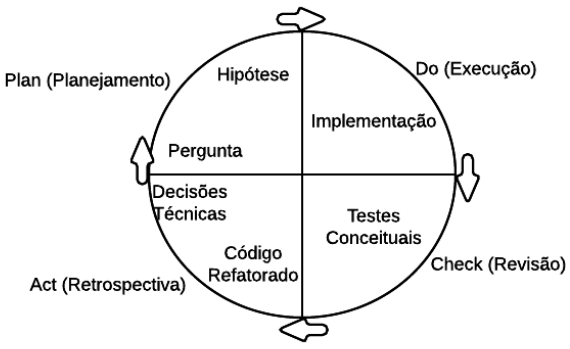

Figura 7 – Modelo de Ciclo Adaptado

[1] https://github.com/josepedro/TCC

Fases do ciclo de desenvolvimento:

- **Pergunta:** No início de cada ciclo é feito uma pergunta a ser respondida que se adere aos objetivos do trabalho.
- **Hipótese:** A partir dessa pergunta é feita hipóteses que possam responder;
- **Implementação:** As hipóteses são pensadas, construidas e implementadas num script;
- **Testes Conceituais:** Cada hipótese implementada é testada conforme a teoria usada;
- **Retrospectiva:** Dado os resultados dos testes conceituais e código refatorado, é feitto uma avaliação do que foi produzido e decisões técnicas são tomadas.

3.3.2 Ciclo 1

- **Pergunta:** Para saber da harmonia da música é preciso saber as notas dela. Se cada nota é uma frequência de vibração sonora, como analisar o sinal no ponto de vista de frequências? É possível?
- **Hipótese:** A Transformada de Fourier pode construir o espectro de frequências do sinal.
- **Implementação:**

```
1 som = som(1:length(som));
2 som = som/max(som);
3
4 t = fft(som);
5 SINAL=sqrt(t.*conj(t));
6 SINAL=SINAL/max(SINAL);
```

- **Testes Conceituais:** Testes foram feitos para ver se os picos de frequência correspondem ao sinal de entrada. O resultado foi positivo e os picos representam a energia das frequências;
- **Retrospectiva:** A Transformada de Fourier, em específico a $FastFourierTransform$, realmente produz resultados satisfatórios em determinar o espectro de energia da frequências. Mas as informações não estão configuradas para localizar as notas musicais.

3.3.3 Ciclo 2

- **Pergunta:** Como configurar essas informações para localizar as notas musicais?
- **Hipótese:** Dado que cada nota musical é um conjunto de frequências, realocar as energias frequenciais da transformada de fourier afim de que cada posição do vetor seje 1 unidedade de frequência (Hz) pode mapear a energia de cada nota.
- **Implementação:**

```
fs = 44100;
f = (0:length(som)-1)*fs/length(som);
freq = f(1:round(length(f)/2));
SOM = abs(fft(som));
SOM = SOM/max(SOM);
SOM = SOM(1:round(length(f)/2));
l = 1;
j = 0;
i = 1;
SOMA = 0;
while (i<length(freq))
    if (round(freq(i)) == round(freq(i+1)))
        SOMA = SOM(i+1) + SOMA;
        j = j + 1;
    else
        respfreq(l) = SOMA/(j+1);
        j = 0;
        SOMA = SOM(i+1);
        l = l+1;
    end
    i = i+1;
end
l = 0; j = 0; i = 0;
```

- **Testes Conceituais:** De fato as notas musicais foram localizadas com mais facilidade em determinados grupos de frequências. Tal ordenamento de frequências resultou num vetor de 22050 posições, independentemente do tamanho da amostra.
- **Retrospectiva:** A estratégia de realocar as energias decimais das frequências numa unidade de frequência se adere corretamente ao objetivo de encontrar as notas musicais. Entretanto as frequências não estão associadas as notas musicais.

3.3.4 Ciclo 3

- **Pergunta:** Dado um conjunto de frequências, como associar essas as notas musicais?

- **Hipótese:** Visto que associar as frequências as notas musicas é uma tarefa muito complexa para uma solução determinística, uma rede neural de aprendizado não supervisionado do tipo *ProbabilisticNeuralNetwork* (PNN) pode classificar um conjunto de frequências em sua respectiva nota musical.

- **Implementação:**

```
%BASE DE DADOS DE NOTAS MUSICAIS DA REDE NEURAL
%NOTAS

notas(12,22050) = 0; %matriz das notas

%Do grave
notas(1,61) = 0.1;
notas(1,62) = 0.2;
notas(1,63) = 0.4;
notas(1,64) = 0.6;
notas(1,65) = 0.8;
notas(1,66) = 1;
notas(1,67) = 0.8;
notas(1,68) = 0.6;
notas(1,69) = 0.4;
notas(1,70) = 0.2;
notas(1,71) = 0.1;

.
.
.

i = 1; %contador para andar ao longo do vetor
b = 0.15; %sensibilidade da rede


while (i <= 12)

      %S1(i) = exp(-(norm(rfeq - notas(i,:))*b));
      %correlacao = corrcoef(rfeq,notas(i,:));
      %S1(i) = correlacao(1,2);
      S1(i) = sum(abs(rfeq.* notas(i,:)));

    i = i + 1;
end
```

- **Testes Conceituais:** Foram testadas 3 funções de transferência do neurônio. A primeira função de transferência - a exponencial da subtração dos valores - não

foi muito eficaz pois para notas adjacentes as mesmas eram confundidas pela rede. Esse fato se dá pelo retorno de baixa magnitude da subtração de valores. A segunda função de transferência - a correlação dos valores - foi bastante eficaz para caracterizar notas. Porém a operação de subtração da média faz com que a energia final seja baixa, além de requisitar mais operações. A terceira função de transferência - a multiplicação dos valores - foi bastante eficaz para caracterizar notas e é rápida pois é uma forma simples da segunda função de transferência.

- **Retrospectiva:** A rede neural PNN foi bastante eficaz em classificar as frequências em termos de notas musicais. Porém as notas musicais não estão associadas a acordes musicais.

3.3.5 Ciclo 4

- **Pergunta:** Como adicionar as próximas camadas da rede para determinação dos acordes?
- **Hipótese:** Para poder mapear as notas é preciso adicionar mais 2 camadas. Uma camada para classificação de acordes, dado um conjunto de notas musicais e a outra para classificação de um acorde dado os conjuntos possíveis de acordes.
- **Implementação:**

```
% BASE DE DADOS PARA ACORDES

%--------------------------------
BD(12,48) = 0;
%--------------------------------

afin1 = 0; afin2 = 0;

%C
%CM
BD(12,1) = afin1;
BD(1,1) = 1; %baixo
BD(2,1) = afin2;
BD(4,1) = afin1;
BD(5,1) = 1; %terca
BD(6,1) = afin2;
BD(7,1) = afin1;
BD(8,1) = 1; %quinta
BD(9,1) = afin2;

.
```

```
22 .
23 .
24 %(430 LINHAS DE ACORDES)
25
26 %RADIAL BASIS LAYER para BD notas
27
28 while (i <= 48)
29     S2(i) = sum(abs(S1.* BD(i,:)));
30     i = i + 1;
31 end
```

- **Testes Conceituais:** Foram testadas todas as possibilidades de acordes e os que não foram evetivamente reconhecido foram as inversões e os acordes aumentados.
- **Retrospectiva:** De certo o acerto não foi total pois falta implementar uma camada que reconheça inversões.

3.3.6 Ciclo 5

- **Pergunta:** Como reconhecer acordes no tempo de tal forma a saber onde eles ocorrem?
- **Hipótese:** Uma solução de reconhecimento de energias ao longo do tempo pode ser eficaz para a determinação do rítmo. Em tese é calcular a energia do sinal e aplicar um filtro passa-baixas para identificação dos picos de energia.
- **Implementação:**

```
1      %filtering the pulses of minor energy
2      signal_filtered = filter_signal(bpm_music);
3      signal_pulses = signal_filtered;
4
5      % Building array with means movies
6      signal_pulses = decrease_resolution(signal_filtered, ...
7       file.fs, 1000);
8
9      % Beginnnig the correlation
10     array_correlation = correlate_moments(signal_pulses);
11
12     array_peaks = filter_peak(signal_pulses);
13
14     peaks = findpeaks(array_peaks);
15
16     number_of_peaks = length(peaks) + 1;
```

- **Testes Conceituais:** Para um caso específico a solução funcionou, porém os outros casos ela não se aderiu.

- **Retrospectiva:** De certo modo detectar onde acorde ocorre no sinal não agrega valor para o escopo desse trabalho pois os níveis de energia são muito variáveis e não há um padrão como para a detecção.

3.3.7 Ciclo 6

- **Pergunta:** Como ler o sinal todo e ter a visilibilidade dele em tempo e frequência?

- **Hipótese:** Para se ter uma resolução completa do sinal em tempo em frequência é preciso de ter uma transformada que agregue esses dois aspectos. Poder testar isso com a transformada wavelets.

- **Implementação:**

```
1  function [signal, imin, imax, iterations, energy] = ...
2  tree_iterator(signal, mini, maxi, imin, imax, iterations, energy)
3
4  if iterations == 0
5      if mini >= 1 && maxi <= 22050
6        [signal, imin, imax, iterations, energy] = ...
7        tree_iterator(signal, mini, maxi, 1, 22050, 1, 0);
8      else
9        imin = 1;
10       imax = 22050;
11       return;
12 end
13 elseif iterations > 0
14     imean = (imax - imin)/2 + imin;
15     %Low
16     if mini >= imin && maxi <= imean
17       [h0, h1] = wfilters('bior6.8');
18       [signal, y1] = decomposition_1level_qmf(h0, h1, signal);
19       energy = sum(abs(signal)) + energy;
20       iterations = iterations + 1;
21       imax = imean;
22       [signal, imin, imax, iterations, energy] = ...
23       tree_iterator(signal, mini, maxi, imin, ...
24       imax, iterations, energy);
25     %High
26     elseif mini >= imean && maxi <= imax
27       [h0, h1] = wfilters('bior6.8');
28       [y0, signal] = decomposition_1level_qmf(h0, h1, signal);
```

```
29        energy = sum(abs(signal)) + energy;
30        iterations = iterations + 1;
31        imin = imean;
32        [signal, imin, imax, iterations, energy] = ...
33        tree_iterator(signal, mini, maxi, ...
34        imin, imax, iterations, energy);
35      else
36        return;
37      end
38  end
```

- **Testes Conceituais:** A solução falhou num teste muito simples. Ao submeter um sinal puro numa frequencia determinada e constante o banco de filtros wavelets distorcia o sinal, deslocando a fase do sinal para frequencias adjacentes da original.
- **Retrospectiva:** Dado a barreira técnica de deslocamento de fase do sinal, ainda não foi encontrado uma solução de resolução tempo-frequência.

3.3.8 Ciclo 7

- **Pergunta:** Como ler o sinal todo e ter a visilibilidade dele em tempo e frequência?
- **Hipótese:** Para se ter uma resolução completa do sinal em tempo em frequência é preciso de ter uma transformada que agregue esses dois aspectos. Poder testar isso com a $ShortFourierTransform$ (Transformada de Fourier Janelada).
- **Implementação:**

```
1  function [notes_time, chords_time, chord_pitch] = DA3(signal, fs)
2
3  load_notes;
4  load_chords;
5  .
6  .
7  .
8  % begin to analyse music
9  time_seconds_total = fix((length(signal)/fs));
10 notes_time(time_seconds_total, 60) = 0;
11 chords_time = {};
12 for time = 1:time_seconds_total
13     signal_time = signal(1+((time-1)*fs):time*fs);
14     window = blackman(length(signal_time));
15     signal_time = window'.*signal_time;
16     signal_time = downsample(signal_time, 21);
17     fs_time = fs/21;
```

```
18      module_fft = abs(fft(signal_time));
19      respfreq(1:fs_time) = 0;
20      window_mean = length(signal_time)/fs_time;
21      for frequency = 1:fs_time
22          respfreq(frequency) = sum(module_fft( ...
23          1+((frequency-1)*window_mean):frequency* ...
24          window_mean))/window_mean;
25      end
26      respfreq = respfreq(1:fix(length(respfreq)/2));
27      for note = 1:60
28          notes_time(time, note) = sum(respfreq.* ...
29          notes(note,:));
30      end
31      energy_chords(1:48) = 0;
32      for chord = 1:48
33          energy_chords(chord) = sum(notes_time(time, :) ...
34          .*chords(chord,:));
35      end
36      chords_time{time} = dictionary_chords{ ...
37      find(energy_chords==max(energy_chords))};
38  end
39  notes_energy_total = notes_time(1,:);
40  for time = 2:time_seconds_total
41      notes_energy_total = notes_energy_total + notes_time(time,:);
42  end
43  energy_chords(1:48) = 0;
44  for chord = 1:48
45      energy_chords(chord) = sum(notes_energy_total.*chords(chord,:));
46  end
47  chord_pitch = dictionary_chords{find(energy_chords== ...
48  max(energy_chords))};
```

- **Testes Conceituais:** A solução da transformada de fourier janelada foi testada com acordes de violão e piano ao longo do tempo e o resultado foi satisfatório exceto para acordes de transição.

- **Retrospectiva:** A solução da transformada de fourier janelada se encaixou bem no conjunto.

3.3.9 Ciclo 8

- **Pergunta:** Como extrair o tom da música?

- **Hipótese:** Para extrair o tom da música é preciso somar a energia das notas totais ao longo da música.

- **Implementação:**

```
function [chord_pitch, chord_pitch_number] = ...
get_chord_pitch(notes_time, time_seconds_total, chords)

  dictionary_chords = ...
.
.
.
  notes_energy_total(60) = 0;
  for note = 1:60
    notes_energy_total(note) = sum([notes_time(:,note)]);
  end

  % discover tone music
  notes_energy_tone(12) = 0;
  for note = 1:12
    notes_energy_tone(note) = notes_energy_total(note) ...
     + notes_energy_total(note + 12) ...
      + notes_energy_total(note + 2*12) + ...
      notes_energy_total(note + 3*12) ...
        + notes_energy_total(note + 4*12);
  end

  % find chord tone
  load_chords_tone;
  chords_tone(48) = 0;
  for chord = 1:48
    chords_tone(chord) = sum((notes_energy_tone.* ...
    chords_tone_mask(:, chord)'.^2));
  end

  chord_pitch_number = find(chords_tone==max(chords_tone));
  chord_pitch = dictionary_chords{chord_pitch_number};
```

- **Testes Conceituais:** A solução foi testada com sequencia de acordes do violão e as vezes o tom não é o certo.

- **Retrospectiva:** A quantidade de energia está atrapalhando a extração do tom. O tom é definido como a frequência de aparecimento das notas ao longo da música.

3.3.10 Ciclo 9

- **Pergunta:** Como trabalhar com a frequencia de aparecimento das notas?

- **Hipótese:** Se binarizar com 1 e 0 o mapa de notas no tempo a soma das notas será unitária, equivalente a frequência.
- **Implementação:**

```
% binarize set of notes
    for set = 1:5
        notes_time = set_of_notes_time{set};

        for time = 1:time_seconds_total
            for note = 1:60
                if notes_time(time, note) < max(max(notes_time))/180
                    notes_time(time, note) = 0;
                else
                    notes_time(time, note) = 1;
                end
            end
        end

        set_of_notes_time{set} = notes_time;
    end
```

- **Testes Conceituais:** A solução foi testada e verificada com picos somente de 1 e vales somente de 0.
- **Retrospectiva:** Com essa binarização o tom da música foi efetivamente corrigido.

3.3.11 Ciclo 10

- **Pergunta:** Em relação aos acordes transitórios, como corrigir?
- **Hipótese:** Se ao deslocar a janela de tamanho de 1 segundo em passos de 0.2 segundos e calcular o espectro de frequência de cada passo pode-se fazer a média do acorde de cada tempo e poder cancelar os acordes transitórios.
- **Implementação:**

```
function set_of_windows_signals = ...
build_window_short_fft(signal, time, fs)
  signal = [signal(:)];

    % part A
    time_start_A = round(1+((time-1)*fs));
    time_end_A = round(time*fs);
```

```
 8      signal_time_A = signal(time_start_A:time_end_A);
 9      window = blackman(length(signal_time_A));
10      signal_time_A = window.*signal_time_A;
11
12      % part B (displacement = + 0.2 seconds)
13      time_start_B = round(1+((time-1)*fs+0.2*fs));
14      time_end_B = round((time+0.2)*fs);
15      if time_start_B < length(signal) && time_end_B <= length(signal)
16          signal_time_B = signal(time_start_B:time_end_B);
17          window = blackman(length(signal_time_B));
18          signal_time_B = window.*signal_time_B;
19      else
20          signal_time_B(length(signal)) = 0;
21      end
22
23      % part C (displacement = + 0.4 seconds)
24      time_start_C = round(1+((time-1)*fs+0.4*fs));
25      time_end_C = round((time+0.4)*fs);
26      if time_start_C < length(signal) && time_end_C <= length(signal)
27          signal_time_C = signal(time_start_C:time_end_C);
28          window = blackman(length(signal_time_C));
29          signal_time_C = window.*signal_time_C;
30      else
31          signal_time_C(length(signal)) = 0;
32      end
33
34      % part D (displacement = + 0.6 seconds)
35      time_start_D = round(1+((time-1)*fs+0.6*fs));
36      time_end_D = round((time+0.6)*fs);
37      if time_start_D < length(signal) && time_end_D <= length(signal)
38          signal_time_D = signal(time_start_D:time_end_D);
39          window = blackman(length(signal_time_D));
40          signal_time_D = window.*signal_time_D;
41      else
42          signal_time_D(length(signal)) = 0;
43      end
44
45      % part E (displacement = + 0.8 seconds)
46      time_start_E = round(1+((time-1)*fs+0.8*fs));
47      time_end_E = round((time+0.8)*fs);
48      if time_start_E < length(signal) && time_end_E <= length(signal)
49          signal_time_E = signal(time_start_E:time_end_E);
50          window = blackman(length(signal_time_E));
51          signal_time_E = window.*signal_time_E;
52      else
53          signal_time_E(length(signal)) = 0;
54      end
```

```
55
56      set_of_windows_signals = {};
57      if length(signal_time_A) == length(signal_time_B) && ...
58          length(signal_time_A) == length(signal_time_C) && ...
59           length(signal_time_A) == length(signal_time_D) &&  ...
60            length(signal_time_A) == length(signal_time_E)
61          set_of_windows_signals{1} = signal_time_A;
62          set_of_windows_signals{2} = signal_time_B;
63          set_of_windows_signals{3} = signal_time_C;
64          set_of_windows_signals{4} = signal_time_D;
65          set_of_windows_signals{5} = signal_time_E;
66      else
67          set_of_windows_signals{1} = signal_time_A;
68          set_of_windows_signals{2} = signal_time_A;
69          set_of_windows_signals{3} = signal_time_A;
70          set_of_windows_signals{4} = signal_time_A;
71          set_of_windows_signals{5} = signal_time_A;
72      end
```

- **Testes Conceituais:** Os testes foram feitos em series de acordes tocados no violão e de fato os acordes transitórios desapareceram.
- **Retrospectiva:** Com essa correção os acordes maiores, menores e diminutos estão sendo reconhecidos corretamente.

3.3.12 Ciclo 11

- **Pergunta:** Como extrair a nota mais grave (baixo) de cada período do tempo?
- **Hipótese:** Com o mapa de ntoas binarizado é possível extrair o baixo pegando a primeira posição com valor 1 de cada tempo.
- **Implementação:**

```
1  function bass_time = get_bass(set_of_notes_time)
2
3    notes_time_A = set_of_notes_time{1};
4    notes_time_B = set_of_notes_time{2};
5    notes_time_C = set_of_notes_time{3};
6    notes_time_D = set_of_notes_time{4};
7    notes_time_E = set_of_notes_time{5};
8
9    total_seconds = length(notes_time_A(:,1));
10   notes_time(total_seconds, 60) = 0;
11   for time = 1:total_seconds
```

```
12      for note = 1:60
13        notes_to_analyse = [notes_time_A(time, note) ...
14         notes_time_B(time, note) ...
15        notes_time_C(time, note) ...
16        notes_time_D(time, note) notes_time_E(time, note)];
17         notes_time(time, note) = mode(notes_to_analyse);
18      end
19    end
20
21      bass_time(1:total_seconds) = 0;
22    for time = 1:total_seconds
23      maxs = find(notes_time(time,:)==max(notes_time(time,:)));
24      bass_time(time) = maxs(1);
25    end
26
27    for bass = 1:length(bass_time)
28      bass_time(bass) = mod(bass_time(bass) - 1, 12) + 1;
29    end
30
31  end
```

- **Testes Conceituais:** Os testes foram feitos com acordes e de fato ele reconhece as notas mais graves.
- **Retrospectiva:** Dado os baixos definidos, os acordes com inversões e aumentados não estão incluídos.

3.3.13 Ciclo 12

- **Pergunta:** Como extrair a nota mais grave (baixo) de cada período do tempo e incluir os acordes aumentados e invertidos?
- **Hipótese:** Com o mapa de ntoas binarizado é possível extrair o baixo pegando a primeira posição com valor 1 de cada tempo.
- **Implementação:**

```
1  function chords_with_bass = ...
2  get_chords_bass(chords_number, bass_time)
3  dictionary_chords = ...
4  .
5  .
6  .
7   % build chords with bass to translate to dictionary
8   chords_with_bass_number = {};
```

```
 9  chord_iterator = 1;
10  for chord = 1:48
11   for bass = 1:12
12     chords_with_bass_number{chord_iterator} = [chord, bass];
13     chord_iterator = chord_iterator + 1;
14   end
15  end
16  chords_with_bass = {};
17  for time = 1:length(chords_number)
18    for chord = 1:length(chords_with_bass_number)
19     peer_chord = chords_with_bass_number{chord};
20     if peer_chord(1) == chords_number(time) && ...
21     peer_chord(2) == bass_time(time)
22       chords_with_bass{time} = dictionary_chords{chord};
23     end
24    end
25  end
26 end
```

- **Testes Conceituais:** Todas as possibilidades de acordes foram reconhecidos com sucesso.
- **Retrospectiva:** Os objetivos do trabalho foram alcançados com sucesso.

4 Resultados

Em vista dos procedimentos teóricos aliados a uma solução computacional, obteve-se os seguintes resultados:

- resposta em frequência;
- sugestão de notas;
- sugestão de acordes;
- detecção de transições rítmicas;
- implementação da transformada wavelets;
- transcrição de notas ao longo do tempo;
- extração da tonalidade do áudio;
- transcrição automática de acordes ao longo do tempo.

4.1 Resposta em Frequência e Sugestões de Notas e Acordes

Para a demonstração de tais resultados foram feitos experimentos com todas as possibilidades de detecção de acordes proporcionados pelo sistema. Todavia serão detalhados e comentados somente 4 pois para os outros equivalem as mesmas considerações. O resumo dos resultados dos outros experimentos estará presente na tabela que se segue logo após. Esses experimentos foram feitos levando em consideração pré-condições e pós-condições.

4.1.1 Pré-condições dos Experimentos

No que tange às pré-condições foram levados em conta:

- teclado yamaha E413 com som de piano para a execução dos acordes;
- somente tríades (3 notas) tocadas;
- o software Audacity [1] foi utilizado para gravação;
- o microfone convencional interno do *notebook* foi utilizado para aquisição dos sinais de áudio;
- o ruído de fundo estava com uma grandeza por volta de 45 db;
- a taxa de amostragem do sinal foi configurada em 44100 Hz;
- gravação do áudio no formato de arquivo .wav em codificação 16 pcm.
- inicialização do software Scilab;
- execução do comando para importar a função principal do sistema (Anexo, A1): exec dc.sci;
- abertura do arquivo de áudio: audio = wavread('arquivo.wav');
- execução do algorítmo tendo o áudio gravado como parâmetro: acorde = DA2(audio);
- o resultado do acorde tocado estará guardado na variável acorde;
- para a geração de resultados automáticos com todas as amostras representantes de todas as possibilidades de acordes, faz-se necessário da execução do seguinte comando (Anexo, A11): exec acordes_test.sci.

[1] http://www.audacity.sourceforge.net

As tríades de acordes foram executadas com base na nota central $C4$ que possui o valor de aproximadamente 261,6 Hz. Segue uma foto ilustrativa das regiões e limites usados (DOZOL, 2014):

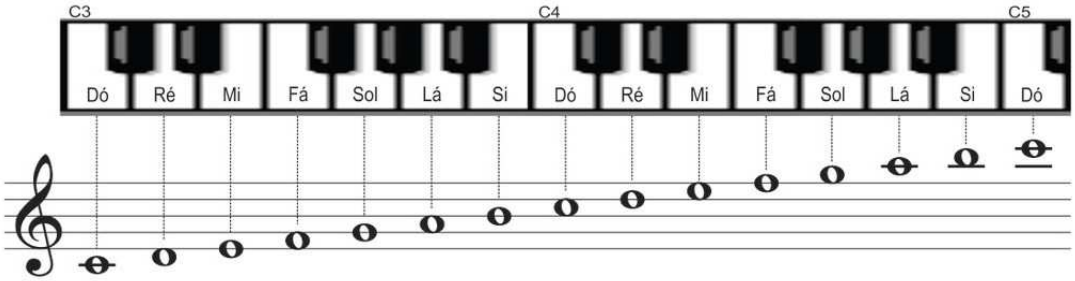

Figura 8 – Teclado ilustrativo para execução dos acordes

O processo de execução do experimento foi dividido em 4 etapas. A primeira relativa a gravação do acorde tocado no teclado via microfone convencional interno do *notebook*. A segunda é a exportação do som no formato de arquivo .wav pelo software audacity. A terceira etapa é a introdução do arquivo na entrada do sistema de detecção de acordes. A última atividade é a classificação do arquivo digital num acorde. Segue esquema ilustrativo do processo:

Figura 9 – Processo ilustrativo da execução dos experimentos

4.1.2 Experimento 1 - Acorde CM

Nesse experimento foi tocado a tríade *Dó* (baixo e tônica), Mi e Sol equivalente ao acorde CM. A tríade foi tocada ao mesmo tempo e com a mesma força para todas as notas.

Segue os gráficos resultantes:

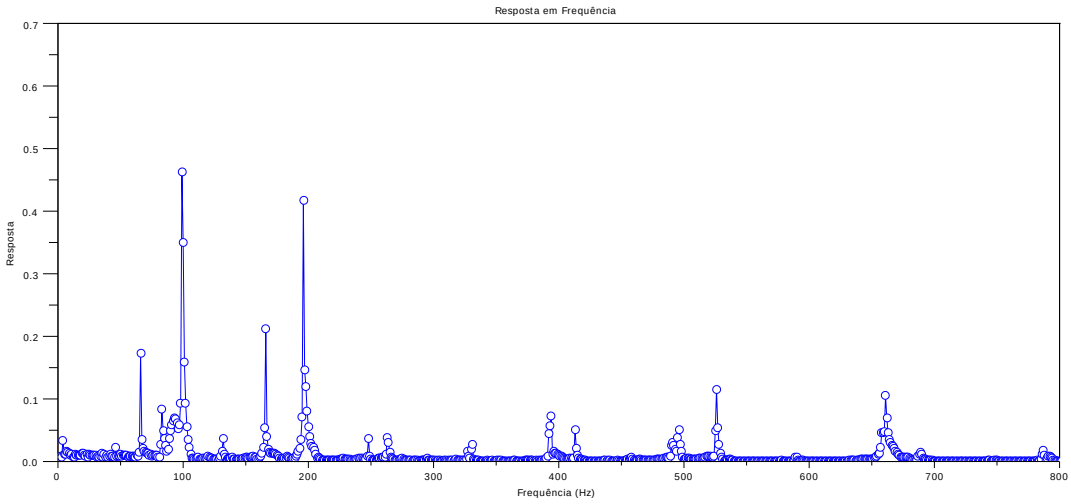

Figura 10 – Gráfico da resposta em frequência para a gravação do acorde CM

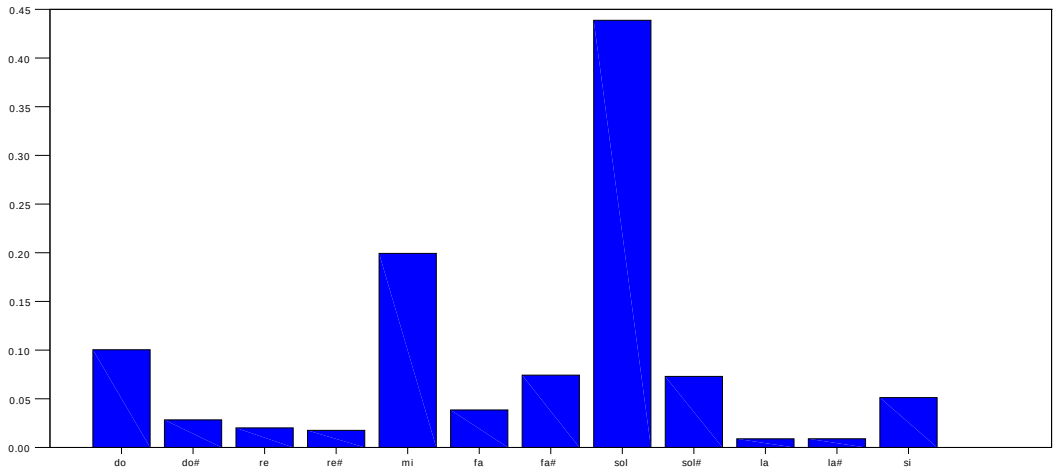

Figura 11 – Gráfico de sugestão de notas para a gravação do acorde CM

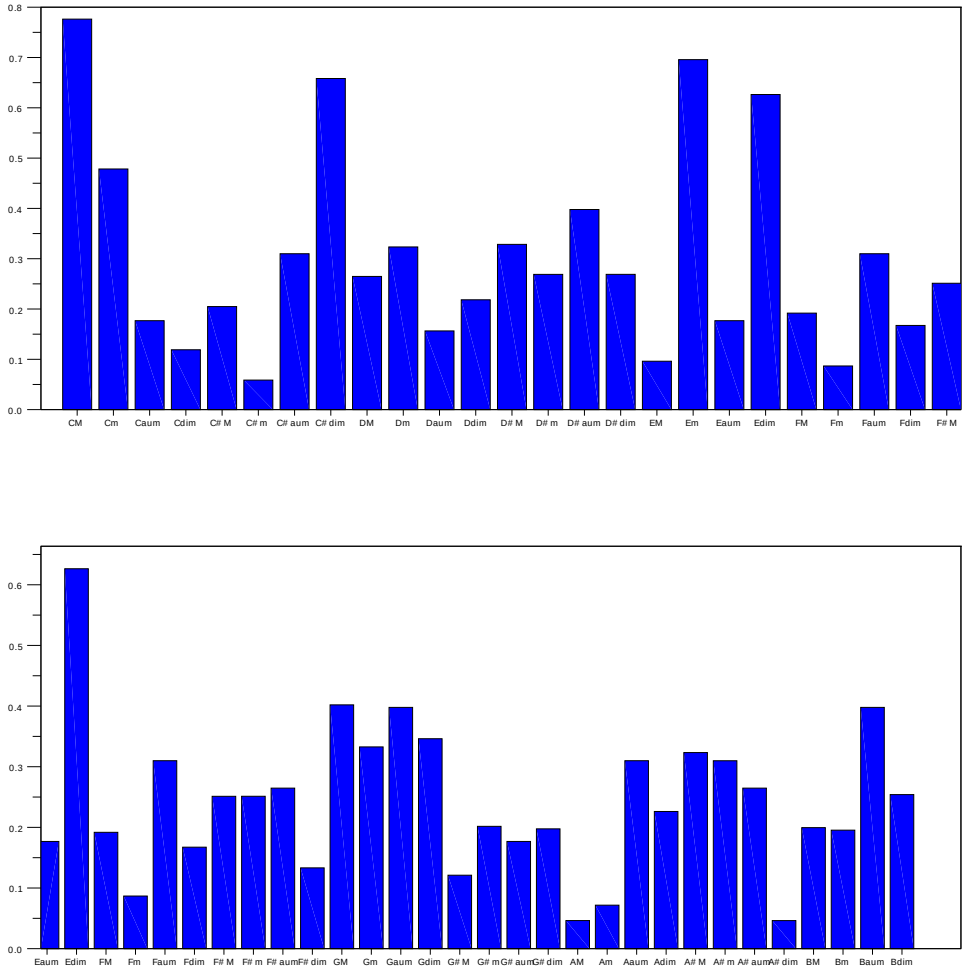

Figura 12 – Gráficos de sugestão de acordes a gravação do acorde CM

Do resultado da primeira camada de processamento é gerado o gráfico da Figura 16. Esse gráfico diz respeito a natureza da composição do sinal em senoides em termos de transformada de fourier. O primeiro pico, no valor de 294 Hz, é relativo a nota *Dó*. O segundo pico, no valor de 371 Hz, é relativo a nota Mi. O terceiro pico, no valor de 441 Hz, é relativo a nota *Sol*. Os picos seguintes são relativos aos harmônicos dessas três notas.

Do resultado da segunda camada de processamento é gerado gráfico da Figura 17. É possível perceber nele que as notas *Dó*, Mi e *Sol* são as que mais possuem energia ou, no ponto de vista de sugestão, as mais sugeridas. De certa forma um dos fatores que contribuiram das notas *Dó* e *Sol* ser de maiores energias foi devido a presença dos harmônicos.

Do resultado da terceira camada de processamento são gerados os gráficos da Figura 18. Essa camada é relativa ao resultados das sugestões de acordes musicais. É

perceptível ver a presença da alta sugestão do acorde CM.

4.1.3 Experimento 2 - Acorde Dm

Nesse experimento foi tocado a tríade Ré (baixo e tônica), Fá e Lá equivalente ao acorde Dm. A tríade foi tocada ao mesmo tempo e com a mesma força para todas as notas.

Segue os gráficos resultantes:

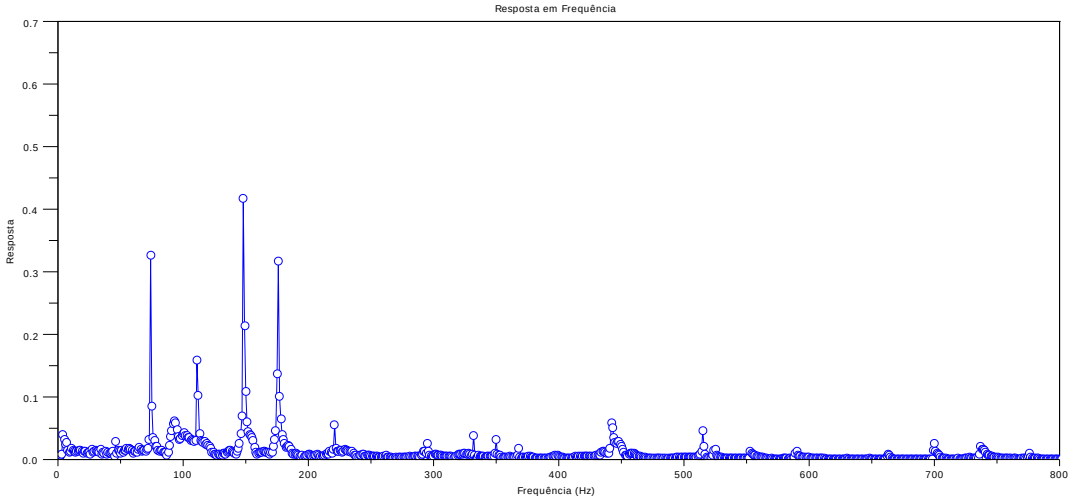

Figura 13 – Gráfico da resposta em frequência para a gravação do acorde Dm

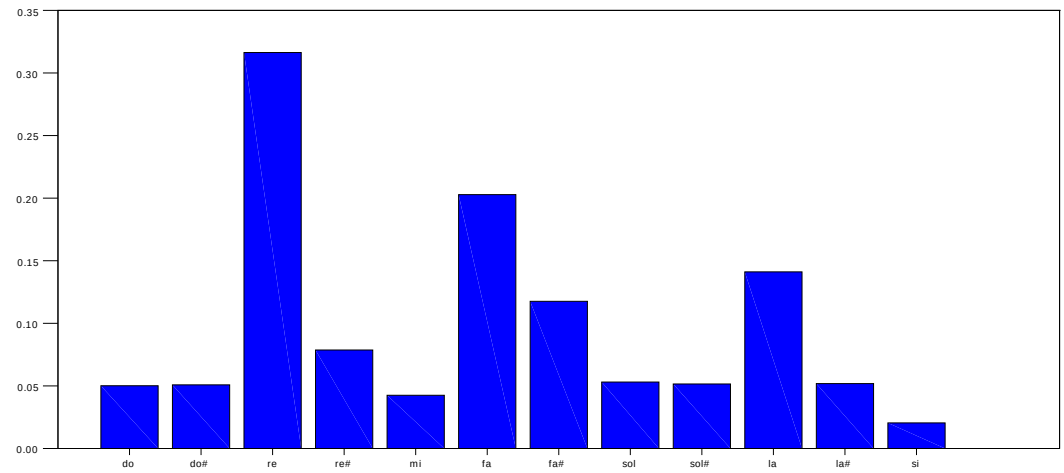

Figura 14 – Gráfico de sugestão de notas para a gravação do acorde Dm

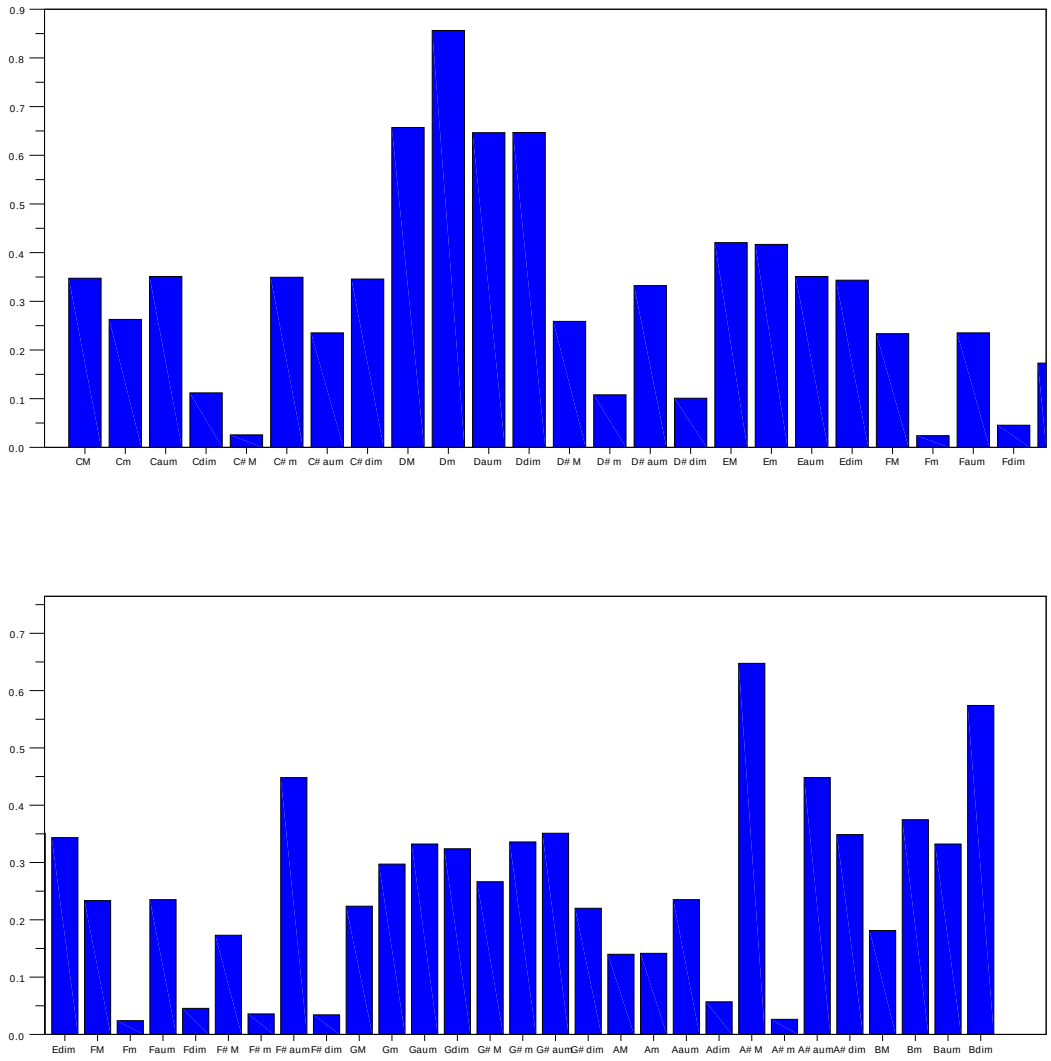

Figura 15 – Gráficos de sugestão de acordes a gravação do acorde *Dm*

Do resultado da primeira camada de processamento é gerado o gráfico da Figura 19. Esse gráfico diz respeito a natureza da composição do sinal em senoides em termos de transformada de fourier. O primeiro pico, no valor de 294 Hz, é relativo a nota *Ré*. O segundo pico, no valor de 350 Hz, é relativo a nota *Fá*. O terceiro pico, no valor de 441 Hz, é relativo a nota *Lá*. Os picos seguintes são relativos aos harmônicos dessas três notas.

Do resultado da segunda camada de processamento é gerado gráfico da Figura 20. É possível perceber nele que as notas *Ré*, *Fá* e *Lá* são as que mais possuem energia ou, no ponto de vista de sugestão, as mais sugeridas. De certa forma um dos fatores que contribuiram das notas *Ré* e *Lá* ser de maiores energias foi devido a presença dos harmônicos.

Do resultado da terceira camada de processamento são gerados os gráficos da Figura 21. Essa camada é relativa ao resultados das sugestões de acordes musicais. É perceptível ver a presença da alta sugestão do acorde Dm.

4.1.4 Experimento 3 - Acorde $Ddim$

Nesse experimento foi tocado a tríade $Ré$ (baixo e tônica), $Fá$ e $Sol\#$ equivalente ao acorde Ddim. A tríade foi tocada ao mesmo tempo e com a mesma força para todas as notas.

Segue os gráficos resultantes:

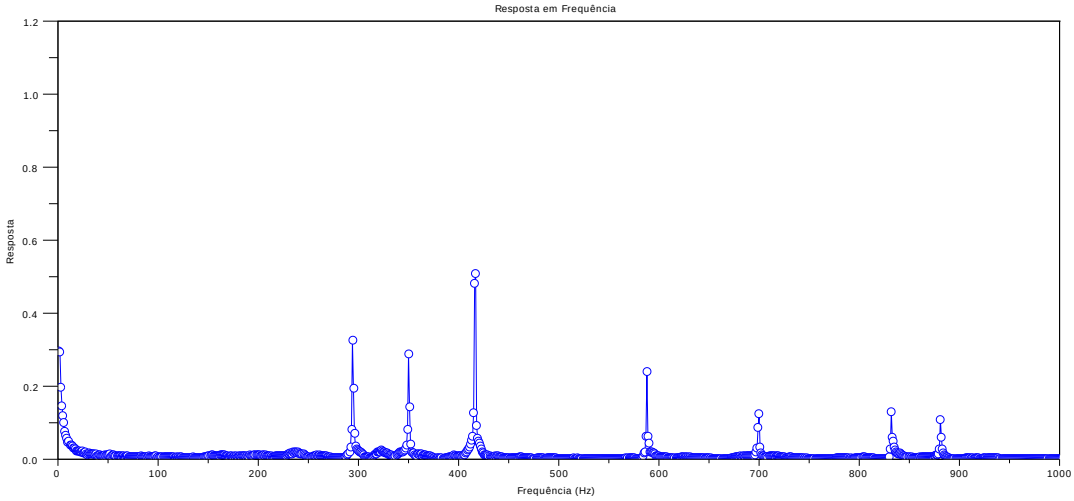

Figura 16 – Gráfico da resposta em frequência para a gravação do acorde $Ddim$

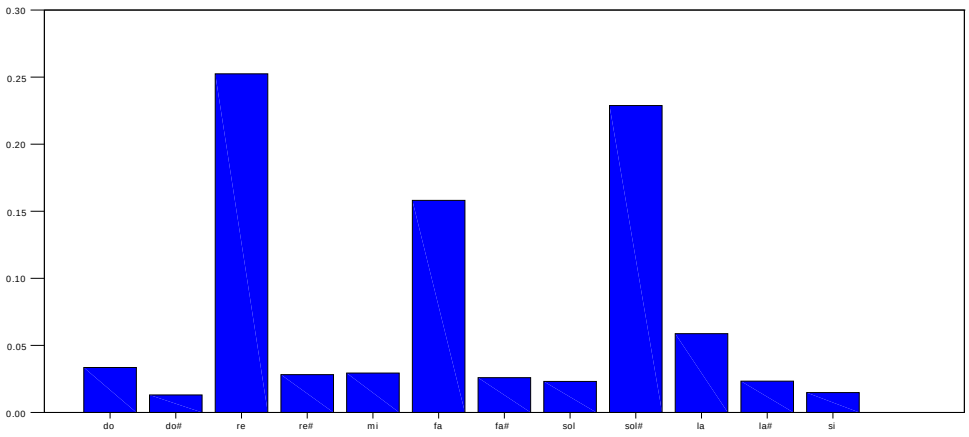

Figura 17 – Gráfico de sugestão de notas para a gravação do acorde $Ddim$

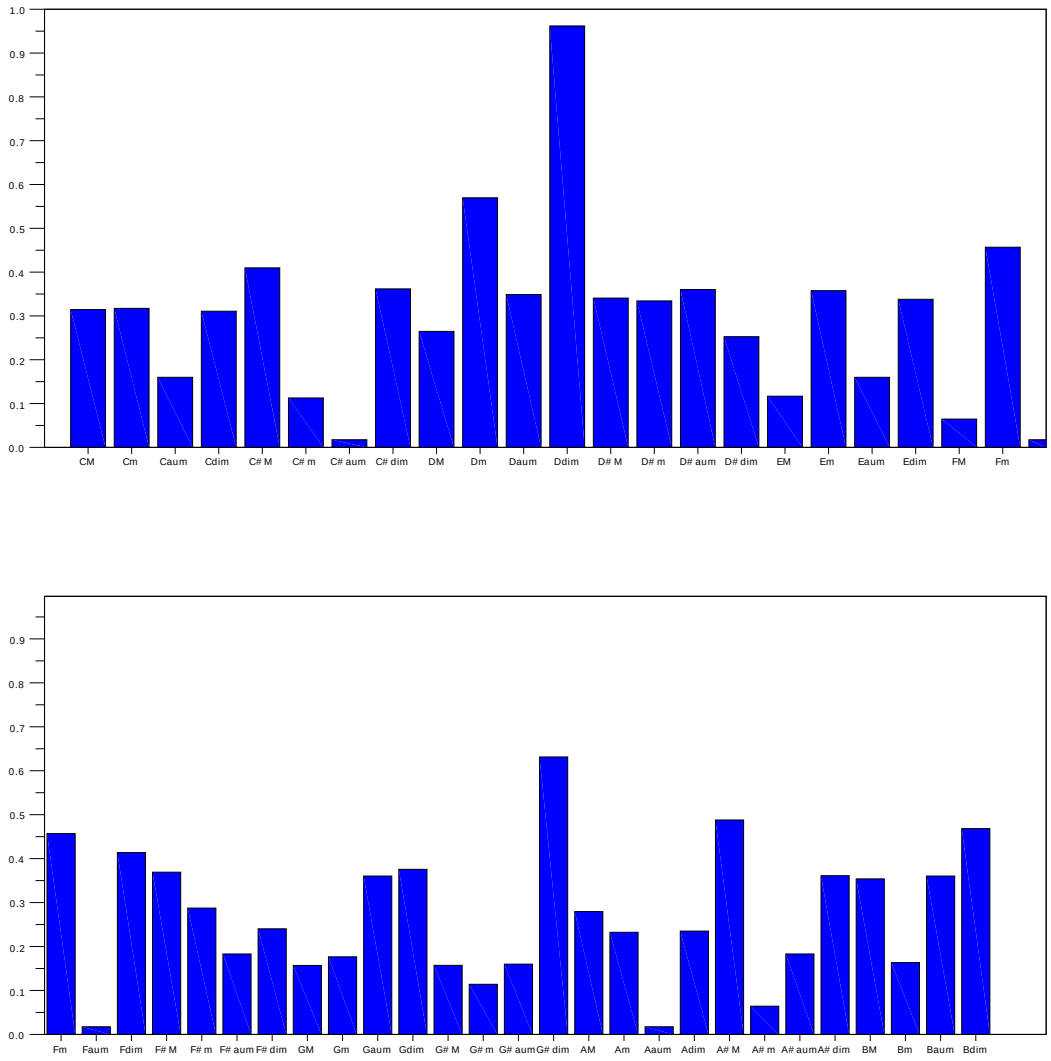

Figura 18 – Gráficos de sugestão de acordes a gravação do acorde *Ddim*

Do resultado da primeira camada de processamento é gerado o gráfico da Figura 22. Esse gráfico diz respeito a natureza da composição do sinal em senoides em termos de transformada de fourier. O primeiro pico, no valor de 294 Hz, é relativo a nota *Ré*. O segundo pico, no valor de 350 Hz, é relativo a nota *Fá*. O terceiro pico, no valor de 417 Hz, é relativo a nota *Sol#*. Os picos seguintes são relativos aos harmônicos dessas três notas.

Do resultado da segunda camada de processamento é gerado gráfico da Figura 23. É possível perceber nele que as notas *Ré*, *Fá* e *Sol#* são as que mais possuem energia ou, no ponto de vista de sugestão, as mais sugeridas.

Do resultado da terceira camada de processamento são gerados os gráficos da Figura 24. Essa camada é relativa ao resultados das sugestões de acordes musicais. É

perceptível ver a presença da alta sugestão do acorde *Ddim*.

4.1.5 Experimento 4 - Acorde *Daum*

Nesse experimento foi tocado a tríade *R*é (baixo e tônica), *F*á# e *L*á# equivalente ao acorde *Daum*. A tríade foi tocada ao mesmo tempo e com a mesma força para todas as notas.

Segue os gráficos resultantes:

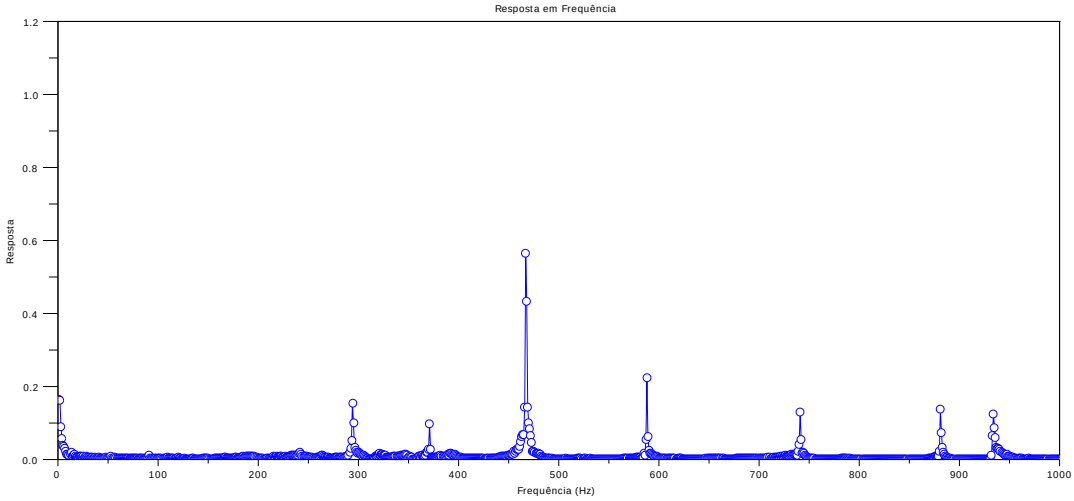

Figura 19 – Gráfico da resposta em frequência para a gravação do acorde *Daum*

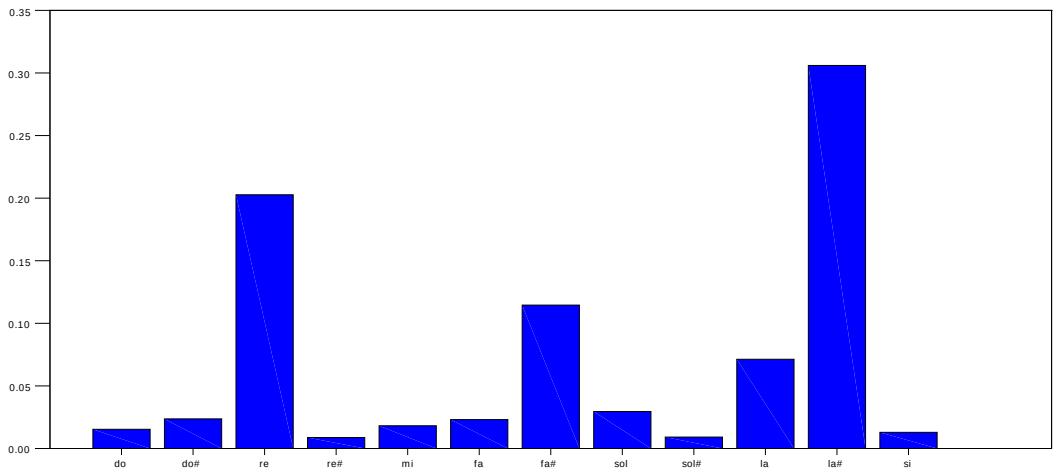

Figura 20 – Gráfico de sugestão de notas para a gravação do acorde *Daum*

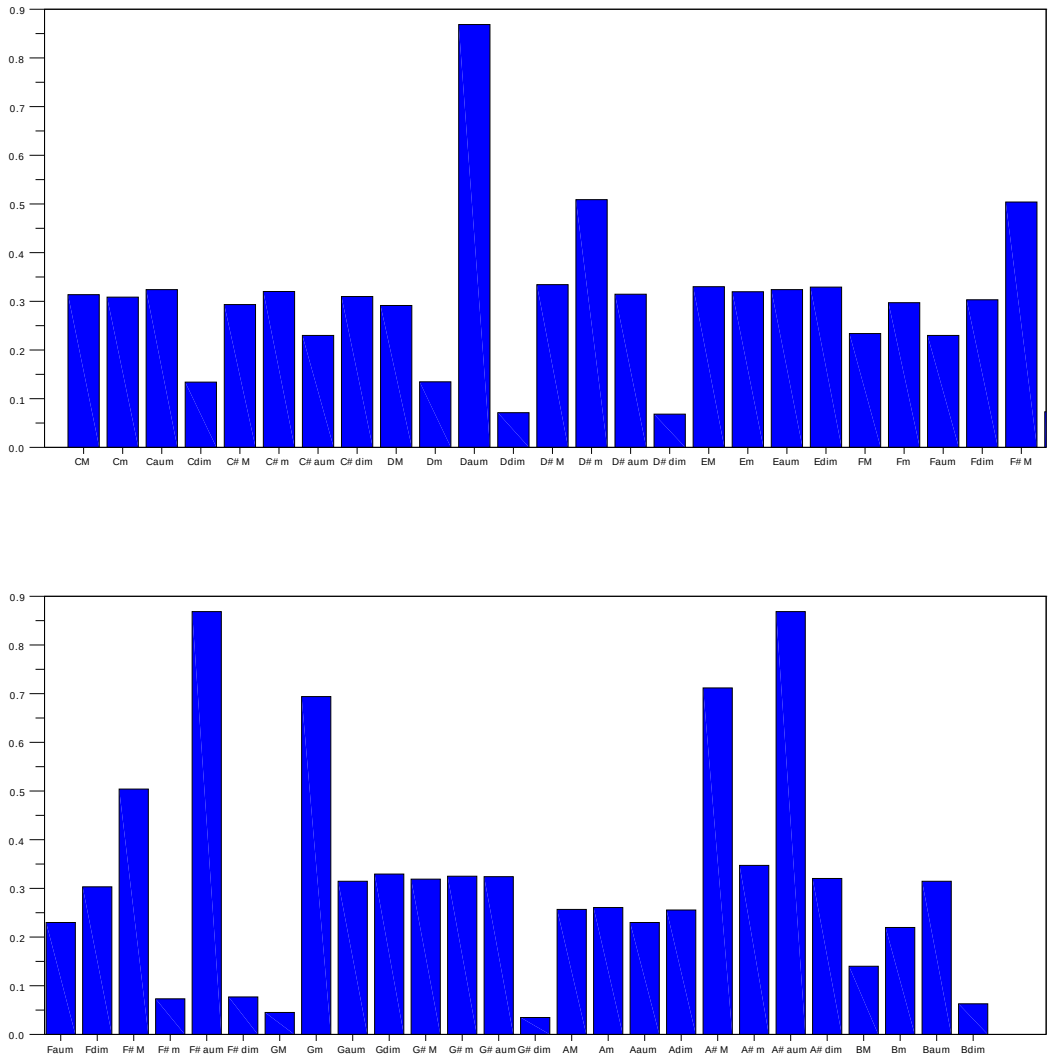

Figura 21 – Gráficos de sugestão de acordes a gravação do acorde *Daum*

Do resultado da primeira camada de processamento é gerado o gráfico da Figura 25. Esse gráfico diz respeito a natureza da composição do sinal em senoides em termos de transformada de fourier. O primeiro pico, no valor de 294 Hz, é relativo a nota *Ré*. O segundo pico, no valor de 371 Hz, é relativo a nota *Fá#*. O terceiro pico, no valor de 467 Hz, é relativo a nota *Lá#*. Os picos seguintes são relativos aos harmônicos dessas três notas.

Do resultado da segunda camada de processamento é gerado gráfico da Figura 26. É possível perceber nele que as notas *Ré*, *Fá#* e *Lá#* são as que mais possuem energia ou, no ponto de vista de sugestão, as mais sugeridas. De certa forma, essas mesmas notas compõe os acordes *F#aum* e *A#aum* causando um erro de redundância de informação no sistema.

Do resultado da terceira camada de processamento são gerados os gráficos da Figura 27. Essa camada é relativa ao resultados das sugestões de acordes musicais. É perceptível a alta sugestão dos acordes Daum, $F\#aum$ e $A\#aum$ com a mesma quantidade de energia. Isso é devido às notas comporem os mesmos acordes, diferenciando um do outro somente pela nota mais grave da tríade. Como o sistema não possui um módulo de detecção de baixos, esses 3 acordes são congruentes.

4.1.6 Tabela de resultados dos acordes tocados

Segue a tabela de resultados do sistema dado um acorde tocado. Esses resultados foram gerados pelo script referenciado em anexo (Anexo, A11). Os acordes em negrito são aqueles que o sistema errou devido a falta de um módulo de software que diferenciasse as notas em termos de baixos e inversões, interpretando-as como tríades no modo fundamental. E esse fato é tão relevante que o sistema desconsidera as inversões.

Tais informações e mais outras estão referencidas na tabela a seguir:

4.2 Detecção de Transições Rítmicas

4.3 Implementação da Transformada Wavelets

4.4 Transcrição de Notas ao Longo do Tempo

4.5 Extração da Tonalidade do Áudio

4.6 Transcrição Automática de Acordes ao Longo do Tempo

-	**Acorde Fundamental**	**Quinta Invertida**	**Terça Invertida**
C	C	C/G	C/E
Cm	Cm	Cm/G	Cm/D#
Caum	Caum	G#aum	Eaum
Cdim	Cdim	Cdim/F#	Cdim/D#
C#	C#	C#/G#	C#/F
C#m	C#m	C#m/G#	C#m/E
C#aum	C#aum	Aaum	Faum
C#dim	C#dim	C#dim/G	C#dim/E
D	D	D/A	D/F#
Dm	Dm	Dm/A	Dm/F
Daum	Daum	A#aum	F#aum
Ddim	Ddim	Ddim/G#	Ddim/F
D#	D#	D#/A#	D#/G
D#m	D#m	D#m/A#	D#m/F#
D#aum	D#aum	Baum	Gaum
D#dim	D#dim	D#dim/A	D#dim/F#
E	E	E/B	E/G#
Em	Em	Em/B	Em/G
Eaum	Eaum	Caum	G#aum
Edim	Edim	Edim/A#	Edim/G
FM	F	F/C	F/A
Fm	Fm	Fm/C	Fm/G#
Faum	Faum	C#aum	Aaum
Fdim	Fdim	Fdim/B	Fdim/G#
F#	F#	F#/C#	F#/A#
F#m	F#m	F#m/C#	F#m/A
F#aum	F#aum	Daum	A#aum
F#dim	F#dim	F#dim/C	F#dim/A
G	G	G/D	G/B
Gm	Gm	Gm/D	Gm/A#
Gaum	Gaum	D#aum	Baum
Gdim	Gdim	Gdim/C#	Gdim/A#
G#	G#	G#/D#	G#/C
G#m	G#m	G#m/D#	G#m/B
G#aum	G#aum	Eaum	Caum
G#dim	G#dim	G#dim/D	G#dim/B
A	A	A/E	A/C#
Am	Am	Am/E	Am/C
Aaum	Aaum	Faum	C#aum
Adim	Adim	Adim/D#	Adim/C
A#	A#	A#/F	A#/D
A#m	A#m	A#m/F	A#m/C#
A#aum	A#aum	F#aum	Daum
A#dim	A#dim	A#dim/E	A#dim/C#
B	B	B/F#	B/D#
Bm	Bm	Bm/F#	Bm/D
Baum	Baum	Gaum	D#aum
Bdim	Bdim	Bdim/F	Bdim/D

Tabela 1 – Tabela de resultados dado os acordes tocados com inversões

5 Considerações Finais

Em vista do que foi exposto, conclui-se que o sistema é de viabilidade significativa no que tange a aplicação e função principal: reconhecimento de acordes num conjunto de amostras de sinal de áudio.

No que tange aos problemas de inversões que impactam os acordes aumentados, uma solução de curto prazo é sugerir o primeiro acorde ocorrido de maior energia no conjunto de sugestões. Outra solução, que é de longo prazo, é a implementação de uma camada para detecção de inversões. A partir da detecção de inversão será possível distinguir acordes aumentados e adicionar novos acordes. Nesse trabalho em específico foi implementada a solução de curto prazo porém para o trabalho que se segue (trabalho de conclusão de curso 2) haverá a implementação da camada de detecção de inversões.

Em relação a consolidação do algorítmo é pertinente expor que o mesmo não está otimizado e nem analisado no ponto de vista de complexidade. No trabalho que se segue haverá uma análise mais profunda sobre esses aspectos.

Não houve teste para a presente solução em outros instrumentos harmônicos como o violão. Subentende-se que poderá funcionar corretamente mas com algumas restrições devido ao ataque do instrumento. Para tal poderá ser cabível uma adaptação para cada tipo diferente de instrumento.

5.1 Evoluções Futuras

No que diz respeito a futuras evoluções, é passível de consideração o uso das transformadas *wavelets* para o aprimoramento da detecção de acordes localizados no tempo. É desejável um algorítmo para análise de audio na detecção de transições rítmicas ao longo do tempo, focando localizar aonde os acordes se encontram num determinado compasso musical. Também há a possibilidade de implementação do sistema num produto de software, mais especificamente numa plataforma móvel Android. Para tais atividades futuras foi feito um cronograma referenciado no apêndice A.12.

Referências

ANTARES. *Auto-Tune 7*. 2014. Disponível em: <http://www.antarestech.com/products/detail.php?product=Auto-Tune_7_1>. Citado na página 23.

BARBANCHO, I. Pic detector for piano chords. *EURASIP Journal on Advances in Signal Processing*, Hindawi Publishing Corp., v. 2010, p. 6, 2010. Citado na página 24.

CALDIERA, V.; ROMBACH, H. D. The goal question metric approach. *Encyclopedia of software engineering*, v. 2, n. 1994, p. 528–532, 1994. Citado na página 40.

COHORTOR.ORG. *Tuner-gStrings Free*. 2014. Disponível em: <https://play.google.com/store/apps/details?id=org.cohortor.gstrings>. Citado na página 23.

DOZOL. 2014. Disponível em: <http://www.adrianodozol.blogspot.com.br>. Citado na página 69.

DRUYVESTEYN. *Coder for incorporating an auxiliary information signal in a digital audio signal, decoder for recovering such signals from the combined signal, and record carrier having such combined signal recorded thereon*. [S.l.]: Google Patents, 1992. US Patent 5,161,210. Citado na página 33.

DYBÅ, T.; DINGSØYR, T. Empirical studies of agile software development: A systematic review. *Information and software technology*, Elsevier, v. 50, n. 9, p. 833–859, 2008. Citado na página 40.

JüRGEN. *Jürgen*. 2014. Disponível em: <http://www.wi.hs-wismar.de/~cleve/>. Citado na página 37.

KLAPURI, A. P. Automatic music transcription as we know it today. *Journal of New Music Research*, Taylor & Francis, v. 33, n. 3, p. 269–282, 2004. Citado na página 24.

LABGARAGEM. 2014. Disponível em: <http://labdegaragem.com/profiles/blogs/hino-de-times-de-futebol-de-sao-paulo-tocados-pelo-arduino>. Citado na página 31.

MED, B. Teoria da música. 4ª edição revista e ampliada. *Brasília-DF, Musimed*, 1996. Citado 3 vezes nas páginas 29, 30 e 31.

MONSON, I. *Saying something: Jazz improvisation and interaction*. [S.l.]: University of Chicago Press, 2009. Citado na página 23.

MORIN, E.; MATOS, D. *Introdução ao pensamento complexo*. [S.l.]: Sulina Porto Alegre, 2007. Citado 2 vezes nas páginas 34 e 39.

OPPENHEIM, A. V.; WILLSKY, A. S.; NAWAB, S. H. *Signals and systems*. [S.l.]: Prentice-Hall Englewood Cliffs, NJ, 1983. Citado na página 33.

RIBEIRO, M. C. M. O. C. *Criatividade em uma Perspectiva Transdisciplinar*. [S.l.]: Liber Livro, 2014. Citado na página 39.

S, S.; HAYKIN. *Neural networks and learning machines*. [S.l.]: Pearson Education Upper Saddle River, 2009. Citado 2 vezes nas páginas 35 e 36.

SANTOS, M. Caracterização de fontes sonoras e aplicação na auralização de ambientes. Florianópolis, SC, 2008. Citado na página 27.

SHEWHART, W. *Economic Control of Quality of Manufactured Product/50th Anniversary Commemorative Issue. Millwauki: American Society for Quality, 1980.* [S.l.], 1980. Citado na página 40.

THÉBERGE, P. *Any Sound You Can Make: Making Music/Consuming Technology.* [S.l.]: University Press of New England, 1997. Citado na página 23.

TYRANGIEL, J. Auto-tune: Why pop music sounds perfect. *Time Magazine*, p. 1877372–3, 2009. Citado na página 23.

UNSER, M. Sampling-50 years after shannon. *Proceedings of the IEEE*, IEEE, v. 88, n. 4, p. 569–587, 2000. Citado na página 33.

WEISS, D. M. et al. Software product-line engineering: a family-based software development process. Addison-Wesley Professional; Har/Cdr edition, 1999. Citado na página 39.

WÖLFFLIN, H.; JÚNIOR, J. A. *Conceitos fundamentais da história da arte: o problema da evolução dos estilos na arte mais recente.* [S.l.]: Martins Fontes, 2000. Citado na página 29.

Apêndices

APÊNDICE A – Primeiro Apêndice

Esse apêndice diz respeito aos códigos feitos na plataforma Scilab.

A.1 Módulo Principal

```
1  function DA2 = DA2(som)
2      exec get_mono_signal.sci;
3      exec get_fourier_transform.sci;
4      exec get_equalization_signal.sci;
5      exec correlate_with_notes.sci;
6      exec correlate_with_chords.sci;
7      exec interpret_correlation.sci;
8
9      //Load notes in constants variables
10     exec load_notes_constants.sci;
11
12     //Load chords in constants variables
13     exec load_chords_constants.sci;
14
15     //Get one dimensional array of signal
16     som = get_mono_signal(som);
17
18     //Get array with slots that are frequencies
19     respfreq = get_fourier_transform(som);
20     scf(1);
21     plot(respfreq, '-o');
22     xtitle('Resposta em Frequencia', 'Frequencia (Hz)', 'Resposta');
23     mtlb_axis([1, 800, 0, 0.7]);
24
25     //Get signal equalizated with maximum is 1
26     rfeq = get_equalization_signal(respfreq);
27
28     //Correlate frequencies with array of notes
29     scf(2);
30     S1 = correlate_with_notes(rfeq);
31     bar(S1);
32     a=gca();
33     notas = ["do" "do#" "re" "re#" "mi" "fa" "fa#" ...
34     "sol" "sol#" "la" "la#" "si"]
35     notas_slot = [1:12]
36     newTicks= a.x_ticks;
```

```
37      newTicks(2)= notas_slot;
38      newTicks(3)= notas;
39      a.x_ticks=newTicks;
40
41      //Correlate frequencies with array of chords
42      scf(3);
43      S2 = correlate_with_chords(S1);
44      bar(S2);
45      a=gca();
46      notas = ["CM" "Cm" "Caum" "Cdim" "C#M" "C#m" "C#aum" "C#dim" ...
47      "DM" "Dm" "Daum" "Ddim" ...
48      "D#M" "D#m" "D#aum" "D#dim" ...
49      "EM" "Em" "Eaum" "Edim" ...
50      "FM" "Fm" "Faum" "Fdim" ...
51      "F#M" "F#m" "F#aum" "F#dim" ...
52      "GM" "Gm" "Gaum" "Gdim" ...
53      "G#M" "G#m" "G#aum" "G#dim" ...
54      "AM" "Am" "Aaum" "Adim" ...
55      "A#M" "A#m" "A#aum" "A#dim" ...
56      "BM" "Bm" "Baum" "Bdim"]
57      notas_slot = [1:48]
58      newTicks= a.x_ticks;
59      newTicks(2)= notas_slot;
60      newTicks(3)= notas;
61      a.x_ticks=newTicks;
62
63      //Get the chord inferred
64      ACORDETOC = interpret_correlation(S2);
65
66      DA2 = ACORDETOC;
67  endfunction
```

A.2 Módulo de Monoficação do Sinal de Áudio

```
1  function som = get_mono_signal(som)
2      //MONO
3      som = som(1:length(som));
4      som = som/max(som);
5  endfunction
```

A.3 Módulo da Transformada de Fourier

```
function respfreq = get_fourier_transform(som)
    //-----------------------------
    //TRANSFORMADA DE FOURIER
    //tx amostragem
    fs = 44100;
     //vetor das frequencias disponiveis no som
    f = (0:length(som)-1)*fs/length(som);
    //vetor das frequencias propriamente ditas
    freq = f(1:round(length(f)/2));
    //transformada de fourier criando vetor respostas
    SOM = abs(fft(som));
    //normalizacao do vetor das respostas
    SOM = SOM/max(SOM);
    //vetor das respostas propriamente ditas
    SOM = SOM(1:round(length(f)/2));
    //Transformando as respectivas frequencias em slots
    //variavel auxiliar de contagem dos slots do novo
    //vetor resposta-frequencia
    l = 1;
    //variavel auxiliar de contagem das respostas
    //numa mesma faixa de frequencia
    j = 0;
    //variavel auxiliar de contagem dos slots do vetor resposta
    i = 1;
    SOMA = 0;
    while (i<length(freq))//laco da acoplacao
        //se as frequencias arrendondadas vizinhas forem iguais...
        if (round(freq(i)) == round(freq(i+1)))
            //soma das respostas das frequencias parecidas
            SOMA = SOM(i+1) + SOMA;
            //contagem de frequencias parecidas achadas
            j = j + 1;
        else
            //incersao da media das frequencias parecidas no vetor
            respfreq(l) = SOMA/(j+1);
            //zerar a contagem das frequencias parecidas
            //para o proximo conjunto de sequencias
            j = 0;
            SOMA = SOM(i+1);//comecando um novo conjunto de sequencias
            l = l+1;//proximo slot do respfreq
        end
        i = i+1;//proximo slot do vetor SOM
    end
    //fim do laco e zerando os contadores usados
    l = 0; j = 0; i = 0;
    //-----------------------------
endfunction
```

A.4 Módulo de Equalização do Sinal

```
function rfeq = get_equalization_signal(respfreq)
    //EQUALIZACAO
    x = gsort(respfreq);
    x = x(length(respfreq)-3);
    a = (max(respfreq)-min(respfreq))/x;
    b = max(respfreq)-a*x;
    rfeq = (respfreq-b)/a;
endfunction
```

A.5 Módulo de Correlação de Notas

```
function S1 = correlate_with_notes(rfeq)

    exec correlation.sci;

    i = 1; //contador para andar ao longo do vetor

    while (i <= 12)
         correlacao = coeffcorr(rfeq,notas(i,:));
         S1(i) = correlacao;
        //S1(i) = exp(-(norm(rfeq - notas(i,:))*b));
        i = i + 1;
    end

endfunction
```

A.6 Módulo de Correlação

```
function [c]=coeffcorr(X,Y)
    // first check consistency of data, return NAN in case of problem
     [lX,cX]=size(X);
     if (lX == 1)
        X=X';
     sX=cX;
     elseif (cX==1)
```

```
8          sX=lX;
9       else
10         c=%nan;
11      return
12      end
13      [lY,cY]=size(Y);
14      if (lY == 1)
15         Y=Y';
16      sY=cY;
17      elseif (cY==1)
18         sY=lY;
19      else
20          c=%nan;
21      return;
22      end
23      if (sX ~= sY)
24         c=%nan;
25      return;
26      end
27      // here begins the actual computation
28      X=X-mean(X);
29      Y=Y-mean(Y);
30      c=X'*Y;
31      if (c==0)
32      return;
33      else
34         c=c/(norm(X)*norm(Y));
35      end
36   endfunction
```

A.7 Módulo de Correlação de Acordes

```
1  function S2 = correlate_with_chords(S1)
2      exec correlation.sci;
3      i = 1; //contador para andar ao longo do vetor
4      while (i <= 48)
5           correlacao = coeffcorr(S1,BD(:,i)');
6           S2(i) = sqrt(correlacao^2);
7          i = i + 1;
8      end
9  endfunction
```

A.8 Módulo de Interpretação de Correlação dos Acordes

```
1  function ACORDETOC = interpret_correlation(S2)
2
3  acordetoc = find(S2==max(S2));
4
5  if (length(acordetoc) > 1)
6      acordetoc = acordetoc(1);
7  end
8
9  //DECODIFICADOR
10
11 if (acordetoc == 1)
12     ACORDETOC = 'CM';
13 end
14 if (acordetoc == 2)
15     ACORDETOC = 'Cm';
16 end
17 if (acordetoc == 3)
18     ACORDETOC = 'Caum';
19 end
20 if (acordetoc == 4)
21     ACORDETOC = 'Cdim';
22 end
23 if (acordetoc == 5)
24     ACORDETOC = 'C#M';
25 end
26 if (acordetoc == 6)
27     ACORDETOC = 'C#m';
28 end
29 if (acordetoc == 7)
30     ACORDETOC = 'C#aum';
31 end
32 if (acordetoc == 8)
33     ACORDETOC = 'C#dim';
34 end
35 if (acordetoc == 9)
36     ACORDETOC = 'DM';
37 end
38 if (acordetoc == 10)
39     ACORDETOC = 'Dm';
40 end
41 if (acordetoc == 11)
42     ACORDETOC = 'Daum';
43 end
44 if (acordetoc == 12)
```

```
45      ACORDETOC = 'Ddim';
46  end
47  if (acordetoc == 13)
48      ACORDETOC = 'D#M ou EbM';
49  end
50  if (acordetoc == 14)
51      ACORDETOC = 'D#m ou Ebm';
52  end
53  if (acordetoc == 15)
54      ACORDETOC = 'D#aum ou Ebaum';
55  end
56  if (acordetoc == 16)
57      ACORDETOC = 'D#dim ou Ebdim';
58  end
59  if (acordetoc == 17)
60      ACORDETOC = 'EM';
61  end
62  if (acordetoc == 18)
63      ACORDETOC = 'Em';
64  end
65  if (acordetoc == 19)
66      ACORDETOC = 'Eaum';
67  end
68  if (acordetoc == 20)
69      ACORDETOC = 'Edim';
70  end
71  if (acordetoc == 21)
72      ACORDETOC = 'FM';
73  end
74  if (acordetoc == 22)
75      ACORDETOC = 'Fm';
76  end
77  if (acordetoc == 23)
78      ACORDETOC = 'Faum';
79  end
80  if (acordetoc == 24)
81      ACORDETOC = 'Fdim';
82  end
83  if (acordetoc == 25)
84      ACORDETOC = 'F#M';
85  end
86  if (acordetoc == 26)
87      ACORDETOC = 'F#m';
88  end
89  if (acordetoc == 27)
90      ACORDETOC = 'F#aum';
91  end
```

```
92 if (acordetoc == 28)
93     ACORDETOC = 'F#dim';
94 end
95 if (acordetoc == 29)
96     ACORDETOC = 'GM';
97 end
98 if (acordetoc == 30)
99     ACORDETOC = 'Gm';
100 end
101 if (acordetoc == 31)
102     ACORDETOC = 'Gaum';
103 end
104 if (acordetoc == 32)
105     ACORDETOC = 'Gdim';
106 end
107 if (acordetoc == 33)
108     ACORDETOC = 'G#M ou AbM';
109 end
110 if (acordetoc == 34)
111     ACORDETOC = 'G#m ou Abm';
112 end
113 if (acordetoc == 35)
114     ACORDETOC = 'G#aum ou Abaum';
115 end
116 if (acordetoc == 36)
117     ACORDETOC = 'G#dim ou Abdim';
118 end
119 if (acordetoc == 37)
120     ACORDETOC = 'AM';
121 end
122 if (acordetoc == 38)
123     ACORDETOC = 'Am';
124 end
125 if (acordetoc == 39)
126     ACORDETOC = 'Aaum';
127 end
128 if (acordetoc == 40)
129     ACORDETOC = 'Adim';
130 end
131 if (acordetoc == 41)
132     ACORDETOC = 'A#M ou BbM';
133 end
134 if (acordetoc == 42)
135     ACORDETOC = 'A#m ou Bbm';
136 end
137 if (acordetoc == 43)
138     ACORDETOC = 'A#aum ou Bbaum';
```

```
139 end
140 if (acordetoc == 44)
141     ACORDETOC = 'A#dim ou Bbdim';
142 end
143 if (acordetoc == 45)
144     ACORDETOC = 'BM';
145 end
146 if (acordetoc == 46)
147     ACORDETOC = 'Bm';
148 end
149 if (acordetoc == 47)
150     ACORDETOC = 'Baum';
151 end
152 if (acordetoc == 48)
153     ACORDETOC = 'Bdim';
154 end
155
156 endfunction
```

A.9 Módulo de Alocação de Constantes para Notas

```
1  //-----------------------------
2  //NOTAS
3
4  notas(12,22050) = 0; //matriz das notas
5
6  //ganhos
7  g1 = 0.01;
8  g2 = 0.05;
9  g3 = 0.1;
10 g4 = 0.4;
11 g5 = 0.8;
12 g6 = 1;
13
14 //* Linha[0] = Do */
15 notas(1 + 0,60) = g1;
16 notas(1 + 0,61) = g2;
17 notas(1 + 0,62) = g3;
18 notas(1 + 0,63) = g4;
19 notas(1 + 0,64) = g5;
20 notas(1 + 0,65) = g6;
21 notas(1 + 0,66) = g5;
22 notas(1 + 0,67) = g4;
23 notas(1 + 0,68) = g3;
```

```
24 notas(1 + 0,69) = g2;
25 notas(1 + 0,70) = g1;
26
27 notas(1 + 0,126) = g1;
28 notas(1 + 0,127) = g2;
29 notas(1 + 0,128) = g3;
30 notas(1 + 0,129) = g4;
31 notas(1 + 0,130) = g5;
32 notas(1 + 0,131) = g6;
33 notas(1 + 0,132) = g5;
34 notas(1 + 0,133) = g4;
35 notas(1 + 0,134) = g3;
36 notas(1 + 0,135) = g2;
37 notas(1 + 0,136) = g1;
38
39 notas(1 + 0,256) = g1;
40 notas(1 + 0,257) = g2;
41 notas(1 + 0,258) = g3;
42 notas(1 + 0,259) = g4;
43 notas(1 + 0,260) = g5;
44 notas(1 + 0,261) = g6;
45 notas(1 + 0,262) = g5;
46 notas(1 + 0,263) = g4;
47 notas(1 + 0,264) = g3;
48 notas(1 + 0,265) = g2;
49 notas(1 + 0,266) = g1;
50
51 notas(1 + 0,518) = g1;
52 notas(1 + 0,519) = g2;
53 notas(1 + 0,520) = g3;
54 notas(1 + 0,521) = g4;
55 notas(1 + 0,522) = g5;
56 notas(1 + 0,523) = g6;
57 notas(1 + 0,524) = g5;
58 notas(1 + 0,525) = g4;
59 notas(1 + 0,526) = g3;
60 notas(1 + 0,527) = g2;
61 notas(1 + 0,528) = g1;
62
63 notas(1 + 0,1041) = g1;
64 notas(1 + 0,1042) = g2;
65 notas(1 + 0,1043) = g3;
66 notas(1 + 0,1044) = g4;
67 notas(1 + 0,1045) = g5;
68 notas(1 + 0,1046) = g6;
69 notas(1 + 0,1047) = g5;
70 notas(1 + 0,1048) = g4;
```

```
71 notas(1 + 0,1049) = g3;
72 notas(1 + 0,1050) = g2;
73 notas(1 + 0,1051) = g1;
74
75 //* Linha[1] = Do# */
76 notas(1 + 1,64) = g1;
77 notas(1 + 1,65) = g2;
78 notas(1 + 1,66) = g3;
79 notas(1 + 1,67) = g4;
80 notas(1 + 1,68) = g5;
81 notas(1 + 1,69) = g6;
82 notas(1 + 1,70) = g5;
83 notas(1 + 1,71) = g4;
84 notas(1 + 1,72) = g3;
85 notas(1 + 1,73) = g2;
86 notas(1 + 1,74) = g1;
87
88 notas(1 + 1,133) = g1;
89 notas(1 + 1,134) = g2;
90 notas(1 + 1,135) = g3;
91 notas(1 + 1,136) = g4;
92 notas(1 + 1,137) = g5;
93 notas(1 + 1,138) = g6;
94 notas(1 + 1,139) = g5;
95 notas(1 + 1,140) = g4;
96 notas(1 + 1,141) = g3;
97 notas(1 + 1,142) = g2;
98 notas(1 + 1,143) = g1;
99
100 notas(1 + 1,272) = g1;
101 notas(1 + 1,273) = g2;
102 notas(1 + 1,274) = g3;
103 notas(1 + 1,275) = g4;
104 notas(1 + 1,276) = g5;
105 notas(1 + 1,277) = g6;
106 notas(1 + 1,278) = g5;
107 notas(1 + 1,279) = g4;
108 notas(1 + 1,280) = g3;
109 notas(1 + 1,281) = g2;
110 notas(1 + 1,282) = g1;
111
112 notas(1 + 1,549) = g1;
113 notas(1 + 1,550) = g2;
114 notas(1 + 1,551) = g3;
115 notas(1 + 1,552) = g4;
116 notas(1 + 1,553) = g5;
117 notas(1 + 1,554) = g6;
```

```
118 notas(1 + 1,555) = g5;
119 notas(1 + 1,556) = g4;
120 notas(1 + 1,557) = g3;
121 notas(1 + 1,558) = g2;
122 notas(1 + 1,559) = g1;
123
124 notas(1 + 1,1104) = g1;
125 notas(1 + 1,1105) = g2;
126 notas(1 + 1,1106) = g3;
127 notas(1 + 1,1107) = g4;
128 notas(1 + 1,1108) = g5;
129 notas(1 + 1,1109) = g6;
130 notas(1 + 1,1110) = g5;
131 notas(1 + 1,1111) = g4;
132 notas(1 + 1,1112) = g3;
133 notas(1 + 1,1113) = g2;
134 notas(1 + 1,1114) = g1;
135
136 //* Linha[2] = Re */
137 notas(1 + 2,68) = g1;
138 notas(1 + 2,69) = g2;
139 notas(1 + 2,70) = g3;
140 notas(1 + 2,71) = g4;
141 notas(1 + 2,72) = g5;
142 notas(1 + 2,73) = g6;
143 notas(1 + 2,74) = g5;
144 notas(1 + 2,75) = g4;
145 notas(1 + 2,76) = g3;
146 notas(1 + 2,77) = g2;
147 notas(1 + 2,78) = g1;
148
149 notas(1 + 2,142) = g1;
150 notas(1 + 2,143) = g2;
151 notas(1 + 2,144) = g3;
152 notas(1 + 2,145) = g4;
153 notas(1 + 2,146) = g5;
154 notas(1 + 2,147) = g6;
155 notas(1 + 2,148) = g5;
156 notas(1 + 2,149) = g4;
157 notas(1 + 2,150) = g3;
158 notas(1 + 2,151) = g2;
159 notas(1 + 2,152) = g1;
160
161 notas(1 + 2,289) = g1;
162 notas(1 + 2,290) = g2;
163 notas(1 + 2,291) = g3;
164 notas(1 + 2,292) = g4;
```

```
165 notas(1 + 2,293) = g5;
166 notas(1 + 2,294) = g6;
167 notas(1 + 2,295) = g5;
168 notas(1 + 2,296) = g4;
169 notas(1 + 2,297) = g3;
170 notas(1 + 2,298) = g2;
171 notas(1 + 2,299) = g1;
172
173 notas(1 + 2,582) = g1;
174 notas(1 + 2,583) = g2;
175 notas(1 + 2,584) = g3;
176 notas(1 + 2,585) = g4;
177 notas(1 + 2,586) = g5;
178 notas(1 + 2,587) = g6;
179 notas(1 + 2,588) = g5;
180 notas(1 + 2,589) = g4;
181 notas(1 + 2,590) = g3;
182 notas(1 + 2,591) = g2;
183 notas(1 + 2,592) = g1;
184
185 notas(1 + 2,1169) = g1;
186 notas(1 + 2,1170) = g2;
187 notas(1 + 2,1171) = g3;
188 notas(1 + 2,1172) = g4;
189 notas(1 + 2,1173) = g5;
190 notas(1 + 2,1174) = g6;
191 notas(1 + 2,1175) = g5;
192 notas(1 + 2,1176) = g4;
193 notas(1 + 2,1177) = g3;
194 notas(1 + 2,1178) = g2;
195 notas(1 + 2,1179) = g1;
196
197 //* Linha[3] = Re# */
198 notas(1 + 3,73) = g1;
199 notas(1 + 3,74) = g2;
200 notas(1 + 3,75) = g3;
201 notas(1 + 3,76) = g4;
202 notas(1 + 3,77) = g5;
203 notas(1 + 3,78) = g6;
204 notas(1 + 3,79) = g5;
205 notas(1 + 3,80) = g4;
206 notas(1 + 3,81) = g3;
207 notas(1 + 3,82) = g2;
208 notas(1 + 3,83) = g1;
209
210 notas(1 + 3,140) = g1;
211 notas(1 + 3,141) = g2;
```

```
212 notas(1 + 3,152) = g3;
213 notas(1 + 3,153) = g4;
214 notas(1 + 3,154) = g5;
215 notas(1 + 3,155) = g6;
216 notas(1 + 3,156) = g5;
217 notas(1 + 3,157) = g4;
218 notas(1 + 3,158) = g3;
219 notas(1 + 3,159) = g2;
220 notas(1 + 3,160) = g1;
221
222 notas(1 + 3,306) = g1;
223 notas(1 + 3,307) = g2;
224 notas(1 + 3,308) = g3;
225 notas(1 + 3,309) = g4;
226 notas(1 + 3,310) = g5;
227 notas(1 + 3,311) = g6;
228 notas(1 + 3,312) = g5;
229 notas(1 + 3,313) = g4;
230 notas(1 + 3,314) = g3;
231 notas(1 + 3,315) = g2;
232 notas(1 + 3,316) = g1;
233
234 notas(1 + 3,617) = g1;
235 notas(1 + 3,618) = g2;
236 notas(1 + 3,619) = g3;
237 notas(1 + 3,620) = g4;
238 notas(1 + 3,621) = g5;
239 notas(1 + 3,622) = g6;
240 notas(1 + 3,623) = g5;
241 notas(1 + 3,624) = g4;
242 notas(1 + 3,625) = g3;
243 notas(1 + 3,626) = g2;
244 notas(1 + 3,627) = g1;
245
246 notas(1 + 3,1240) = g1;
247 notas(1 + 3,1241) = g2;
248 notas(1 + 3,1242) = g3;
249 notas(1 + 3,1243) = g4;
250 notas(1 + 3,1244) = g5;
251 notas(1 + 3,1245) = g6;
252 notas(1 + 3,1246) = g5;
253 notas(1 + 3,1247) = g4;
254 notas(1 + 3,1248) = g3;
255 notas(1 + 3,1249) = g2;
256 notas(1 + 3,1250) = g1;
257
258 //* Linha[4] = Mi */
```

```
259 notas(1 + 4,77) = g1;
260 notas(1 + 4,78) = g2;
261 notas(1 + 4,79) = g3;
262 notas(1 + 4,80) = g4;
263 notas(1 + 4,81) = g5;
264 notas(1 + 4,82) = g6;
265 notas(1 + 4,83) = g5;
266 notas(1 + 4,84) = g4;
267 notas(1 + 4,85) = g3;
268 notas(1 + 4,86) = g2;
269 notas(1 + 4,87) = g1;
270
271 notas(1 + 4,160) = g1;
272 notas(1 + 4,161) = g2;
273 notas(1 + 4,162) = g3;
274 notas(1 + 4,163) = g4;
275 notas(1 + 4,164) = g5;
276 notas(1 + 4,165) = g6;
277 notas(1 + 4,166) = g5;
278 notas(1 + 4,167) = g4;
279 notas(1 + 4,168) = g3;
280 notas(1 + 4,169) = g2;
281 notas(1 + 4,170) = g1;
282
283 notas(1 + 4,325) = g1;
284 notas(1 + 4,326) = g2;
285 notas(1 + 4,327) = g3;
286 notas(1 + 4,328) = g4;
287 notas(1 + 4,329) = g5;
288 notas(1 + 4,330) = g6;
289 notas(1 + 4,331) = g5;
290 notas(1 + 4,332) = g4;
291 notas(1 + 4,333) = g3;
292 notas(1 + 4,334) = g2;
293 notas(1 + 4,335) = g1;
294
295 notas(1 + 4,655) = g1;
296 notas(1 + 4,656) = g2;
297 notas(1 + 4,657) = g3;
298 notas(1 + 4,658) = g4;
299 notas(1 + 4,659) = g5;
300 notas(1 + 4,660) = g6;
301 notas(1 + 4,661) = g5;
302 notas(1 + 4,662) = g4;
303 notas(1 + 4,663) = g3;
304 notas(1 + 4,664) = g2;
305 notas(1 + 4,665) = g1;
```

```
306
307 notas(1 + 4,1313) = g1;
308 notas(1 + 4,1314) = g2;
309 notas(1 + 4,1315) = g3;
310 notas(1 + 4,1316) = g4;
311 notas(1 + 4,1317) = g5;
312 notas(1 + 4,1318) = g6;
313 notas(1 + 4,1319) = g5;
314 notas(1 + 4,1320) = g4;
315 notas(1 + 4,1321) = g3;
316 notas(1 + 4,1322) = g2;
317 notas(1 + 4,1323) = g1;
318
319 //* Linha[5] = Fa */
320 notas(1 + 5,82) = g1;
321 notas(1 + 5,83) = g2;
322 notas(1 + 5,84) = g3;
323 notas(1 + 5,85) = g4;
324 notas(1 + 5,86) = g5;
325 notas(1 + 5,87) = g6;
326 notas(1 + 5,88) = g5;
327 notas(1 + 5,89) = g4;
328 notas(1 + 5,90) = g3;
329 notas(1 + 5,91) = g2;
330 notas(1 + 5,92) = g1;
331
332 notas(1 + 5,170) = g1;
333 notas(1 + 5,171) = g2;
334 notas(1 + 5,172) = g3;
335 notas(1 + 5,173) = g4;
336 notas(1 + 5,174) = g5;
337 notas(1 + 5,175) = g6;
338 notas(1 + 5,176) = g5;
339 notas(1 + 5,177) = g4;
340 notas(1 + 5,178) = g3;
341 notas(1 + 5,189) = g2;
342 notas(1 + 5,190) = g1;
343
344 notas(1 + 5,344) = g1;
345 notas(1 + 5,345) = g2;
346 notas(1 + 5,346) = g3;
347 notas(1 + 5,347) = g4;
348 notas(1 + 5,348) = g5;
349 notas(1 + 5,349) = g6;
350 notas(1 + 5,350) = g5;
351 notas(1 + 5,351) = g4;
352 notas(1 + 5,352) = g3;
```

```
353 notas(1 + 5,353) = g2;
354 notas(1 + 5,354) = g1;
355
356 notas(1 + 5,693) = g1;
357 notas(1 + 5,694) = g2;
358 notas(1 + 5,695) = g3;
359 notas(1 + 5,696) = g4;
360 notas(1 + 5,697) = g5;
361 notas(1 + 5,698) = g6;
362 notas(1 + 5,699) = g5;
363 notas(1 + 5,700) = g4;
364 notas(1 + 5,701) = g3;
365 notas(1 + 5,702) = g2;
366 notas(1 + 5,703) = g1;
367
368 notas(1 + 5,1392) = g1;
369 notas(1 + 5,1393) = g2;
370 notas(1 + 5,1394) = g3;
371 notas(1 + 5,1395) = g4;
372 notas(1 + 5,1396) = g5;
373 notas(1 + 5,1397) = g6;
374 notas(1 + 5,1398) = g5;
375 notas(1 + 5,1399) = g4;
376 notas(1 + 5,1400) = g3;
377 notas(1 + 5,1401) = g2;
378 notas(1 + 5,1402) = g1;
379
380 //* Linha[6] = Fa# */
381 notas(1 + 6,88) = g1;
382 notas(1 + 6,89) = g2;
383 notas(1 + 6,90) = g3;
384 notas(1 + 6,91) = g4;
385 notas(1 + 6,92) = g5;
386 notas(1 + 6,93) = g6;
387 notas(1 + 6,94) = g5;
388 notas(1 + 6,95) = g4;
389 notas(1 + 6,96) = g3;
390 notas(1 + 6,97) = g2;
391 notas(1 + 6,98) = g1;
392
393 notas(1 + 6,180) = g1;
394 notas(1 + 6,181) = g2;
395 notas(1 + 6,182) = g3;
396 notas(1 + 6,183) = g4;
397 notas(1 + 6,184) = g5;
398 notas(1 + 6,185) = g6;
399 notas(1 + 6,186) = g5;
```

```
400 notas(1 + 6,187) = g4;
401 notas(1 + 6,188) = g3;
402 notas(1 + 6,189) = g2;
403 notas(1 + 6,190) = g1;
404
405 notas(1 + 6,365) = g1;
406 notas(1 + 6,366) = g2;
407 notas(1 + 6,367) = g3;
408 notas(1 + 6,368) = g4;
409 notas(1 + 6,369) = g5;
410 notas(1 + 6,370) = g6;
411 notas(1 + 6,371) = g5;
412 notas(1 + 6,372) = g4;
413 notas(1 + 6,373) = g3;
414 notas(1 + 6,374) = g2;
415 notas(1 + 6,375) = g1;
416
417 notas(1 + 6,735) = g1;
418 notas(1 + 6,736) = g2;
419 notas(1 + 6,737) = g3;
420 notas(1 + 6,738) = g4;
421 notas(1 + 6,739) = g5;
422 notas(1 + 6,740) = g6;
423 notas(1 + 6,741) = g5;
424 notas(1 + 6,742) = g4;
425 notas(1 + 6,743) = g3;
426 notas(1 + 6,744) = g2;
427 notas(1 + 6,745) = g1;
428
429 notas(1 + 6,1475) = g1;
430 notas(1 + 6,1476) = g2;
431 notas(1 + 6,1477) = g3;
432 notas(1 + 6,1478) = g4;
433 notas(1 + 6,1479) = g5;
434 notas(1 + 6,1480) = g6;
435 notas(1 + 6,1481) = g5;
436 notas(1 + 6,1482) = g4;
437 notas(1 + 6,1483) = g3;
438 notas(1 + 6,1484) = g2;
439 notas(1 + 6,1485) = g1;
440
441 //* Linha[7] = Sol */
442 notas(1 + 7,93) = g1;
443 notas(1 + 7,94) = g2;
444 notas(1 + 7,95) = g3;
445 notas(1 + 7,96) = g4;
446 notas(1 + 7,97) = g5;
```

```
447 notas(1 + 7,98) = g6;
448 notas(1 + 7,99) = g5;
449 notas(1 + 7,100) = g4;
450 notas(1 + 7,101) = g3;
451 notas(1 + 7,102) = g2;
452 notas(1 + 7,103) = g1;
453
454 notas(1 + 7,191) = g1;
455 notas(1 + 7,192) = g2;
456 notas(1 + 7,193) = g3;
457 notas(1 + 7,194) = g4;
458 notas(1 + 7,195) = g5;
459 notas(1 + 7,196) = g6;
460 notas(1 + 7,197) = g5;
461 notas(1 + 7,198) = g4;
462 notas(1 + 7,199) = g3;
463 notas(1 + 7,200) = g2;
464 notas(1 + 7,201) = g1;
465
466 notas(1 + 7,387) = g1;
467 notas(1 + 7,388) = g2;
468 notas(1 + 7,389) = g3;
469 notas(1 + 7,390) = g4;
470 notas(1 + 7,391) = g5;
471 notas(1 + 7,392) = g6;
472 notas(1 + 7,393) = g5;
473 notas(1 + 7,394) = g4;
474 notas(1 + 7,395) = g3;
475 notas(1 + 7,396) = g2;
476 notas(1 + 7,397) = g1;
477
478 notas(1 + 7,779) = g1;
479 notas(1 + 7,780) = g2;
480 notas(1 + 7,781) = g3;
481 notas(1 + 7,782) = g4;
482 notas(1 + 7,783) = g5;
483 notas(1 + 7,784) = g6;
484 notas(1 + 7,785) = g5;
485 notas(1 + 7,786) = g4;
486 notas(1 + 7,787) = g3;
487 notas(1 + 7,788) = g2;
488 notas(1 + 7,789) = g1;
489
490 notas(1 + 7,1563) = g1;
491 notas(1 + 7,1564) = g2;
492 notas(1 + 7,1565) = g3;
493 notas(1 + 7,1566) = g4;
```

```
494 notas(1 + 7,1567) = g5;
495 notas(1 + 7,1568) = g6;
496 notas(1 + 7,1569) = g5;
497 notas(1 + 7,1570) = g4;
498 notas(1 + 7,1571) = g3;
499 notas(1 + 7,1572) = g2;
500 notas(1 + 7,1573) = g1;
501
502 //* Linha[8] = Sol# */
503 notas(1 + 8,101) = g1;
504 notas(1 + 8,100) = g2;
505 notas(1 + 8,101) = g3;
506 notas(1 + 8,102) = g4;
507 notas(1 + 8,103) = g5;
508 notas(1 + 8,104) = g6;
509 notas(1 + 8,105) = g5;
510 notas(1 + 8,106) = g4;
511 notas(1 + 8,107) = g3;
512 notas(1 + 8,108) = g2;
513 notas(1 + 8,109) = g1;
514
515 notas(1 + 8,203) = g1;
516 notas(1 + 8,204) = g2;
517 notas(1 + 8,205) = g3;
518 notas(1 + 8,206) = g4;
519 notas(1 + 8,207) = g5;
520 notas(1 + 8,208) = g6;
521 notas(1 + 8,209) = g5;
522 notas(1 + 8,210) = g4;
523 notas(1 + 8,211) = g3;
524 notas(1 + 8,212) = g2;
525 notas(1 + 8,213) = g1;
526
527 notas(1 + 8,410) = g1;
528 notas(1 + 8,411) = g2;
529 notas(1 + 8,412) = g3;
530 notas(1 + 8,413) = g4;
531 notas(1 + 8,414) = g5;
532 notas(1 + 8,415) = g6;
533 notas(1 + 8,416) = g5;
534 notas(1 + 8,417) = g4;
535 notas(1 + 8,418) = g3;
536 notas(1 + 8,419) = g2;
537 notas(1 + 8,420) = g1;
538
539 notas(1 + 8,825) = g1;
540 notas(1 + 8,826) = g2;
```

```
541 notas(1 + 8,827) = g3;
542 notas(1 + 8,828) = g4;
543 notas(1 + 8,829) = g5;
544 notas(1 + 8,830) = g6;
545 notas(1 + 8,831) = g5;
546 notas(1 + 8,832) = g4;
547 notas(1 + 8,833) = g3;
548 notas(1 + 8,834) = g2;
549 notas(1 + 8,835) = g1;
550
551 notas(1 + 8,1656) = g1;
552 notas(1 + 8,1657) = g2;
553 notas(1 + 8,1658) = g3;
554 notas(1 + 8,1659) = g4;
555 notas(1 + 8,1660) = g5;
556 notas(1 + 8,1661) = g6;
557 notas(1 + 8,1662) = g5;
558 notas(1 + 8,1663) = g4;
559 notas(1 + 8,1664) = g3;
560 notas(1 + 8,1665) = g2;
561 notas(1 + 8,1666) = g1;
562
563 //* Linha[9] = La */
564 notas(1 + 9,105) = g1;
565 notas(1 + 9,106) = g2;
566 notas(1 + 9,107) = g3;
567 notas(1 + 9,108) = g4;
568 notas(1 + 9,109) = g5;
569 notas(1 + 9,110) = g6;
570 notas(1 + 9,111) = g5;
571 notas(1 + 9,112) = g4;
572 notas(1 + 9,113) = g3;
573 notas(1 + 9,114) = g2;
574 notas(1 + 9,115) = g1;
575
576 notas(1 + 9,215) = g1;
577 notas(1 + 9,216) = g2;
578 notas(1 + 9,217) = g3;
579 notas(1 + 9,218) = g4;
580 notas(1 + 9,219) = g5;
581 notas(1 + 9,220) = g6;
582 notas(1 + 9,221) = g5;
583 notas(1 + 9,222) = g4;
584 notas(1 + 9,223) = g3;
585 notas(1 + 9,224) = g2;
586 notas(1 + 9,225) = g1;
587
```

```
588 notas(1 + 9,435) = g1;
589 notas(1 + 9,436) = g2;
590 notas(1 + 9,437) = g3;
591 notas(1 + 9,438) = g4;
592 notas(1 + 9,439) = g5;
593 notas(1 + 9,440) = g6;
594 notas(1 + 9,441) = g5;
595 notas(1 + 9,442) = g4;
596 notas(1 + 9,443) = g3;
597 notas(1 + 9,444) = g2;
598 notas(1 + 9,445) = g1;
599
600 notas(1 + 9,875) = g1;
601 notas(1 + 9,876) = g2;
602 notas(1 + 9,877) = g3;
603 notas(1 + 9,878) = g4;
604 notas(1 + 9,879) = g5;
605 notas(1 + 9,880) = g6;
606 notas(1 + 9,881) = g5;
607 notas(1 + 9,882) = g4;
608 notas(1 + 9,883) = g3;
609 notas(1 + 9,884) = g2;
610 notas(1 + 9,885) = g1;
611
612 notas(1 + 9,1755) = g1;
613 notas(1 + 9,1756) = g2;
614 notas(1 + 9,1757) = g3;
615 notas(1 + 9,1758) = g4;
616 notas(1 + 9,1759) = g5;
617 notas(1 + 9,1760) = g6;
618 notas(1 + 9,1761) = g5;
619 notas(1 + 9,1762) = g4;
620 notas(1 + 9,1763) = g3;
621 notas(1 + 9,1764) = g2;
622 notas(1 + 9,1765) = g1;
623
624 //* Linha[10] = La# */
625 notas(1 + 10,111) = g1;
626 notas(1 + 10,112) = g2;
627 notas(1 + 10,113) = g3;
628 notas(1 + 10,114) = g4;
629 notas(1 + 10,115) = g5;
630 notas(1 + 10,116) = g6;
631 notas(1 + 10,117) = g5;
632 notas(1 + 10,118) = g4;
633 notas(1 + 10,119) = g3;
634 notas(1 + 10,120) = g2;
```

```
635 notas(1 + 10,121) = g1;
636
637 notas(1 + 10,228) = g1;
638 notas(1 + 10,229) = g2;
639 notas(1 + 10,230) = g3;
640 notas(1 + 10,231) = g4;
641 notas(1 + 10,232) = g5;
642 notas(1 + 10,233) = g6;
643 notas(1 + 10,234) = g5;
644 notas(1 + 10,235) = g4;
645 notas(1 + 10,236) = g3;
646 notas(1 + 10,237) = g2;
647 notas(1 + 10,238) = g1;
648
649 notas(1 + 10,461) = g1;
650 notas(1 + 10,462) = g2;
651 notas(1 + 10,463) = g3;
652 notas(1 + 10,464) = g4;
653 notas(1 + 10,465) = g5;
654 notas(1 + 10,466) = g6;
655 notas(1 + 10,467) = g5;
656 notas(1 + 10,468) = g4;
657 notas(1 + 10,469) = g3;
658 notas(1 + 10,470) = g2;
659 notas(1 + 10,471) = g1;
660
661 notas(1 + 10,927) = g1;
662 notas(1 + 10,928) = g2;
663 notas(1 + 10,929) = g3;
664 notas(1 + 10,930) = g4;
665 notas(1 + 10,931) = g5;
666 notas(1 + 10,932) = g6;
667 notas(1 + 10,933) = g5;
668 notas(1 + 10,934) = g4;
669 notas(1 + 10,935) = g3;
670 notas(1 + 10,936) = g2;
671 notas(1 + 10,937) = g1;
672
673 notas(1 + 10,1859) = g1;
674 notas(1 + 10,1860) = g2;
675 notas(1 + 10,1861) = g3;
676 notas(1 + 10,1862) = g4;
677 notas(1 + 10,1863) = g5;
678 notas(1 + 10,1864) = g6;
679 notas(1 + 10,1865) = g5;
680 notas(1 + 10,1866) = g4;
681 notas(1 + 10,1867) = g3;
```

```
682 notas(1 + 10,1868) = g2;
683 notas(1 + 10,1869) = g1;
684
685 //* Linha[11] = Si */
686 notas(1 + 11,119) = g1;
687 notas(1 + 11,120) = g2;
688 notas(1 + 11,121) = g3;
689 notas(1 + 11,122) = g4;
690 notas(1 + 11,123) = g5;
691 notas(1 + 11,124) = g6;
692 notas(1 + 11,125) = g5;
693 notas(1 + 11,126) = g4;
694 notas(1 + 11,127) = g3;
695 notas(1 + 11,128) = g2;
696 notas(1 + 11,129) = g1;
697
698 notas(1 + 11,242) = g1;
699 notas(1 + 11,243) = g2;
700 notas(1 + 11,244) = g3;
701 notas(1 + 11,245) = g4;
702 notas(1 + 11,246) = g5;
703 notas(1 + 11,247) = g6;
704 notas(1 + 11,248) = g5;
705 notas(1 + 11,249) = g4;
706 notas(1 + 11,250) = g3;
707 notas(1 + 11,251) = g2;
708 notas(1 + 11,252) = g1;
709
710 notas(1 + 11,489) = g1;
711 notas(1 + 11,490) = g2;
712 notas(1 + 11,491) = g3;
713 notas(1 + 11,492) = g4;
714 notas(1 + 11,493) = g5;
715 notas(1 + 11,494) = g6;
716 notas(1 + 11,495) = g5;
717 notas(1 + 11,496) = g4;
718 notas(1 + 11,497) = g3;
719 notas(1 + 11,498) = g2;
720 notas(1 + 11,499) = g1;
721
722 notas(1 + 11,983) = g1;
723 notas(1 + 11,984) = g2;
724 notas(1 + 11,985) = g3;
725 notas(1 + 11,986) = g4;
726 notas(1 + 11,987) = g5;
727 notas(1 + 11,988) = g6;
728 notas(1 + 11,989) = g5;
```

```
729 notas(1 + 11,990) = g4;
730 notas(1 + 11,991) = g3;
731 notas(1 + 11,992) = g2;
732 notas(1 + 11,993) = g1;
733
734 notas(1 + 11,1971) = g1;
735 notas(1 + 11,1972) = g2;
736 notas(1 + 11,1973) = g3;
737 notas(1 + 11,1974) = g4;
738 notas(1 + 11,1975) = g5;
739 notas(1 + 11,1976) = g6;
740 notas(1 + 11,1977) = g5;
741 notas(1 + 11,1978) = g4;
742 notas(1 + 11,1979) = g3;
743 notas(1 + 11,1980) = g2;
744 notas(1 + 11,1981) = g1;
```

A.10 Módulo de Alocação de Constantes para Acordes

```
1
2 //-------------------------------
3 BD(12,48) = 0; // inicializando o banco de dados para acordes
4 //-------------------------------
5
6 afin1 = 0; afin2 = 0;
7
8 //C
9 //CM
10 BD(12,1) = afin1;
11 BD(1,1) = 1; //baixo
12 BD(2,1) = afin2;
13 BD(4,1) = afin1;
14 BD(5,1) = 1; //terca
15 BD(6,1) = afin2;
16 BD(7,1) = afin1;
17 BD(8,1) = 1; //quinta
18 BD(9,1) = afin2;
19 //Cm
20 BD(12,2) = afin1;
21 BD(1,2) = 1; //baixo
22 BD(2,2) = afin2;
23 BD(3,2) = afin1;
24 BD(4,2) = 1; //terca
25 BD(5,2) = afin2;
```

```
26 BD(7,2) = afin1;
27 BD(8,2) = 1; //quinta
28 BD(9,2) = afin2;
29 //Caum
30 BD(12,3) = afin1;
31 BD(1,3) = 1; //baixo
32 BD(2,3) = afin2;
33 BD(4,3) = afin1;
34 BD(5,3) = 1; //terca
35 BD(6,3) = afin2;
36 BD(8,3) = afin1;
37 BD(9,3) = 1; //quinta
38 BD(10,3) = afin2;
39 //Cdim
40 BD(12,4) = afin1;
41 BD(1,4) = 1; //baixo
42 BD(2,4) = afin2;
43 BD(3,4) = afin1;
44 BD(4,4) = 1; //terca
45 BD(5,4) = afin2;
46 BD(6,4) = afin1;
47 BD(7,4) = 1; //quinta
48 BD(8,4) = afin2;
49
50 //C#
51 //C#M
52 BD(1,5) = afin1;
53 BD(2,5) = 1; //baixo
54 BD(3,5) = afin2;
55 BD(5,5) = afin1;
56 BD(6,5) = 1; //terca
57 BD(7,5) = afin2;
58 BD(8,5) = afin1;
59 BD(9,5) = 1; //quinta
60 BD(10,5) = afin2;
61 //C#m
62 BD(1,6) = afin1;
63 BD(2,6) = 1; //baixo
64 BD(3,6) = afin2;
65 BD(4,6) = afin1;
66 BD(5,6) = 1; //terca
67 BD(6,6) = afin2;
68 BD(8,6) = afin1;
69 BD(9,6) = 1; //quinta
70 BD(10,6) = afin2;
71 //C#aum
72 BD(1,7) = afin1;
```

```
73  BD(2,7) = 1; //baixo
74  BD(3,7) = afin2;
75  BD(5,7) = afin1;
76  BD(6,7) = 1; //terca
77  BD(7,7) = afin2;
78  BD(9,7) = afin1;
79  BD(10,7) = 1; //quinta
80  BD(11,7) = afin2;
81  //C#dim
82  BD(1,8) = afin1;
83  BD(2,8) = 1; //baixo
84  BD(3,8) = afin2;
85  BD(4,8) = afin1;
86  BD(5,8) = 1; //terca
87  BD(6,8) = afin2;
88  BD(7,8) = afin1;
89  BD(8,8) = 1; //quinta
90  BD(9,8) = afin2;
91
92  //D
93  //DM
94  BD(2,9) = afin1;
95  BD(3,9) = 1; //baixo
96  BD(4,9) = afin2;
97  BD(6,9) = afin1;
98  BD(7,9) = 1; //terca
99  BD(8,9) = afin2;
100 BD(9,9) = afin1;
101 BD(10,9) = 1; //quinta
102 BD(11,9) = afin2;
103 //Dm
104 BD(2,10) = afin1;
105 BD(3,10) = 1; //baixo
106 BD(4,10) = afin2;
107 BD(5,10) = afin1;
108 BD(6,10) = 1; //terca
109 BD(7,10) = afin2;
110 BD(9,10) = afin1;
111 BD(10,10) = 1; //quinta
112 BD(11,10) = afin2;
113 //Daum
114 BD(2,11) = afin1;
115 BD(3,11) = 1; //baixo
116 BD(4,11) = afin2;
117 BD(6,11) = afin1;
118 BD(7,11) = 1; //terca
119 BD(8,11) = afin2;
```

```
120 BD(10,11) = afin1;
121 BD(11,11) = 1; //quinta
122 BD(11,11) = afin2;
123 //Ddim
124 BD(2,12) = afin1;
125 BD(3,12) = 1; //baixo
126 BD(4,12) = afin2;
127 BD(5,12) = afin1;
128 BD(6,12) = 1; //terca
129 BD(7,12) = afin2;
130 BD(8,12) = afin1;
131 BD(9,12) = 1; //quinta
132 BD(10,12) = afin2;
133
134 //D#
135 //D#M
136 BD(3,13) = afin1;
137 BD(4,13) = 1; //baixo
138 BD(5,13) = afin2;
139 BD(7,13) = afin1;
140 BD(8,13) = 1; //terca
141 BD(9,13) = afin2;
142 BD(10,13) = afin1;
143 BD(11,13) = 1; //quinta
144 BD(12,13) = afin2;
145 //D#m
146 BD(3,14) = afin1;
147 BD(4,14) = 1; //baixo
148 BD(5,14) = afin2;
149 BD(6,14) = afin1;
150 BD(7,14) = 1; //terca
151 BD(8,14) = afin2;
152 BD(10,14) = afin1;
153 BD(11,14) = 1; //quinta
154 BD(12,14) = afin2;
155 //D#aum
156 BD(3,15) = afin1;
157 BD(4,15) = 1; //baixo
158 BD(5,15) = afin2;
159 BD(7,15) = afin1;
160 BD(8,15) = 1; //terca
161 BD(9,15) = afin2;
162 BD(11,15) = afin1;
163 BD(12,15) = 1; //quinta
164 BD(1,15) = afin2;
165 //D#dim
166 BD(3,16) = afin1;
```

```
167 BD(4,16) = 1; //baixo
168 BD(5,16) = afin2;
169 BD(6,16) = afin1;
170 BD(7,16) = 1; //terca
171 BD(8,16) = afin2;
172 BD(9,16) = afin1;
173 BD(10,16) = 1; //quinta
174 BD(11,16) = afin2;
175
176 //E
177 //EM
178 BD(4,17) = afin1;
179 BD(5,17) = 1; //baixo
180 BD(6,17) = afin2;
181 BD(8,17) = afin1;
182 BD(9,17) = 1; //terca
183 BD(10,17) = afin2;
184 BD(11,17) = afin1;
185 BD(12,17) = 1; //quinta
186 BD(1,17) = afin2;
187 //Em
188 BD(4,18) = afin1;
189 BD(5,18) = 1; //baixo
190 BD(6,18) = afin2;
191 BD(7,18) = afin1;
192 BD(8,18) = 1; //terca
193 BD(9,18) = afin2;
194 BD(11,18) = afin1;
195 BD(12,18) = 1; //quinta
196 BD(1,18) = afin2;
197 //Eaum
198 BD(4,19) = afin1;
199 BD(5,19) = 1; //baixo
200 BD(6,19) = afin2;
201 BD(8,19) = afin1;
202 BD(9,19) = 1; //terca
203 BD(10,19) = afin2;
204 BD(12,19) = afin1;
205 BD(1,19) = 1; //quinta
206 BD(2,19) = afin2;
207 //Edim
208 BD(4,20) = afin1;
209 BD(5,20) = 1; //baixo
210 BD(6,20) = afin2;
211 BD(7,20) = afin1;
212 BD(8,20) = 1; //terca
213 BD(9,20) = afin2;
```

```
214 BD(10,20) = afin1;
215 BD(11,20) = 1; //quinta
216 BD(12,20) = afin2;
217
218 //F
219 //FM
220 BD(5,21) = afin1;
221 BD(6,21) = 1; //tonica
222 BD(7,21) = afin2;
223 BD(9,21) = afin1;
224 BD(10,21) = 1; //terca
225 BD(11,21) = afin2;
226 BD(12,21) = afin1;
227 BD(1,21) = 1; //quinta
228 BD(2,21) = afin2;
229 //Fm
230 BD(5,22) = afin1;
231 BD(6,22) = 1;
232 BD(7,22) = afin2;
233 BD(8,22) = afin1;
234 BD(9,22) = 1;
235 BD(10,22) = afin2;
236 BD(12,22) = afin1;
237 BD(1,22) = 1;
238 BD(2,22) = afin2;
239 //Faum
240 BD(5,23) = afin1;
241 BD(6,23) = 1;
242 BD(7,23) = afin2;
243 BD(9,23) = afin1;
244 BD(10,23) = 1;
245 BD(11,23) = afin2;
246 BD(1,23) = afin1;
247 BD(2,23) = 1;
248 BD(3,23) = afin2;
249 //Fdim
250 BD(5,24) = afin1;
251 BD(6,24) = 1;
252 BD(7,24) = afin2;
253 BD(8,24) = afin1;
254 BD(9,24) = 1;
255 BD(10,24) = afin2;
256 BD(11,24) = afin1;
257 BD(12,24) = 1;
258 BD(1,24) = afin2;
259
260 //F#
```

```
261 //F#M
262 BD(6,25) = afin1;
263 BD(7,25) = 1;
264 BD(8,25) = afin2;
265 BD(10,25) = afin1;
266 BD(11,25) = 1;
267 BD(12,25) = afin2;
268 BD(1,25) = afin1;
269 BD(2,25) = 1;
270 BD(3,25) = afin2;
271 //F#m
272 BD(6,26) = afin1;
273 BD(7,26) = 1;
274 BD(8,26) = afin2;
275 BD(9,26) = afin1;
276 BD(10,26) = 1;
277 BD(11,26) = afin2;
278 BD(1,26) = afin1;
279 BD(2,26) = 1;
280 BD(3,26) = afin2;
281 //F#aum
282 BD(6,27) = afin1;
283 BD(7,27) = 1;
284 BD(8,27) = afin2;
285 BD(10,27) = afin1;
286 BD(11,27) = 1;
287 BD(12,27) = afin2;
288 BD(2,27) = afin1;
289 BD(3,27) = 1;
290 BD(4,27) = afin2;
291 //F#dim
292 BD(6,28) = afin1;
293 BD(7,28) = 1;
294 BD(8,28) = afin2;
295 BD(9,28) = afin1;
296 BD(10,28) = 1;
297 BD(11,28) = afin2;
298 BD(12,28) = afin1;
299 BD(1,28) = 1;
300 BD(2,28) = afin2;
301
302 //G
303 //GM
304 BD(7,29) = afin1;
305 BD(8,29) = 1;
306 BD(9,29) = afin2;
307 BD(11,29) = afin1;
```

```
308 BD(12,29) = 1;
309 BD(1,29) = afin2;
310 BD(2,29) = afin1;
311 BD(3,29) = 1;
312 BD(4,29) = afin2;
313 //Gm
314 BD(7,30) = afin1;
315 BD(8,30) = 1;
316 BD(9,30) = afin2;
317 BD(10,30) = afin1;
318 BD(11,30) = 1;
319 BD(12,30) = afin2;
320 BD(2,30) = afin1;
321 BD(3,30) = 1;
322 BD(4,30) = afin2;
323 //Gaum
324 BD(7,31) = afin1;
325 BD(8,31) = 1;
326 BD(9,31) = afin2;
327 BD(11,31) = afin1;
328 BD(12,31) = 1;
329 BD(1,31) = afin2;
330 BD(3,31) = afin1;
331 BD(4,31) = 1;
332 BD(5,31) = afin2;
333 //Gdim
334 BD(7,32) = afin1;
335 BD(8,32) = 1;
336 BD(9,32) = afin2;
337 BD(10,32) = afin1;
338 BD(11,32) = 1;
339 BD(12,32) = afin2;
340 BD(1,32) = afin1;
341 BD(2,32) = 1;
342 BD(3,32) = afin2;
343
344 //G#
345 //G#M
346 BD(8,33) = afin1;
347 BD(9,33) = 1;
348 BD(10,33) = afin2;
349 BD(12,33) = afin1;
350 BD(1,33) = 1;
351 BD(2,33) = afin2;
352 BD(3,33) = afin1;
353 BD(4,33) = 1;
354 BD(5,33) = afin2;
```

```
355 //G#m
356 BD(8,34) = afin1;
357 BD(9,34) = 1;
358 BD(10,34) = afin2;
359 BD(11,34) = afin1;
360 BD(12,34) = 1;
361 BD(1,34) = afin2;
362 BD(3,34) = afin1;
363 BD(4,34) = 1;
364 BD(5,34) = afin2;
365 //G#aum
366 BD(8,35) = afin1;
367 BD(9,35) = 1;
368 BD(10,35) = afin2;
369 BD(12,35) = afin1;
370 BD(1,35) = 1;
371 BD(2,35) = afin2;
372 BD(4,35) = afin1;
373 BD(5,35) = 1;
374 BD(6,35) = afin2;
375 //G#dim
376 BD(8,36) = afin1;
377 BD(9,36) = 1;
378 BD(10,36) = afin2;
379 BD(11,36) = afin1;
380 BD(12,36) = 1;
381 BD(1,36) = afin2;
382 BD(2,36) = afin1;
383 BD(3,36) = 1;
384 BD(4,36) = afin2;
385
386 //A
387 //AM
388 BD(9,37) = afin1;
389 BD(10,37) = 1;
390 BD(11,37) = afin2;
391 BD(1,37) = afin1;
392 BD(2,37) = 1;
393 BD(3,37) = afin2;
394 BD(4,37) = afin1;
395 BD(5,37) = 1;
396 BD(6,37) = afin2;
397 //Am
398 BD(9,38) = afin1;
399 BD(10,38) = 1;
400 BD(11,38) = afin2;
401 BD(12,38) = afin1;
```

```
402 BD(1,38) = 1;
403 BD(2,38) = afin2;
404 BD(4,38) = afin1;
405 BD(5,38) = 1;
406 BD(6,38) = afin2;
407 //Aaum
408 BD(9,39) = afin1;
409 BD(10,39) = 1;
410 BD(11,39) = afin2;
411 BD(1,39) = afin1;
412 BD(2,39) = 1;
413 BD(3,39) = afin2;
414 BD(5,39) = afin1;
415 BD(6,39) = 1;
416 BD(7,39) = afin2;
417 //Adim
418 BD(9,40) = afin1;
419 BD(10,40) = 1;
420 BD(11,40) = afin2;
421 BD(12,40) = afin1;
422 BD(1,40) = 1;
423 BD(2,40) = afin2;
424 BD(3,40) = afin1;
425 BD(4,40) = 1;
426 BD(5,40) = afin2;
427
428 //A#
429 //A#M
430 BD(10,41) = afin1;
431 BD(11,41) = 1;
432 BD(12,41) = afin2;
433 BD(2,41) = afin1;
434 BD(3,41) = 1;
435 BD(4,41) = afin2;
436 BD(5,41) = afin1;
437 BD(6,41) = 1;
438 BD(7,41) = afin2;
439 //A#m
440 BD(10,42) = afin1;
441 BD(11,42) = 1;
442 BD(12,42) = afin2;
443 BD(1,42) = afin1;
444 BD(2,42) = 1;
445 BD(3,42) = afin2;
446 BD(5,42) = afin1;
447 BD(6,42) = 1;
448 BD(7,42) = afin2;
```

```
449 //A#aum
450 BD(10,43) = afin1;
451 BD(11,43) = 1;
452 BD(12,43) = afin2;
453 BD(2,43) = afin1;
454 BD(3,43) = 1;
455 BD(4,43) = afin2;
456 BD(6,43) = afin1;
457 BD(7,43) = 1;
458 BD(8,43) = afin2;
459 //A#dim
460 BD(10,44) = afin1;
461 BD(11,44) = 1;
462 BD(12,44) = afin2;
463 BD(1,44) = afin1;
464 BD(2,44) = 1;
465 BD(3,44) = afin2;
466 BD(4,44) = afin1;
467 BD(5,44) = 1;
468 BD(6,44) = afin2;
469
470 //B
471 //BM
472 BD(11,45) = afin1;
473 BD(12,45) = 1;
474 BD(1,45) = afin2;
475 BD(3,45) = afin1;
476 BD(4,45) = 1;
477 BD(5,45) = afin2;
478 BD(6,45) = afin1;
479 BD(7,45) = 1;
480 BD(8,45) = afin2;
481 //Bm
482 BD(11,46) = afin1;
483 BD(12,46) = 1;
484 BD(1,46) = afin2;
485 BD(2,46) = afin1;
486 BD(3,46) = 1;
487 BD(4,46) = afin2;
488 BD(6,46) = afin1;
489 BD(7,46) = 1;
490 BD(8,46) = afin2;
491 //Baum
492 BD(11,47) = afin1;
493 BD(12,47) = 1;
494 BD(1,47) = afin2;
495 BD(3,47) = afin1;
```

```
496 BD(4,47) = 1;
497 BD(5,47) = afin2;
498 BD(7,47) = afin1;
499 BD(8,47) = 1;
500 BD(9,47) = afin2;
501 //Bdim
502 BD(11,48) = afin1;
503 BD(12,48) = 1;
504 BD(1,48) = afin2;
505 BD(2,48) = afin1;
506 BD(3,48) = 1;
507 BD(4,48) = afin2;
508 BD(5,48) = afin1;
509 BD(6,48) = 1;
510 BD(7,48) = afin2;
```

A.11 Módulo de Testes em Amostras

```
1  exec dc.sci;
2
3  //Script to tests
4
5  //Open the file to write results
6  file_results = mopen('chords_results.txt','w');
7
8  chord = 0;
9
10 //Testing combinations of CM
11 chord_result_1 = DA2(wavread('acordes_teste/CM1.wav'));
12 chord = chord + 1;
13 disp(chord)
14 chord_result_2 = DA2(wavread('acordes_teste/CM2.wav'));
15 chord = chord + 1;
16 disp(chord)
17 chord_result_3 = DA2(wavread('acordes_teste/CM3.wav'));
18 chord = chord + 1;
19 disp(chord)
20 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
21 //Testing combinations of Cm
22 chord_result_1 = DA2(wavread('acordes_teste/Cm1.wav'));
23 chord = chord + 1;
24 disp(chord)
25 chord_result_2 = DA2(wavread('acordes_teste/Cm2.wav'));
26 chord = chord + 1;
```

```
27 disp(chord)
28 chord_result_3 = DA2(wavread('acordes_teste/Cm3.wav'));
29 chord = chord + 1;
30 disp(chord)
31 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
32 //Testing combinations of Caum
33 chord_result_1 = DA2(wavread('acordes_teste/Caum1.wav'));
34 chord = chord + 1;
35 disp(chord)
36 chord_result_2 = DA2(wavread('acordes_teste/Caum2.wav'));
37 chord = chord + 1;
38 disp(chord)
39 chord_result_3 = DA2(wavread('acordes_teste/Caum3.wav'));
40 chord = chord + 1;
41 disp(chord)
42 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
43 //Testing combinations of Cdim
44 chord_result_1 = DA2(wavread('acordes_teste/Cdim1.wav'));
45 chord = chord + 1;
46 disp(chord)
47 chord_result_2 = DA2(wavread('acordes_teste/Cdim2.wav'));
48 chord = chord + 1;
49 disp(chord)
50 chord_result_3 = DA2(wavread('acordes_teste/Cdim3.wav'));
51 chord = chord + 1;
52 disp(chord)
53 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
54
55 //Testing combinations of C#M
56 chord_result_1 = DA2(wavread('acordes_teste/C#M1.wav'));
57 chord = chord + 1;
58 disp(chord)
59 chord_result_2 = DA2(wavread('acordes_teste/C#M2.wav'));
60 chord = chord + 1;
61 disp(chord)
62 chord_result_3 = DA2(wavread('acordes_teste/C#M3.wav'));
63 chord = chord + 1;
64 disp(chord)
65 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
66 //Testing combinations of C#m
67 chord_result_1 = DA2(wavread('acordes_teste/C#m1.wav'));
68 chord = chord + 1;
69 disp(chord)
70 chord_result_2 = DA2(wavread('acordes_teste/C#m2.wav'));
71 chord = chord + 1;
72 disp(chord)
73 chord_result_3 = DA2(wavread('acordes_teste/C#m3.wav'));
```

```
74 chord = chord + 1;
75 disp(chord)
76 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
77 //Testing combinations of C#aum
78 chord_result_1 = DA2(wavread('acordes_teste/C#aum1.wav'));
79 chord = chord + 1;
80 disp(chord)
81 chord_result_2 = DA2(wavread('acordes_teste/C#aum2.wav'));
82 chord = chord + 1;
83 disp(chord)
84 chord_result_3 = DA2(wavread('acordes_teste/C#aum3.wav'));
85 chord = chord + 1;
86 disp(chord)
87 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
88 //Testing combinations of C#dim
89 chord_result_1 = DA2(wavread('acordes_teste/C#dim1.wav'));
90 chord = chord + 1;
91 disp(chord)
92 chord_result_2 = DA2(wavread('acordes_teste/C#dim2.wav'));
93 chord = chord + 1;
94 disp(chord)
95 chord_result_3 = DA2(wavread('acordes_teste/C#dim3.wav'));
96 chord = chord + 1;
97 disp(chord)
98 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
99
100 //Testing combinations of DM
101 chord_result_1 = DA2(wavread('acordes_teste/DM1.wav'));
102 chord = chord + 1;
103 disp(chord)
104 chord_result_2 = DA2(wavread('acordes_teste/DM2.wav'));
105 chord = chord + 1;
106 disp(chord)
107 chord_result_3 = DA2(wavread('acordes_teste/DM3.wav'));
108 chord = chord + 1;
109 disp(chord)
110 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
111 //Testing combinations of Dm
112 chord_result_1 = DA2(wavread('acordes_teste/Dm1.wav'));
113 chord = chord + 1;
114 disp(chord)
115 chord_result_2 = DA2(wavread('acordes_teste/Dm2.wav'));
116 chord = chord + 1;
117 disp(chord)
118 chord_result_3 = DA2(wavread('acordes_teste/Dm3.wav'));
119 chord = chord + 1;
120 disp(chord)
```

```
121 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
122 //Testing combinations of Daum
123 chord_result_1 = DA2(wavread('acordes_teste/Daum1.wav'));
124 chord = chord + 1;
125 disp(chord)
126 chord_result_2 = DA2(wavread('acordes_teste/Daum2.wav'));
127 chord = chord + 1;
128 disp(chord)
129 chord_result_3 = DA2(wavread('acordes_teste/Daum3.wav'));
130 chord = chord + 1;
131 disp(chord)
132 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
133 //Testing combinations of Ddim
134 chord_result_1 = DA2(wavread('acordes_teste/Ddim1.wav'));
135 chord = chord + 1;
136 disp(chord)
137 chord_result_2 = DA2(wavread('acordes_teste/Ddim2.wav'));
138 chord = chord + 1;
139 disp(chord)
140 chord_result_3 = DA2(wavread('acordes_teste/Ddim3.wav'));
141 chord = chord + 1;
142 disp(chord)
143 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
144
145 //Testing combinations of D#M
146 chord_result_1 = DA2(wavread('acordes_teste/D#M1.wav'));
147 chord = chord + 1;
148 disp(chord)
149 chord_result_2 = DA2(wavread('acordes_teste/D#M2.wav'));
150 chord = chord + 1;
151 disp(chord)
152 chord_result_3 = DA2(wavread('acordes_teste/D#M3.wav'));
153 chord = chord + 1;
154 disp(chord)
155 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
156 //Testing combinations of D#m
157 chord_result_1 = DA2(wavread('acordes_teste/D#m1.wav'));
158 chord = chord + 1;
159 disp(chord)
160 chord_result_2 = DA2(wavread('acordes_teste/D#m2.wav'));
161 chord = chord + 1;
162 disp(chord)
163 chord_result_3 = DA2(wavread('acordes_teste/D#m3.wav'));
164 chord = chord + 1;
165 disp(chord)
166 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
167 //Testing combinations of D#aum
```

```
168 chord_result_1 = DA2(wavread('acordes_teste/D#aum1.wav'));
169 chord = chord + 1;
170 disp(chord)
171 chord_result_2 = DA2(wavread('acordes_teste/D#aum2.wav'));
172 chord = chord + 1;
173 disp(chord)
174 chord_result_3 = DA2(wavread('acordes_teste/D#aum3.wav'));
175 chord = chord + 1;
176 disp(chord)
177 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
178 //Testing combinations of D#dim
179 chord_result_1 = DA2(wavread('acordes_teste/D#dim1.wav'));
180 chord = chord + 1;
181 disp(chord)
182 chord_result_2 = DA2(wavread('acordes_teste/D#dim2.wav'));
183 chord = chord + 1;
184 disp(chord)
185 chord_result_3 = DA2(wavread('acordes_teste/D#dim3.wav'));
186 chord = chord + 1;
187 disp(chord)
188 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
189
190 //Testing combinations of EM
191 chord_result_1 = DA2(wavread('acordes_teste/EM1.wav'));
192 chord = chord + 1;
193 disp(chord)
194 chord_result_2 = DA2(wavread('acordes_teste/EM2.wav'));
195 chord = chord + 1;
196 disp(chord)
197 chord_result_3 = DA2(wavread('acordes_teste/EM3.wav'));
198 chord = chord + 1;
199 disp(chord)
200 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
201 //Testing combinations of Em
202 chord_result_1 = DA2(wavread('acordes_teste/Em1.wav'));
203 chord = chord + 1;
204 disp(chord)
205 chord_result_2 = DA2(wavread('acordes_teste/Em2.wav'));
206 chord = chord + 1;
207 disp(chord)
208 chord_result_3 = DA2(wavread('acordes_teste/Em3.wav'));
209 chord = chord + 1;
210 disp(chord)
211 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
212 //Testing combinations of Eaum
213 chord_result_1 = DA2(wavread('acordes_teste/Eaum1.wav'));
214 chord = chord + 1;
```

```
215 disp(chord)
216 chord_result_2 = DA2(wavread('acordes_teste/Eaum2.wav'));
217 chord = chord + 1;
218 disp(chord)
219 chord_result_3 = DA2(wavread('acordes_teste/Eaum3.wav'));
220 chord = chord + 1;
221 disp(chord)
222 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
223 //Testing combinations of Edim
224 chord_result_1 = DA2(wavread('acordes_teste/Edim1.wav'));
225 chord = chord + 1;
226 disp(chord)
227 chord_result_2 = DA2(wavread('acordes_teste/Edim2.wav'));
228 chord = chord + 1;
229 disp(chord)
230 chord_result_3 = DA2(wavread('acordes_teste/Edim3.wav'));
231 chord = chord + 1;
232 disp(chord)
233 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
234
235 //Testing combinations of FM
236 chord_result_1 = DA2(wavread('acordes_teste/FM1.wav'));
237 chord = chord + 1;
238 disp(chord)
239 chord_result_2 = DA2(wavread('acordes_teste/FM2.wav'));
240 chord = chord + 1;
241 disp(chord)
242 chord_result_3 = DA2(wavread('acordes_teste/FM3.wav'));
243 chord = chord + 1;
244 disp(chord)
245 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
246 //Testing combinations of Fm
247 chord_result_1 = DA2(wavread('acordes_teste/Fm1.wav'));
248 chord = chord + 1;
249 disp(chord)
250 chord_result_2 = DA2(wavread('acordes_teste/Fm2.wav'));
251 chord = chord + 1;
252 disp(chord)
253 chord_result_3 = DA2(wavread('acordes_teste/Fm3.wav'));
254 chord = chord + 1;
255 disp(chord)
256 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
257 //Testing combinations of Faum
258 chord_result_1 = DA2(wavread('acordes_teste/Faum1.wav'));
259 chord = chord + 1;
260 disp(chord)
261 chord_result_2 = DA2(wavread('acordes_teste/Faum2.wav'));
```

```
262 chord = chord + 1;
263 disp(chord)
264 chord_result_3 = DA2(wavread('acordes_teste/Faum3.wav'));
265 chord = chord + 1;
266 disp(chord)
267 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
268 //Testing combinations of Fdim
269 chord_result_1 = DA2(wavread('acordes_teste/Fdim1.wav'));
270 chord = chord + 1;
271 disp(chord)
272 chord_result_2 = DA2(wavread('acordes_teste/Fdim2.wav'));
273 chord = chord + 1;
274 disp(chord)
275 chord_result_3 = DA2(wavread('acordes_teste/Fdim3.wav'));
276 chord = chord + 1;
277 disp(chord)
278 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
279
280 //Testing combinations of F#M
281 chord_result_1 = DA2(wavread('acordes_teste/F#M1.wav'));
282 chord = chord + 1;
283 disp(chord)
284 chord_result_2 = DA2(wavread('acordes_teste/F#M2.wav'));
285 chord = chord + 1;
286 disp(chord)
287 chord_result_3 = DA2(wavread('acordes_teste/F#M3.wav'));
288 chord = chord + 1;
289 disp(chord)
290 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
291 //Testing combinations of F#m
292 chord_result_1 = DA2(wavread('acordes_teste/F#m1.wav'));
293 chord = chord + 1;
294 disp(chord)
295 chord_result_2 = DA2(wavread('acordes_teste/F#m2.wav'));
296 chord = chord + 1;
297 disp(chord)
298 chord_result_3 = DA2(wavread('acordes_teste/F#m3.wav'));
299 chord = chord + 1;
300 disp(chord)
301 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
302 //Testing combinations of F#aum
303 chord_result_1 = DA2(wavread('acordes_teste/F#aum1.wav'));
304 chord = chord + 1;
305 disp(chord)
306 chord_result_2 = DA2(wavread('acordes_teste/F#aum2.wav'));
307 chord = chord + 1;
308 disp(chord)
```

```
309 chord_result_3 = DA2(wavread('acordes_teste/F#aum3.wav'));
310 chord = chord + 1;
311 disp(chord)
312 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
313 //Testing combinations of F#dim
314 chord_result_1 = DA2(wavread('acordes_teste/F#dim1.wav'));
315 chord = chord + 1;
316 disp(chord)
317 chord_result_2 = DA2(wavread('acordes_teste/F#dim2.wav'));
318 chord = chord + 1;
319 disp(chord)
320 chord_result_3 = DA2(wavread('acordes_teste/F#dim3.wav'));
321 chord = chord + 1;
322 disp(chord)
323 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
324
325 //Testing combinations of GM
326 chord_result_1 = DA2(wavread('acordes_teste/GM1.wav'));
327 chord = chord + 1;
328 disp(chord)
329 chord_result_2 = DA2(wavread('acordes_teste/GM2.wav'));
330 chord = chord + 1;
331 disp(chord)
332 chord_result_3 = DA2(wavread('acordes_teste/GM3.wav'));
333 chord = chord + 1;
334 disp(chord)
335 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
336 //Testing combinations of Gm
337 chord_result_1 = DA2(wavread('acordes_teste/Gm1.wav'));
338 chord = chord + 1;
339 disp(chord)
340 chord_result_2 = DA2(wavread('acordes_teste/Gm2.wav'));
341 chord = chord + 1;
342 disp(chord)
343 chord_result_3 = DA2(wavread('acordes_teste/Gm3.wav'));
344 chord = chord + 1;
345 disp(chord)
346 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
347 //Testing combinations of Gaum
348 chord_result_1 = DA2(wavread('acordes_teste/Gaum1.wav'));
349 chord = chord + 1;
350 disp(chord)
351 chord_result_2 = DA2(wavread('acordes_teste/Gaum2.wav'));
352 chord = chord + 1;
353 disp(chord)
354 chord_result_3 = DA2(wavread('acordes_teste/Gaum3.wav'));
355 chord = chord + 1;
```

```
disp(chord)
mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
//Testing combinations of Gdim
chord_result_1 = DA2(wavread('acordes_teste/Gdim1.wav'));
chord = chord + 1;
disp(chord)
chord_result_2 = DA2(wavread('acordes_teste/Gdim2.wav'));
chord = chord + 1;
disp(chord)
chord_result_3 = DA2(wavread('acordes_teste/Gdim3.wav'));
chord = chord + 1;
disp(chord)
mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);

//Testing combinations of G#M
chord_result_1 = DA2(wavread('acordes_teste/G#M1.wav'));
chord = chord + 1;
disp(chord)
chord_result_2 = DA2(wavread('acordes_teste/G#M2.wav'));
chord = chord + 1;
disp(chord)
chord_result_3 = DA2(wavread('acordes_teste/G#M3.wav'));
chord = chord + 1;
disp(chord)
mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
//Testing combinations of G#m
chord_result_1 = DA2(wavread('acordes_teste/G#m1.wav'));
chord = chord + 1;
disp(chord)
chord_result_2 = DA2(wavread('acordes_teste/G#m2.wav'));
chord = chord + 1;
disp(chord)
chord_result_3 = DA2(wavread('acordes_teste/G#m3.wav'));
chord = chord + 1;
disp(chord)
mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
//Testing combinations of G#aum
chord_result_1 = DA2(wavread('acordes_teste/G#aum1.wav'));
chord = chord + 1;
disp(chord)
chord_result_2 = DA2(wavread('acordes_teste/G#aum2.wav'));
chord = chord + 1;
disp(chord)
chord_result_3 = DA2(wavread('acordes_teste/G#aum3.wav'));
chord = chord + 1;
disp(chord)
mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
```

```
403 //Testing combinations of G#dim
404 chord_result_1 = DA2(wavread('acordes_teste/G#dim1.wav'));
405 chord = chord + 1;
406 disp(chord)
407 chord_result_2 = DA2(wavread('acordes_teste/G#dim2.wav'));
408 chord = chord + 1;
409 disp(chord)
410 chord_result_3 = DA2(wavread('acordes_teste/G#dim3.wav'));
411 chord = chord + 1;
412 disp(chord)
413 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
414
415 //Testing combinations of AM
416 chord_result_1 = DA2(wavread('acordes_teste/AM1.wav'));
417 chord = chord + 1;
418 disp(chord)
419 chord_result_2 = DA2(wavread('acordes_teste/AM2.wav'));
420 chord = chord + 1;
421 disp(chord)
422 chord_result_3 = DA2(wavread('acordes_teste/AM3.wav'));
423 chord = chord + 1;
424 disp(chord)
425 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
426 //Testing combinations of Am
427 chord_result_1 = DA2(wavread('acordes_teste/Am1.wav'));
428 chord = chord + 1;
429 disp(chord)
430 chord_result_2 = DA2(wavread('acordes_teste/Am2.wav'));
431 chord = chord + 1;
432 disp(chord)
433 chord_result_3 = DA2(wavread('acordes_teste/Am3.wav'));
434 chord = chord + 1;
435 disp(chord)
436 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
437 //Testing combinations of Aaum
438 chord_result_1 = DA2(wavread('acordes_teste/Aaum1.wav'));
439 chord = chord + 1;
440 disp(chord)
441 chord_result_2 = DA2(wavread('acordes_teste/Aaum2.wav'));
442 chord = chord + 1;
443 disp(chord)
444 chord_result_3 = DA2(wavread('acordes_teste/Aaum3.wav'));
445 chord = chord + 1;
446 disp(chord)
447 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
448 //Testing combinations of Adim
449 chord_result_1 = DA2(wavread('acordes_teste/Adim1.wav'));
```

```
450 chord = chord + 1;
451 disp(chord)
452 chord_result_2 = DA2(wavread('acordes_teste/Adim2.wav'));
453 chord = chord + 1;
454 disp(chord)
455 chord_result_3 = DA2(wavread('acordes_teste/Adim3.wav'));
456 chord = chord + 1;
457 disp(chord)
458 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
459
460 //Testing combinations of A#M
461 chord_result_1 = DA2(wavread('acordes_teste/A#M1.wav'));
462 chord = chord + 1;
463 disp(chord)
464 chord_result_2 = DA2(wavread('acordes_teste/A#M2.wav'));
465 chord = chord + 1;
466 disp(chord)
467 chord_result_3 = DA2(wavread('acordes_teste/A#M3.wav'));
468 chord = chord + 1;
469 disp(chord)
470 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
471 //Testing combinations of A#m
472 chord_result_1 = DA2(wavread('acordes_teste/A#m1.wav'));
473 chord = chord + 1;
474 disp(chord)
475 chord_result_2 = DA2(wavread('acordes_teste/A#m2.wav'));
476 chord = chord + 1;
477 disp(chord)
478 chord_result_3 = DA2(wavread('acordes_teste/A#m3.wav'));
479 chord = chord + 1;
480 disp(chord)
481 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
482 //Testing combinations of A#aum
483 chord_result_1 = DA2(wavread('acordes_teste/A#aum1.wav'));
484 chord = chord + 1;
485 disp(chord)
486 chord_result_2 = DA2(wavread('acordes_teste/A#aum2.wav'));
487 chord = chord + 1;
488 disp(chord)
489 chord_result_3 = DA2(wavread('acordes_teste/A#aum3.wav'));
490 chord = chord + 1;
491 disp(chord)
492 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
493 //Testing combinations of A#dim
494 chord_result_1 = DA2(wavread('acordes_teste/A#dim1.wav'));
495 chord = chord + 1;
496 disp(chord)
```

```
497 chord_result_2 = DA2(wavread('acordes_teste/A#dim2.wav'));
498 chord = chord + 1;
499 disp(chord)
500 chord_result_3 = DA2(wavread('acordes_teste/A#dim3.wav'));
501 chord = chord + 1;
502 disp(chord)
503 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
504
505 //Testing combinations of BM
506 chord_result_1 = DA2(wavread('acordes_teste/BM1.wav'));
507 chord = chord + 1;
508 disp(chord)
509 chord_result_2 = DA2(wavread('acordes_teste/BM2.wav'));
510 chord = chord + 1;
511 disp(chord)
512 chord_result_3 = DA2(wavread('acordes_teste/BM3.wav'));
513 chord = chord + 1;
514 disp(chord)
515 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
516 //Testing combinations of Bm
517 chord_result_1 = DA2(wavread('acordes_teste/Bm1.wav'));
518 chord = chord + 1;
519 disp(chord)
520 chord_result_2 = DA2(wavread('acordes_teste/Bm2.wav'));
521 chord = chord + 1;
522 disp(chord)
523 chord_result_3 = DA2(wavread('acordes_teste/Bm3.wav'));
524 chord = chord + 1;
525 disp(chord)
526 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
527 //Testing combinations of Baum
528 chord_result_1 = DA2(wavread('acordes_teste/Baum1.wav'));
529 chord = chord + 1;
530 disp(chord)
531 chord_result_2 = DA2(wavread('acordes_teste/Baum2.wav'));
532 chord = chord + 1;
533 disp(chord)
534 chord_result_3 = DA2(wavread('acordes_teste/Baum3.wav'));
535 chord = chord + 1;
536 disp(chord)
537 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
538 //Testing combinations of Bdim
539 chord_result_1 = DA2(wavread('acordes_teste/Bdim1.wav'));
540 chord = chord + 1;
541 disp(chord)
542 chord_result_2 = DA2(wavread('acordes_teste/Bdim2.wav'));
543 chord = chord + 1;
```

```
544 disp(chord)
545 chord_result_3 = DA2(wavread('acordes_teste/Bdim3.wav'));
546 chord = chord + 1;
547 disp(chord)
548 mputl(chord_result_1+';'+chord_result_2+';'+chord_result_3,file_results);
549
550 mclose(file_results);
```

A.12 Cronograma para Próximas Atividades

Segue cronograma de trabalho para as atividades do Trabalho de Conclusão de Curso 2:

- 20/12/2014 até 26/12/2014 - efetivar correções da banca examinadora;
- 27/12/2014 até 03/01/2015 - implementação da camada de detecção de inversões;
- 04/01/2014 até 12/01/2015 - implementação da camada de detecção de inversões;
- 13/01/2014 até 21/01/2015 - implementação da camada de transições rítmicas;
- 22/01/2014 até 31/01/2015 - implementação da transformada de wavelets;
- 01/02/2014 até 29/05/2015 - escrita do trabalho.