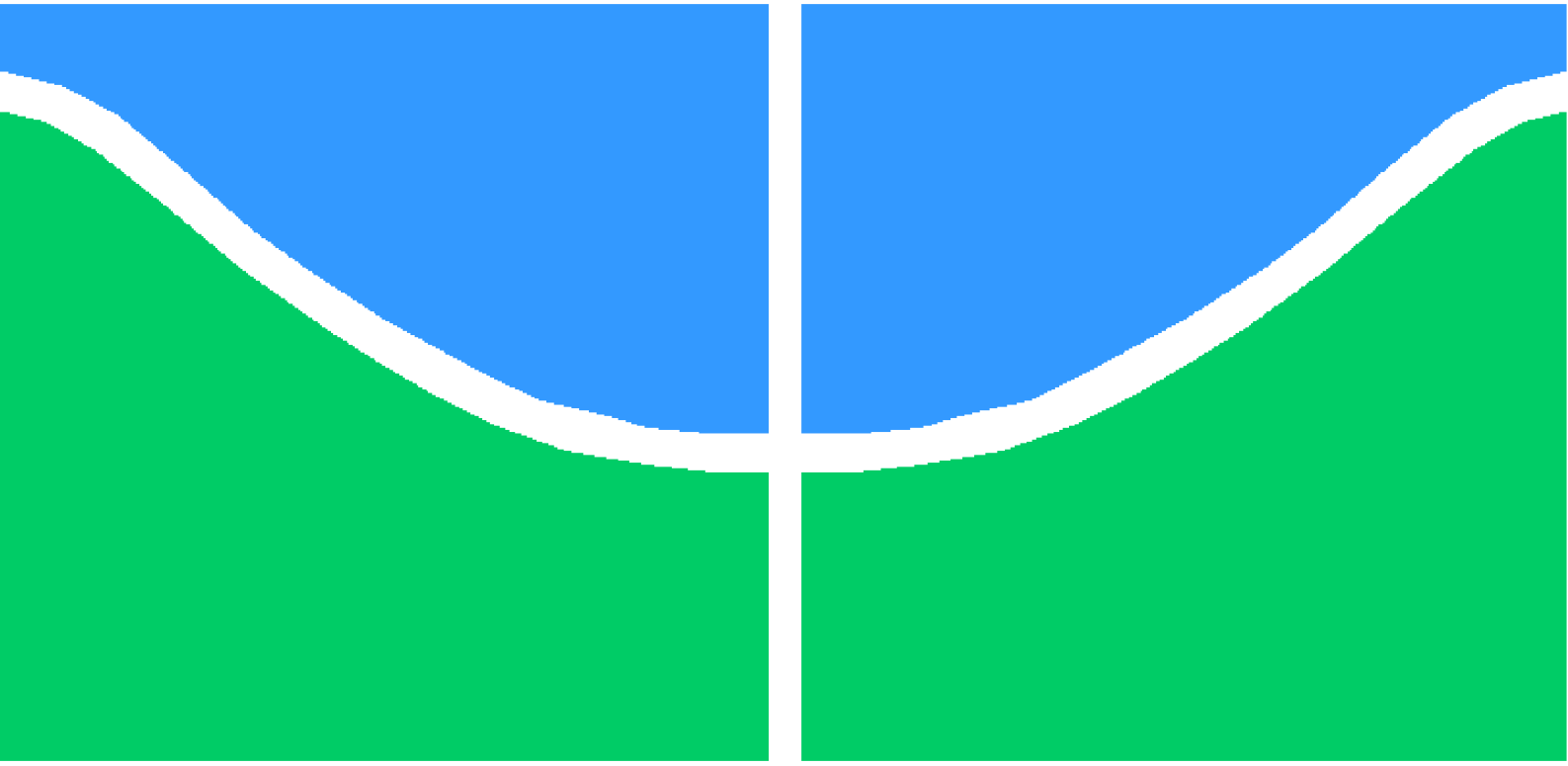

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

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-
90 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

Lista de tabelas

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

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:

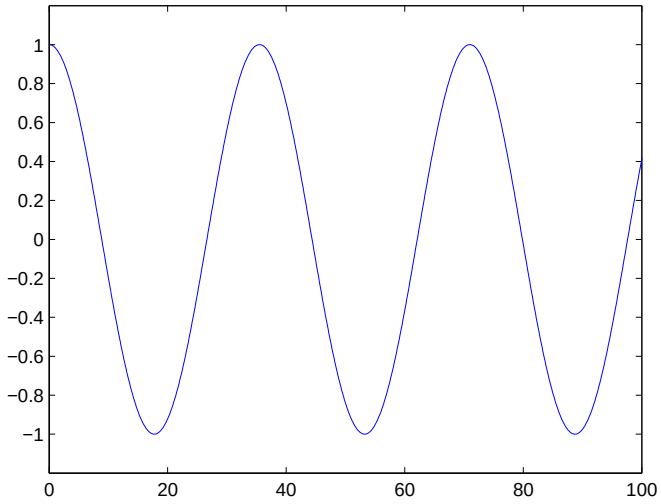

Figura 1 – Função da Equação 2.3

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:

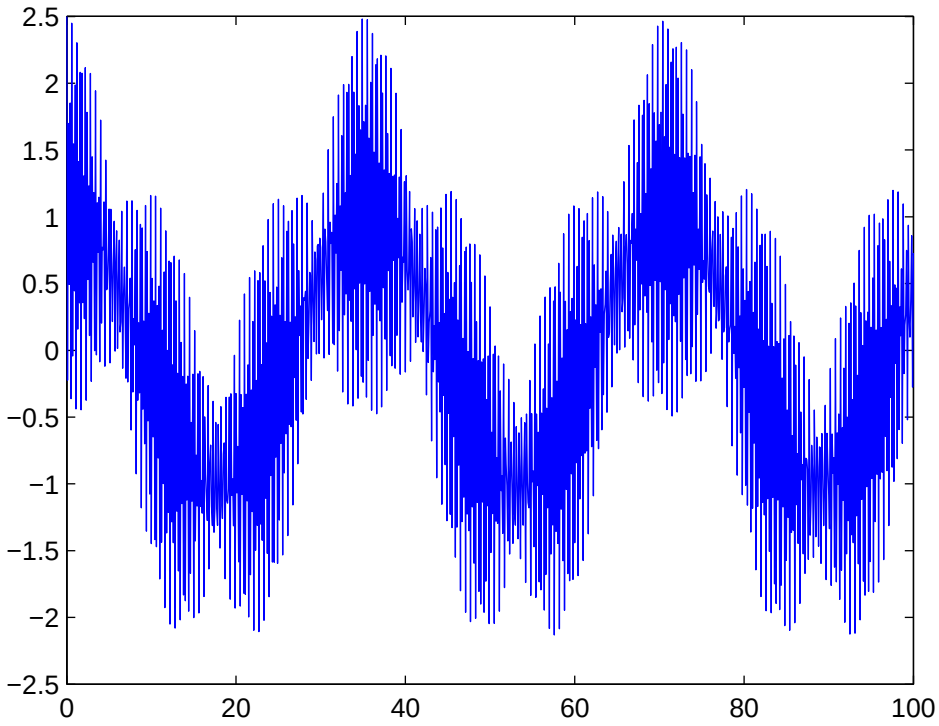

Figura 2 – Função da Equação 2.4

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):

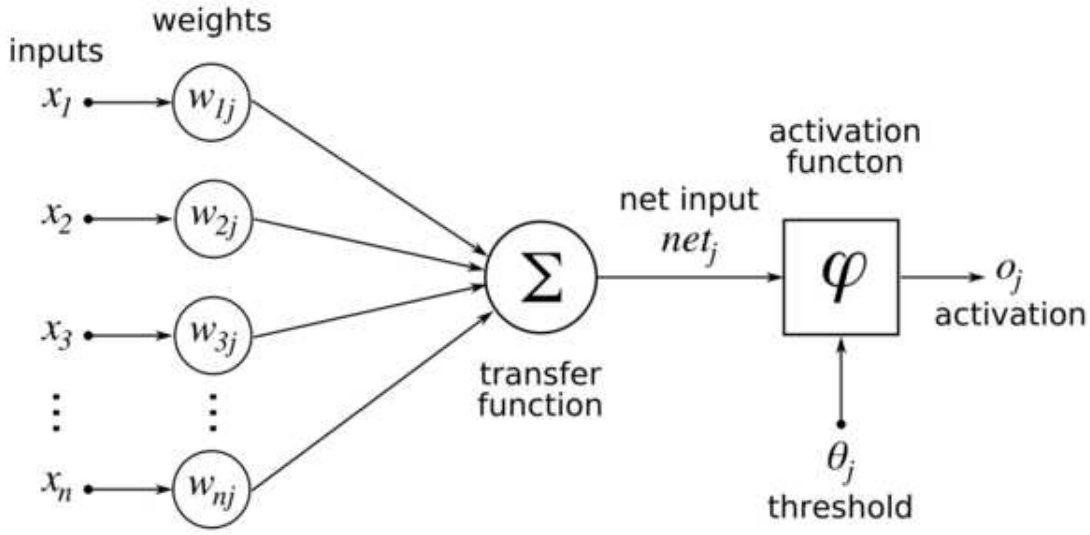

Figura 4 – Modelo de um neurônio

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):

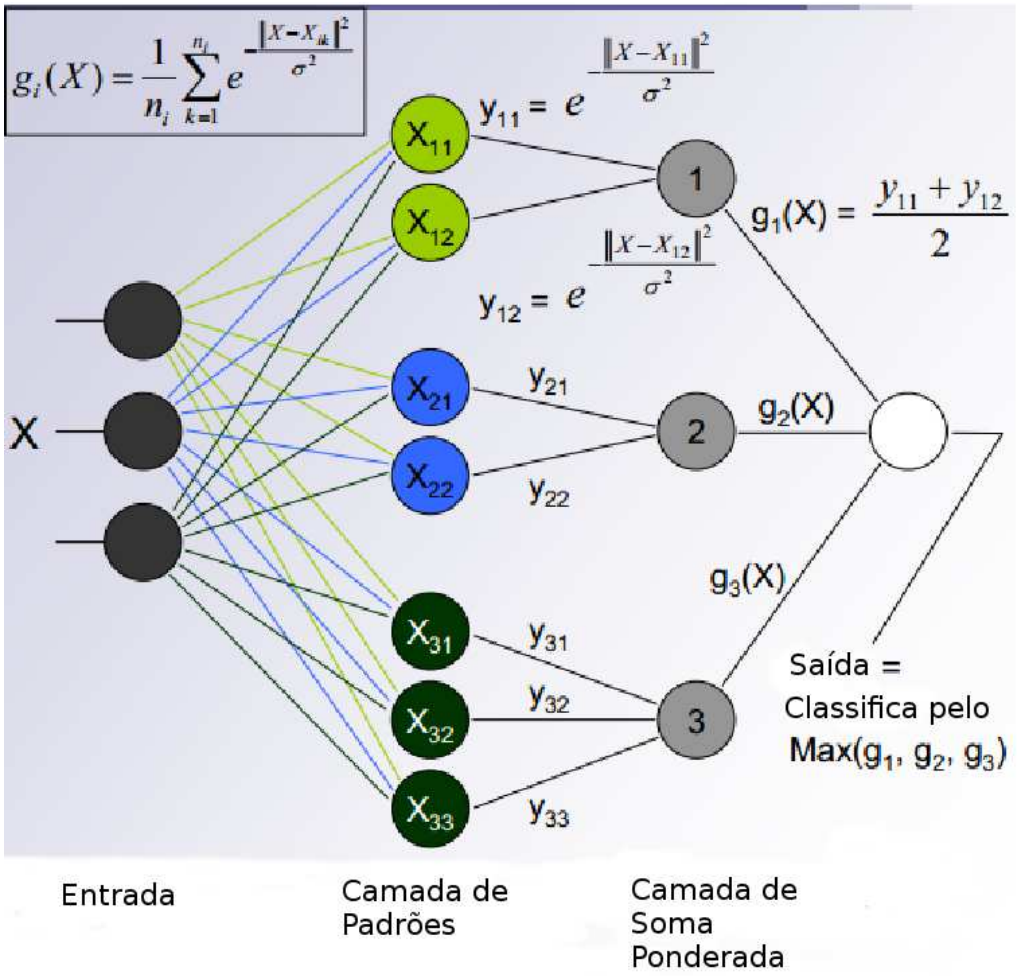

Figura 5 – Modelo arquitetural da PNN

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 em 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 constituído 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.

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:

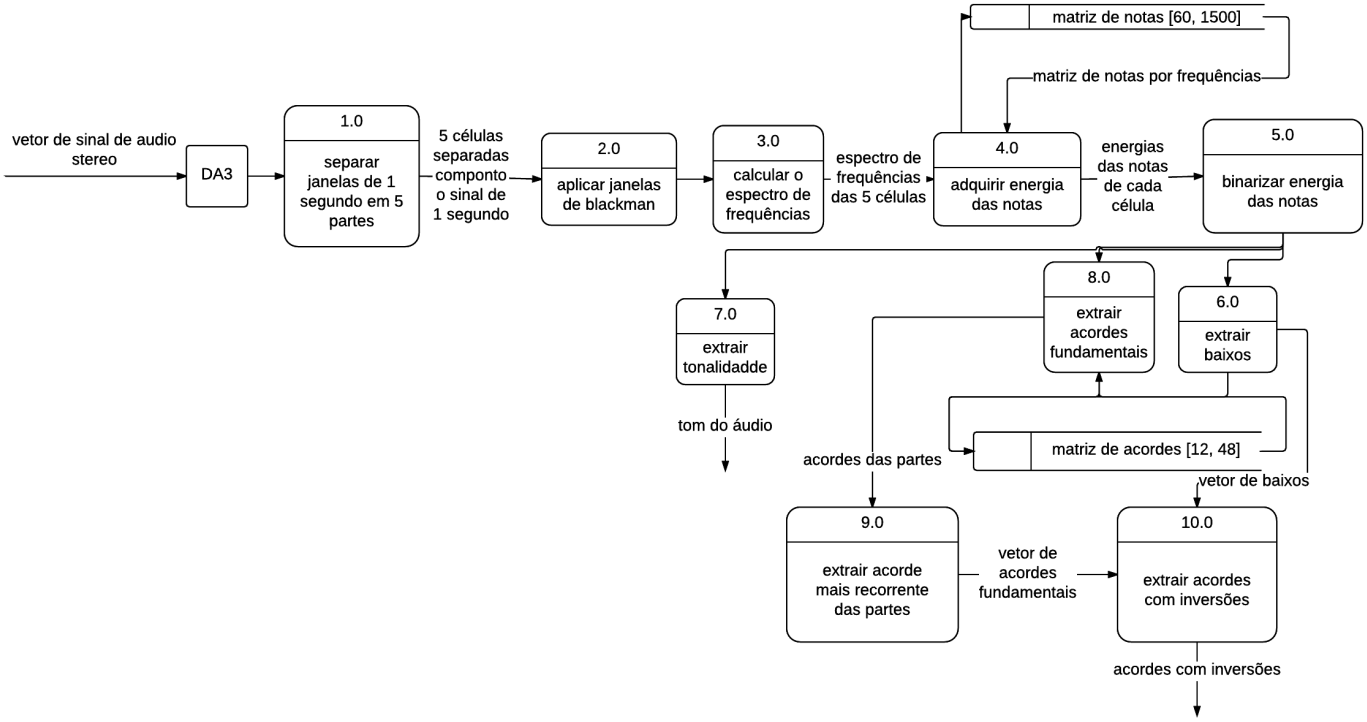

Figura 6 – Diagrama de Fluxo de Dados

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. A figura 7 ilustra o processo descrito e o código do mesmo está presente secção A.1 dos apêndices.

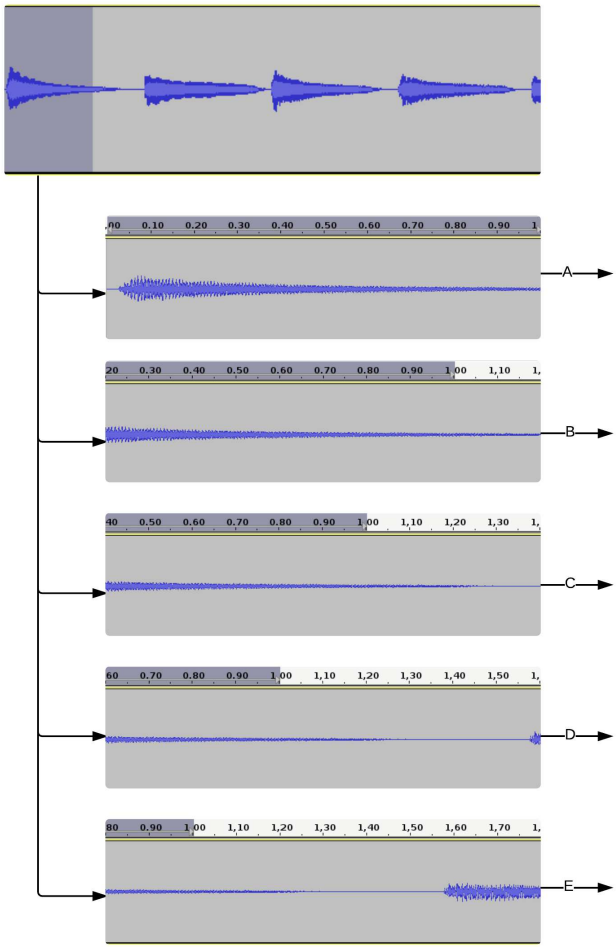

Figura 7 – Ilustração do Processo 1

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-Time-Fourier-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. A figura 8 ilustra o processo descrito e o código do mesmo está presente secção A.2 dos apêndices.

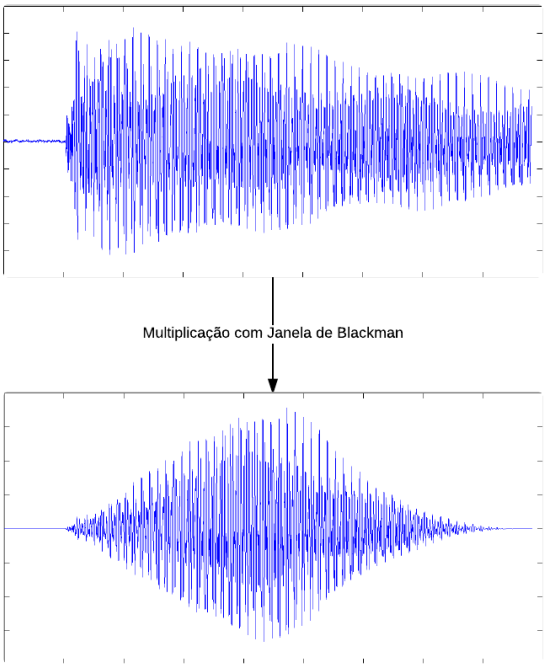

Figura 8 – Ilustração do Processo 2

3.2.3 Procedimento 3: Calcular o Espectro de Frequências

********** FIGURA ********** 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. Esse procedimento pode ser conferido na secção A.3 dos apêndices.

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

********** FIGURA ********** Nesse procedimento cada espectro de frequência é correlacionado com conjunto de notas musicais dado um conjunto de frequên-

cias que nelas estão presentes. Ao final do processo espera-se matrizes de conjuntos de notas para cada uma das 5 partes da janela. Esse procedimento pode ser conferido na secção A.4 dos apêndices.

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

********** FIGURA ********** 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. Esse procedimento pode ser conferido na secção A.5 dos apêndices.

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

********** FIGURA ********** 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. Esse procedimento pode ser conferido na secção A.6 dos apêndices.

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

********** FIGURA ********** 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. Esse procedimento pode ser conferido na secção A.7 dos apêndices.

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

********** FIGURA ********** 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. Esse procedimento pode ser conferido na secção A.8 dos apêndices.

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

********** FIGURA ********** 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. Esse procedimento pode ser conferido na secção A.9 dos apêndices.

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

********** FIGURA ********** 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. Esse procedimento pode ser conferido na secção A.10 dos apêndices.

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 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

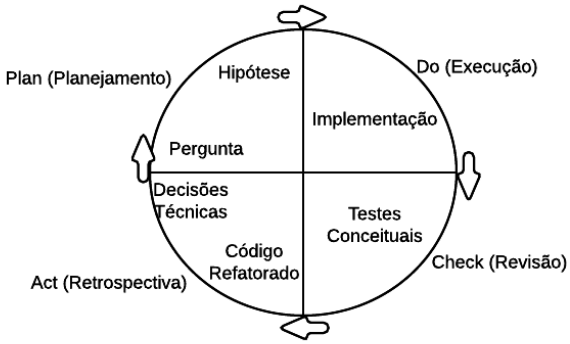

Figura 9 – Modelo de Ciclo Adaptado

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.

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

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:**

```
1 fs = 44100;
2 f = (0:length(som)-1)*fs/length(som);
3 freq = f(1:round(length(f)/2));
4 SOM = abs(fft(som));
5 SOM = SOM/max(SOM);
6 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;

.
.
.
%(430 LINHAS DE ACORDES)

%RADIAL BASIS LAYER para BD notas

while (i <= 48)
    S2(i) = sum(abs(S1.* BD(i,:)));
    i = i + 1;
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:**

```
function [signal, imin, imax, iterations, energy] = ...
tree_iterator(signal, mini, maxi, imin, imax, iterations, energy)

if iterations == 0
    if mini >= 1 && maxi <= 22050
      [signal, imin, imax, iterations, energy] = ...
      tree_iterator(signal, mini, maxi, 1, 22050, 1, 0);
    else
      imin = 1;
      imax = 22050;
      return;
end
elseif iterations > 0
    imean = (imax - imin)/2 + imin;
    %Low
    if mini >= imin && maxi <= imean
      [h0, h1] = wfilters('bior6.8');
      [signal, y1] = decomposition_1level_qmf(h0, h1, signal);
      energy = sum(abs(signal)) + energy;
      iterations = iterations + 1;
      imax = imean;
      [signal, imin, imax, iterations, energy] = ...
      tree_iterator(signal, mini, maxi, imin, ...
      imax, iterations, energy);
    %High
    elseif mini >= imean && maxi <= imax
      [h0, h1] = wfilters('bior6.8');
      [y0, signal] = decomposition_1level_qmf(h0, h1, signal);
      energy = sum(abs(signal)) + energy;
      iterations = iterations + 1;
      imin = imean;
      [signal, imin, imax, iterations, energy] = ...
      tree_iterator(signal, mini, maxi, ...
      imin, imax, iterations, energy);
    else
      return;
    end
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:**

```
1  function [chord_pitch, chord_pitch_number] = ...
2  get_chord_pitch(notes_time, time_seconds_total, chords)
3
4    dictionary_chords = ...
5  .
6  .
7  .
8    notes_energy_total(60) = 0;
9    for note = 1:60
10     notes_energy_total(note) = sum([notes_time(:,note)]);
```

```
11    end
12
13    % discover tone music
14    notes_energy_tone(12) = 0;
15    for note = 1:12
16      notes_energy_tone(note) = notes_energy_total(note) ...
17       + notes_energy_total(note + 12) ...
18        + notes_energy_total(note + 2*12) + ...
19        notes_energy_total(note + 3*12) ...
20          + notes_energy_total(note + 4*12);
21    end
22
23    % find chord tone
24    load_chords_tone;
25    chords_tone(48) = 0;
26    for chord = 1:48
27      chords_tone(chord) = sum((notes_energy_tone.* ...
28      chords_tone_mask(:, chord)'.^2));
29    end
30
31    chord_pitch_number = find(chords_tone==max(chords_tone));
32    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:**

```
1  % binarize set of notes
2      for set = 1:5
3          notes_time = set_of_notes_time{set};
4
5          for time = 1:time_seconds_total
6              for note = 1:60
7                  if notes_time(time, note) < max(max(notes_time))/180
```

```
8                       notes_time(time, note) = 0;
9                   else
10                      notes_time(time, note) = 1;
11                  end
12              end
13          end
14
15          set_of_notes_time{set} = notes_time;
16      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:**

```
1  function set_of_windows_signals = ...
2  build_window_short_fft(signal, time, fs)
3    signal = [signal(:)];
4
5      % part A
6      time_start_A = round(1+((time-1)*fs));
7      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.

Em visto do que foi desenvolvido nesse capítulo, foi apresentado a metodologia de execução do presente trabalho, o desenvolvimento do sistema-solução como um todo e o andamento em ciclos de execução de desenvolvimento da solução proposta. Vale ressaltar que cada módulo do sistema-solução desenvolvido gera resultados no que tange o processamento de áudio. No capítulo seguinte serão apresentados os resultados.

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 reconhecimento 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 acordes 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.

[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. A figura 10 ilustra as regiões e limites usados (DOZOL, 2014):

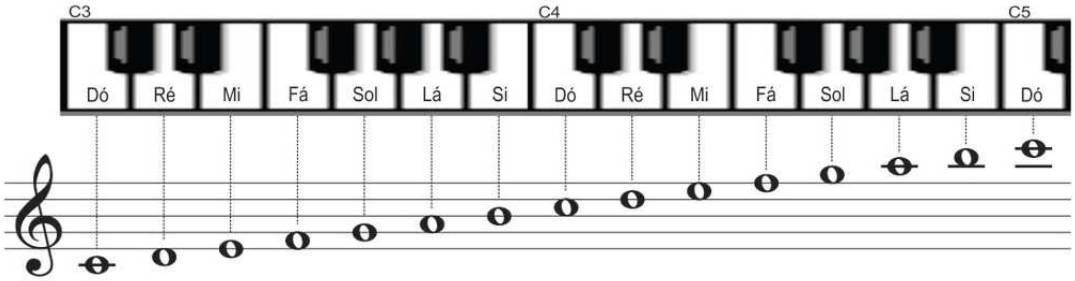

Figura 10 – Teclado ilustrativo para execução dos acordes.

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. A figura 11 ilustra o processo esquematizado.

Figura 11 – 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:

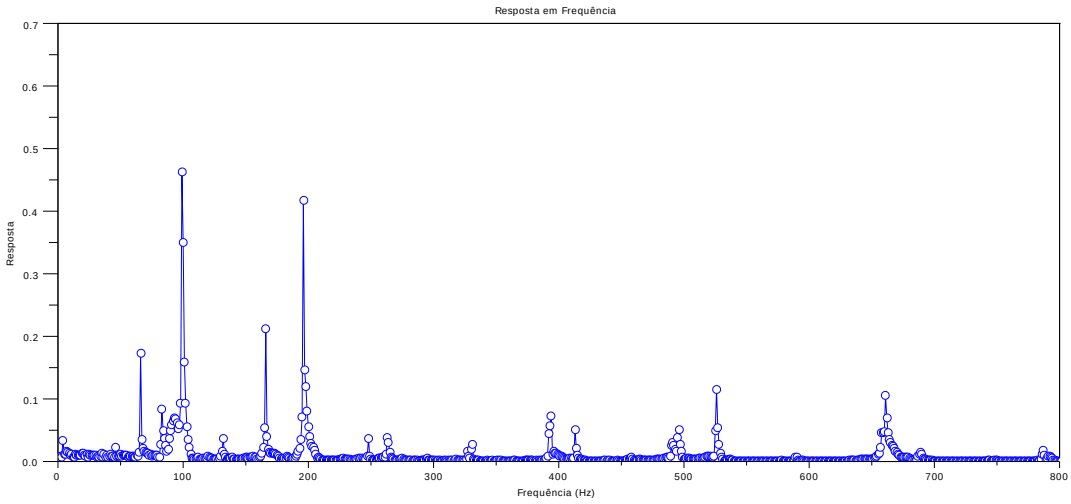

Figura 12 – Gráfico da resposta em frequência para a gravação do acorde CM.

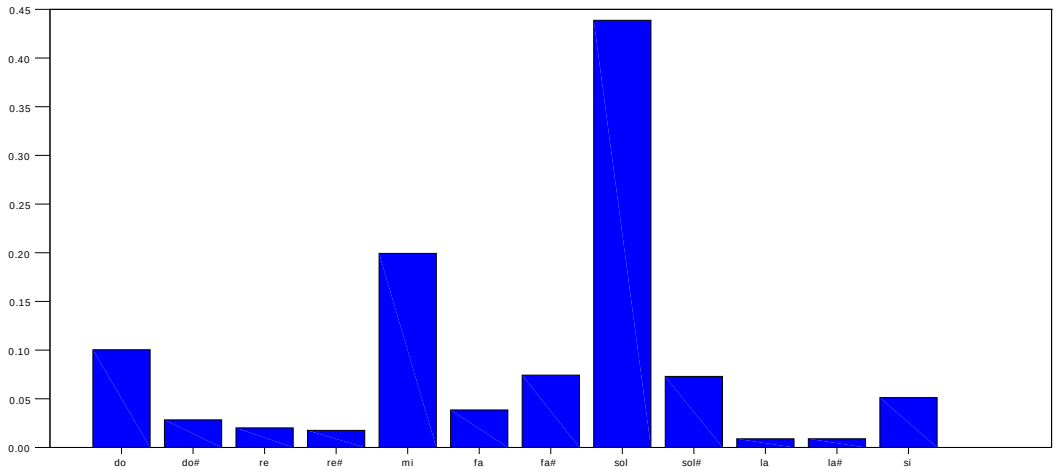

Figura 13 – Gráfico de sugestão de notas para a gravação do acorde CM.

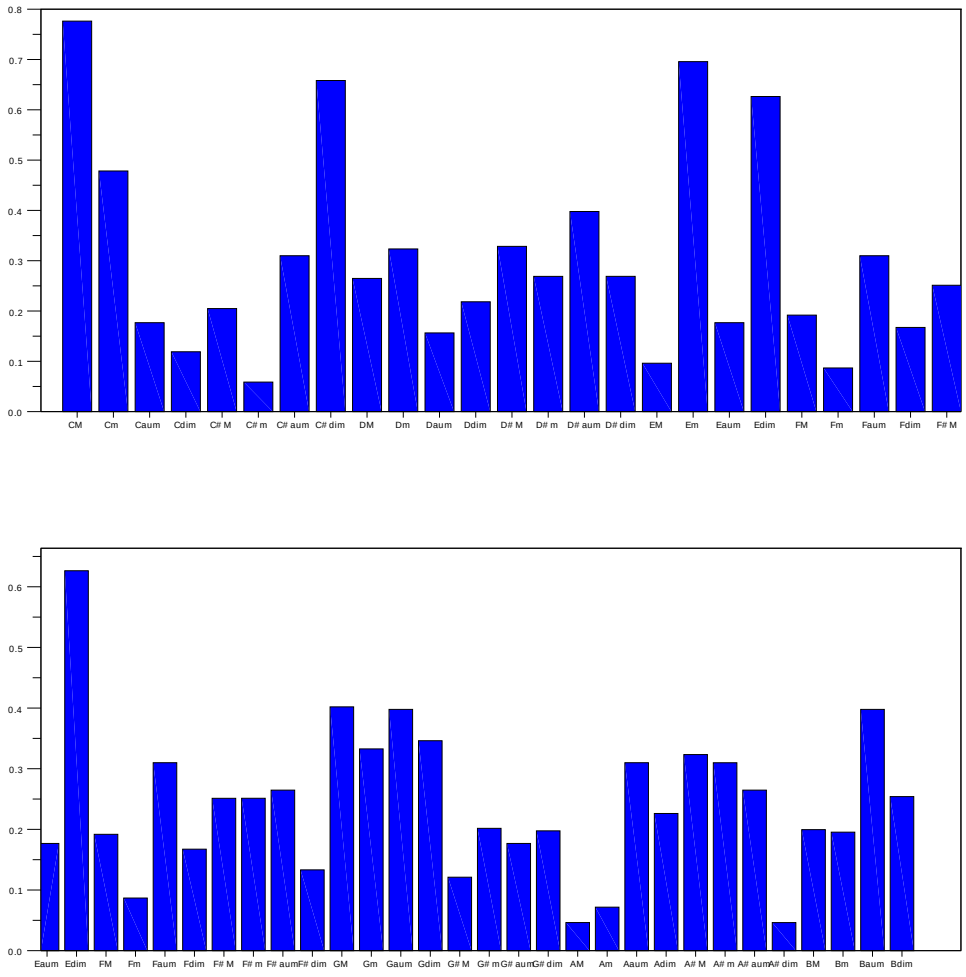

Figura 14 – Gráficos de sugestão de acordes a gravação do acorde CM.

Do resultado da primeira camada de processamento é gerado o gráfico da figura 12. 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 13. 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 14. 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:

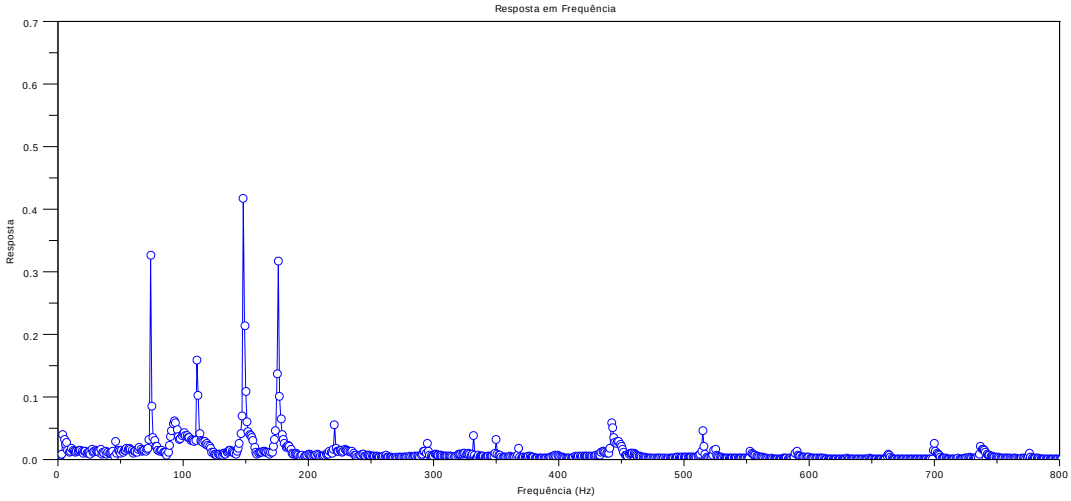

Figura 15 – Gráfico da resposta em frequência para a gravação do acorde Dm.

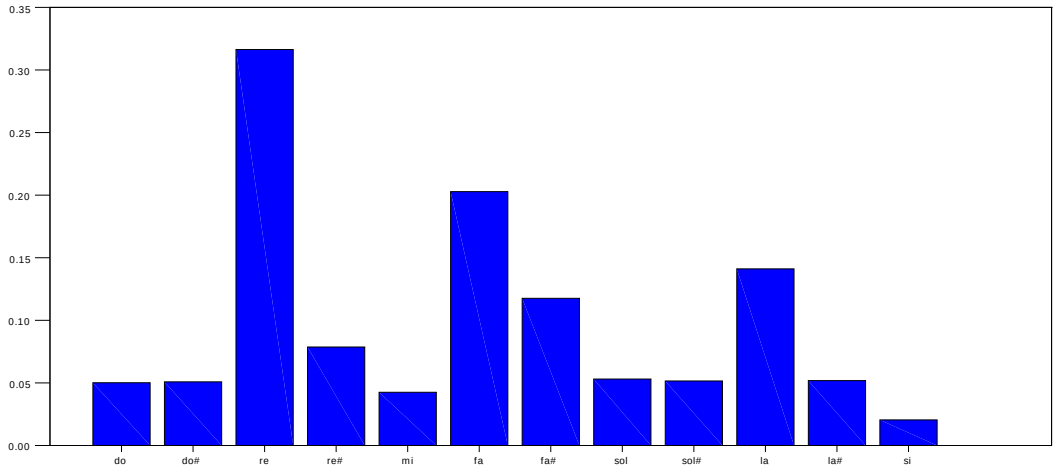

Figura 16 – Gráfico de sugestão de notas para a gravação do acorde Dm.

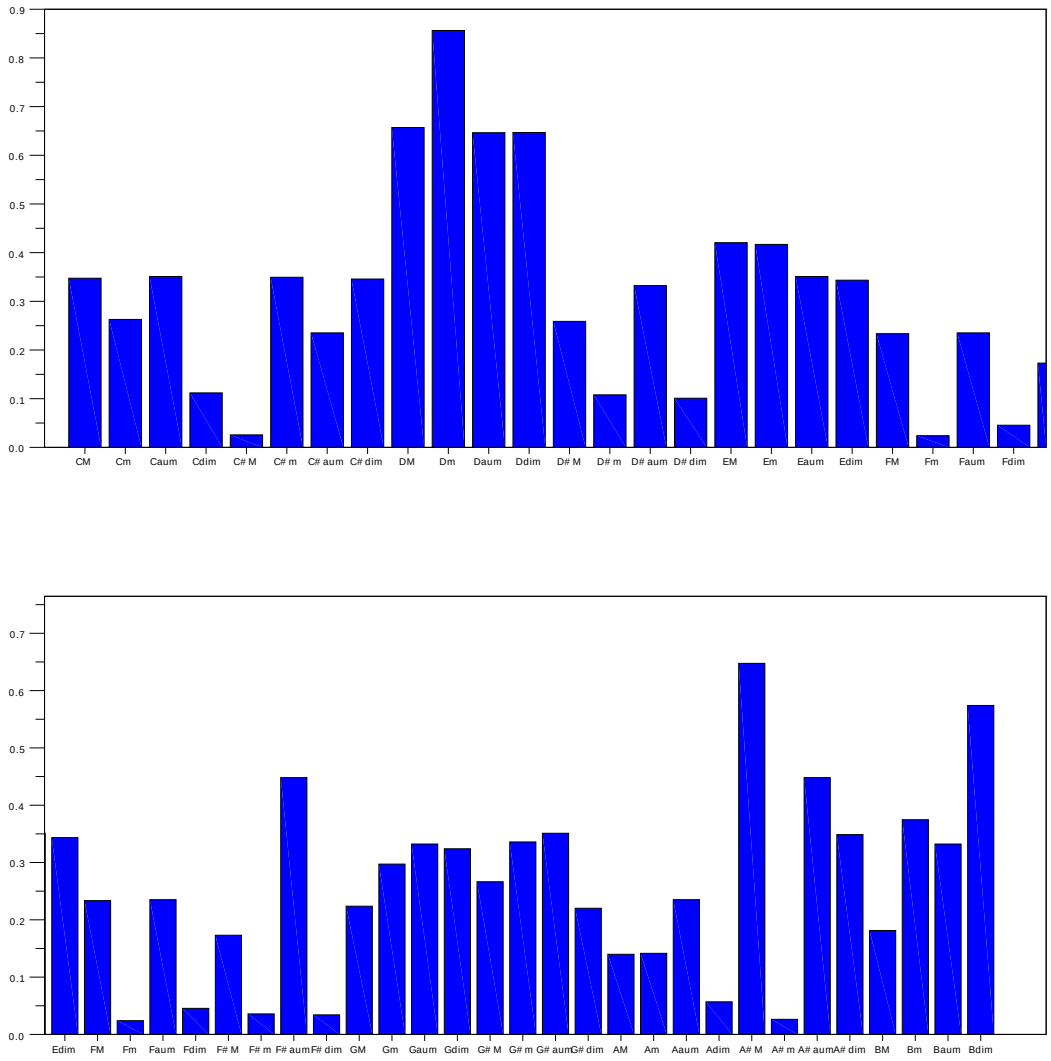

Figura 17 – Gráficos de sugestão de acordes a gravação do acorde *Dm*.

Do resultado da primeira camada de processamento é gerado o gráfico da figura 15. 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 16. 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 17. 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:

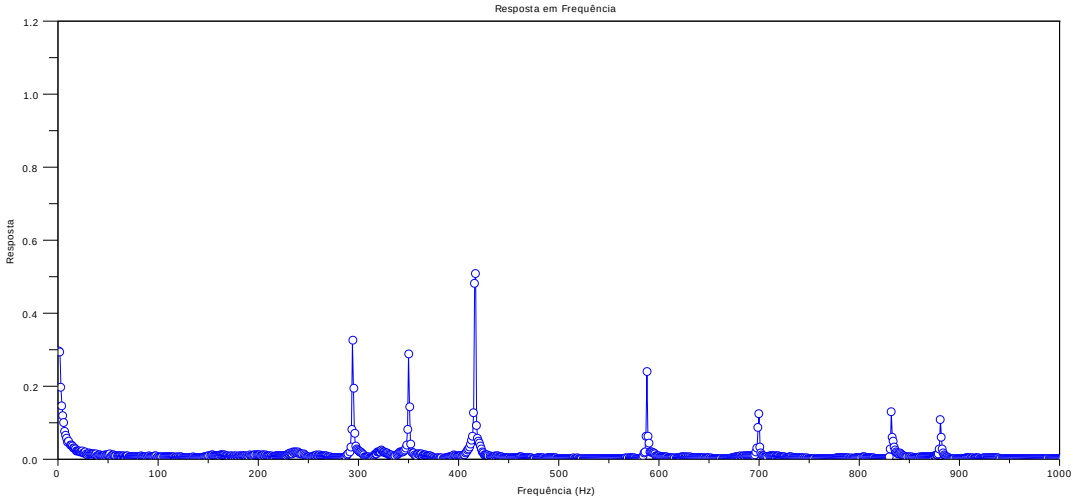

Figura 18 – Gráfico da resposta em frequência para a gravação do acorde $Ddim$.

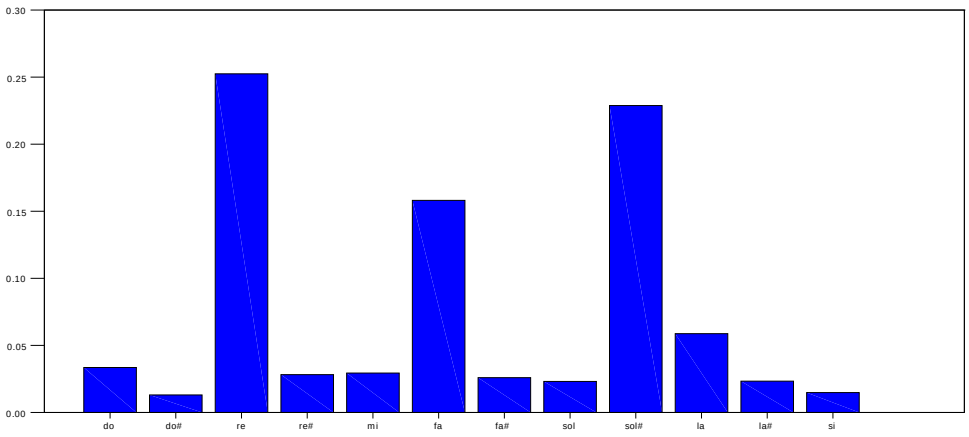

Figura 19 – Gráfico de sugestão de notas para a gravação do acorde $Ddim$.

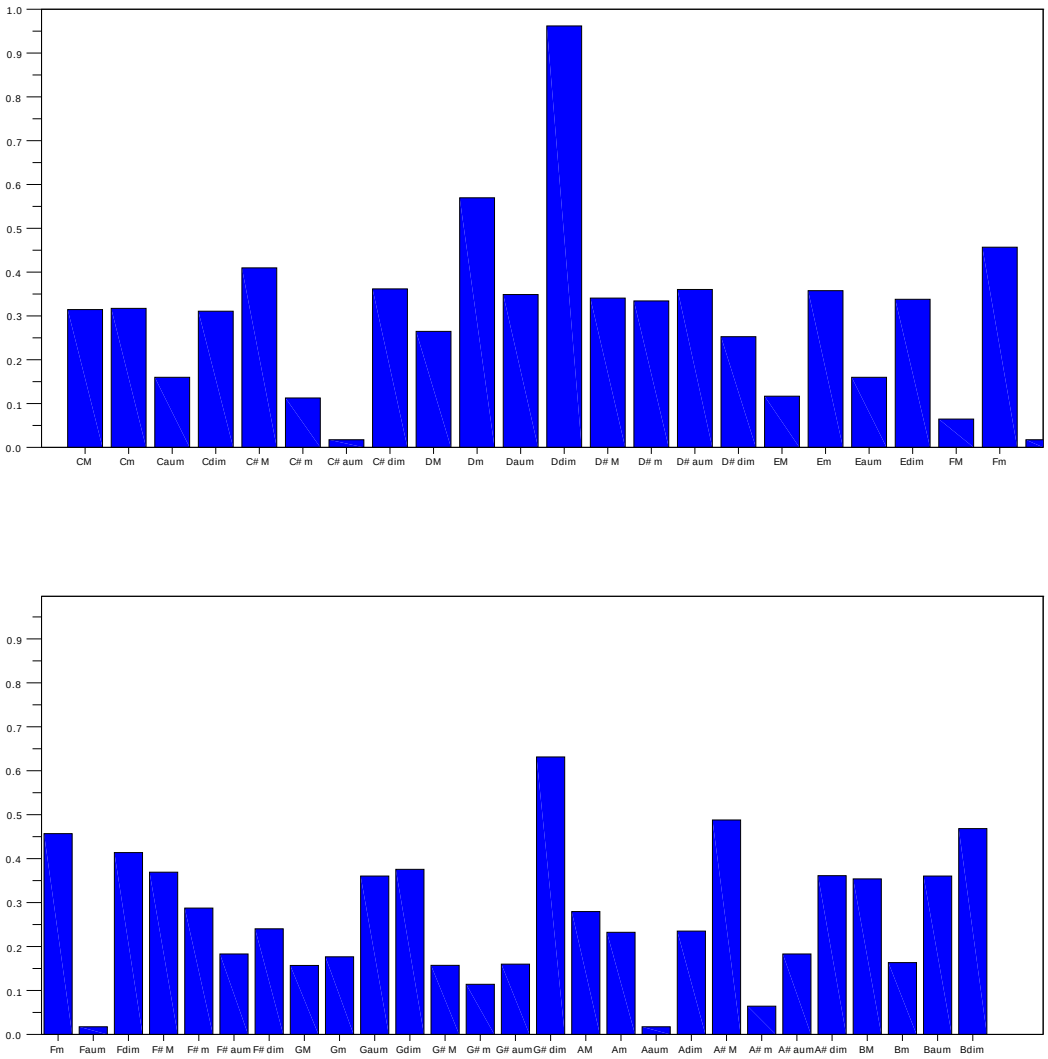

Figura 20 – Gráficos de sugestão de acordes a gravação do acorde *Ddim*.

Do resultado da primeira camada de processamento é gerado o gráfico da figura 18. 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 19. 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 20. 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:

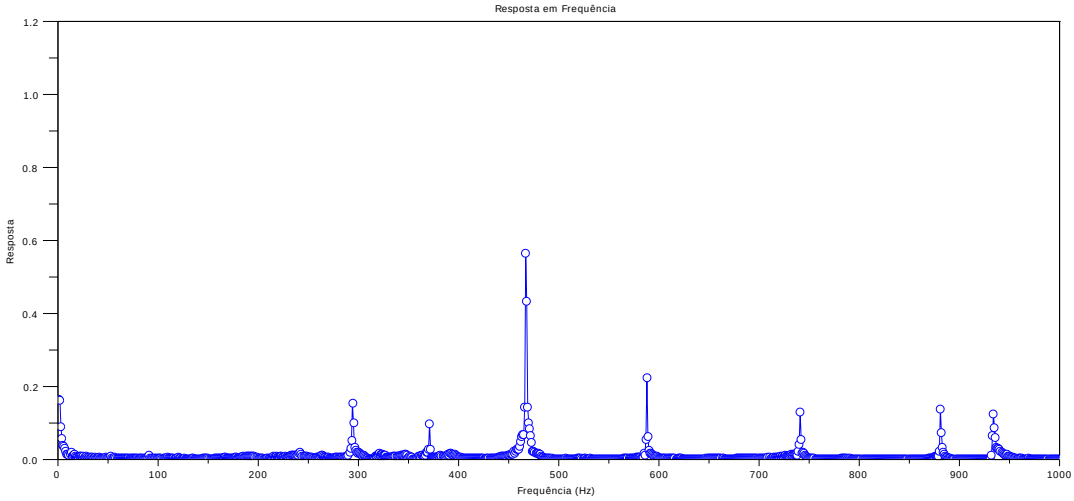

Figura 21 – Gráfico da resposta em frequência para a gravação do acorde *Daum*.

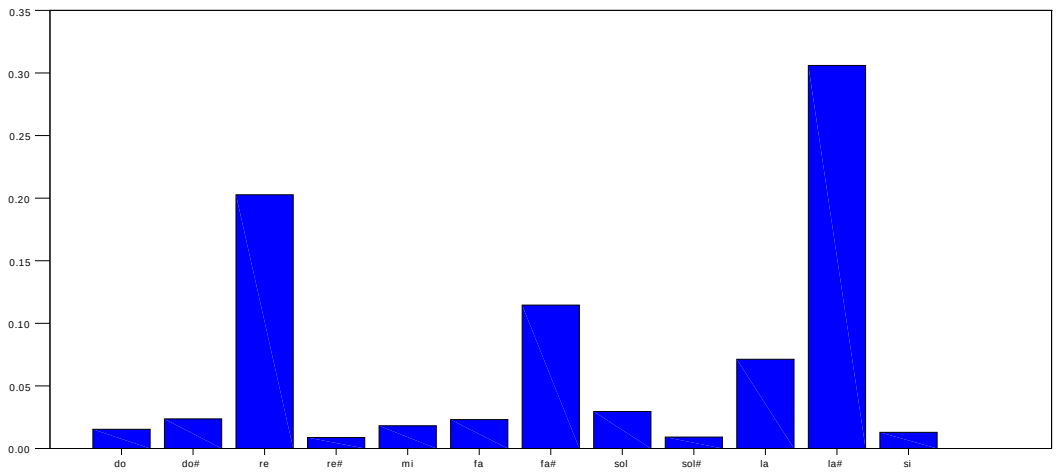

Figura 22 – Gráfico de sugestão de notas para a gravação do acorde *Daum*.

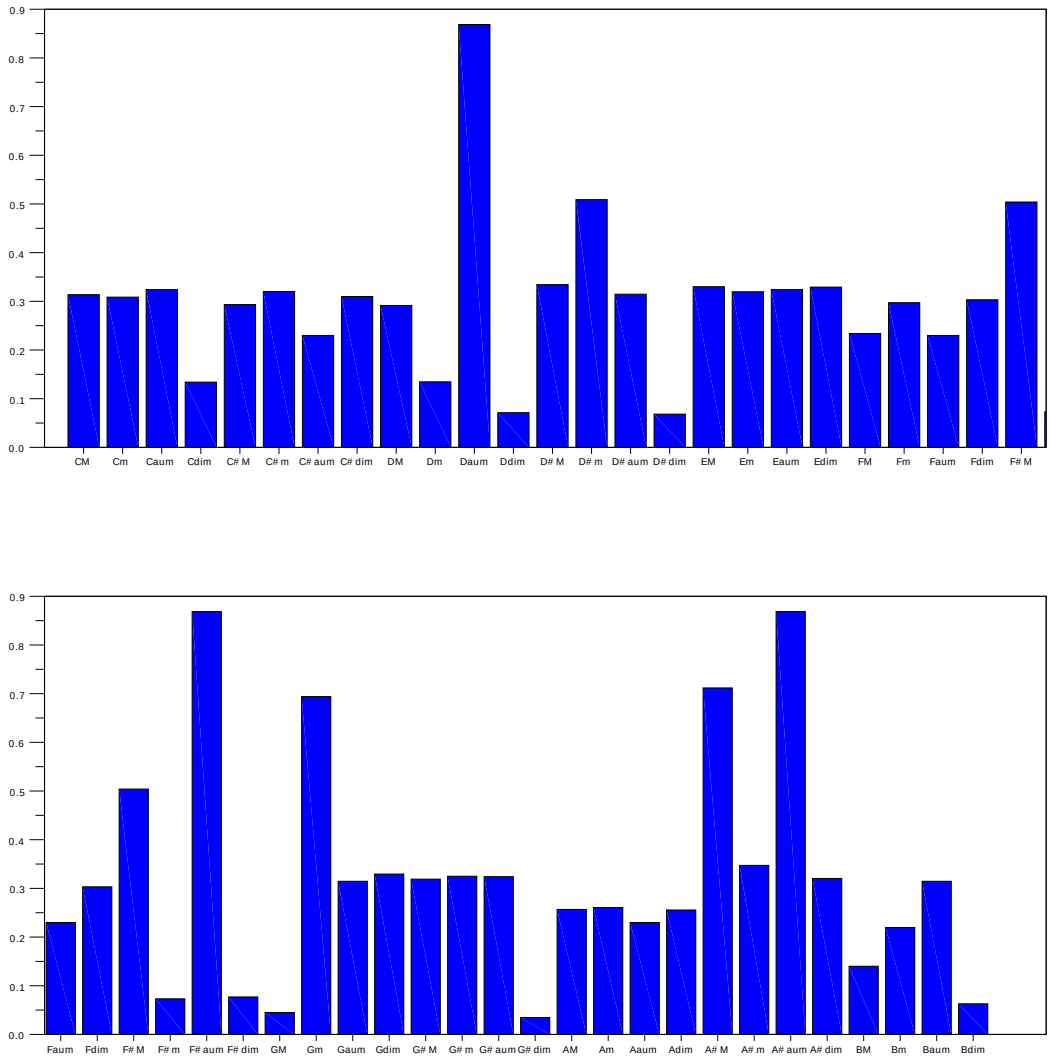

Figura 23 – Gráficos de sugestão de acordes a gravação do acorde *Daum*.

Do resultado da primeira camada de processamento é gerado o gráfico da figura 21. 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 22. 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.

Do resultado da terceira camada de processamento são gerados os gráficos da figura 23. Essa camada é relativa aos 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. Visto que o sistema possui o módulo de extração de baixos, a solução indicou que o acorde tocado foi *Daum*, visto que, como é mostrado no gráfico do espectro de frequências, a nota *Ré* é a nota mais grave da tríade.

Segue na tabela 1 os resultados do sistema dado todas as combinações dos conjuntos de acordes de tríades possíveis. Esses resultados foram gerados pelo script disponível no repositório desse trabalho [2].

4.2 Detecção de Transições Rítmicas

Com o intuito de detectar acordes ao longo do tempo, foi pensado um algoritmo baseado correlação de níveis de energia ao longo do tempo que delimitariam, em teoria, ocorrências de acordes. Com essa delimitação seria possível ajustar a janela de análise em frequência nos limites energéticos.

A figura 24 ilustra um exemplo de acordes tocados ao longo do tempo.

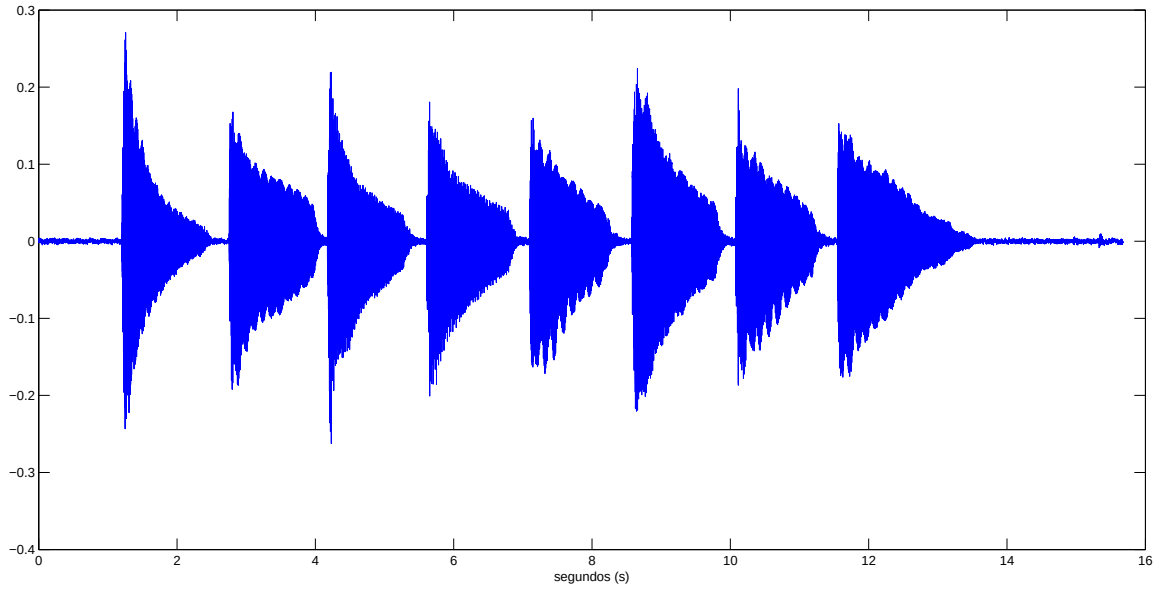

Figura 24 – Acordes tocados ao longo do tempo.

Na figura acima é perceptível observar, pelos níveis de energia, 8 acordes tocados. Para validar a solução, o gráfico de picos de transição rítmica deverá apresentar 8 picos ao longo do tempo, destacando, além do número de picos, a localidade coesa de cada pico.

A figura 25 mostra o gráfico de picos de transição rítmica.

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

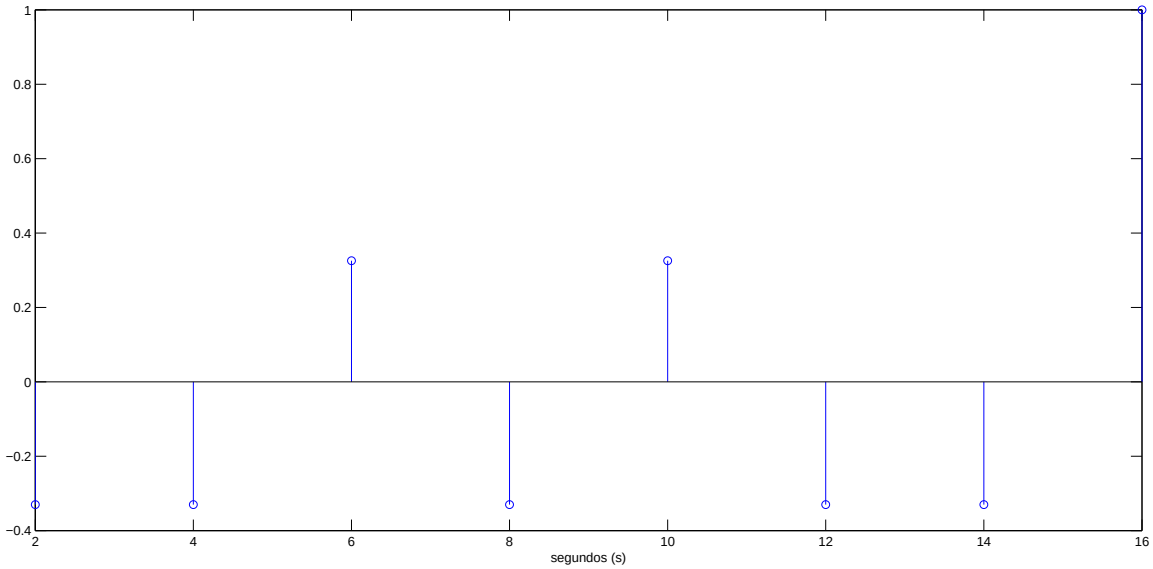

Figura 25 – Acordes tocados ao longo do tempo.

Pode-se observar da figura acima que a quantidade de picos é coerente com a quantidade encontrar na figura 24, entretanto as localizações

4.3 Implementação da Transformada Wavelets

4.4 Transcrição de Notas ao Longo do Tempo

4.5 Transcrição Automática de Acordes ao Longo do Tempo

4.6 Extração da Tonalidade do Áudio

-	**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.11.

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.

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 65.

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 39.

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 MATLAB.

A.1 Código do Procedimento 1

```
% get total seconds of time to mensure the length of music
signal = signal(:,1);
time_seconds_total = fix((length(signal)/fs));

% preparing struct to allocate notes in time
set_of_notes_time = {};
for set = 1:5
    notes_time(time_seconds_total, 60) = 0;
    set_of_notes_time{set} = notes_time;
end
```

A.2 Código do Procedimento 2

```

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

```

```
21      % part C (displacement = + 0.4 seconds)
22      time_start_C = round(1+((time-1)*fs+0.4*fs));
23      time_end_C = round((time+0.4)*fs);
24      if time_start_C < length(signal) && time_end_C <= length(signal)
25          signal_time_C = signal(time_start_C:time_end_C);
26          signal_time_C = blackman(length(signal_time_C)).*signal_time_C;
27      else
28          signal_time_C(length(signal)) = 0;
29      end
30
31      % part D (displacement = + 0.6 seconds)
32      time_start_D = round(1+((time-1)*fs+0.6*fs));
33      time_end_D = round((time+0.6)*fs);
34      if time_start_D < length(signal) && time_end_D <= length(signal)
35          signal_time_D = signal(time_start_D:time_end_D);
36          signal_time_D = blackman(length(signal_time_D)).*signal_time_D;
37      else
38          signal_time_D(length(signal)) = 0;
39      end
40
41      % part E (displacement = + 0.8 seconds)
42      time_start_E = round(1+((time-1)*fs+0.8*fs));
43      time_end_E = round((time+0.8)*fs);
44      if time_start_E < length(signal) && time_end_E <= length(signal)
45          signal_time_E = signal(time_start_E:time_end_E);
46          signal_time_E = blackman(length(signal_time_E)).*signal_time_E;
47      else
48          signal_time_E(length(signal)) = 0;
49      end
50
51      set_of_windows_signals = {};
52      if length(signal_time_A) == length(signal_time_B) && ...
53          length(signal_time_A) == length(signal_time_C) && ...
54           length(signal_time_A) == length(signal_time_D) &&  ...
55            length(signal_time_A) == length(signal_time_E)
56          set_of_windows_signals{1} = signal_time_A;
57          set_of_windows_signals{2} = signal_time_B;
58          set_of_windows_signals{3} = signal_time_C;
59          set_of_windows_signals{4} = signal_time_D;
60          set_of_windows_signals{5} = signal_time_E;
61      else
62          set_of_windows_signals{1} = signal_time_A;
63          set_of_windows_signals{2} = signal_time_A;
64          set_of_windows_signals{3} = signal_time_A;
65          set_of_windows_signals{4} = signal_time_A;
66          set_of_windows_signals{5} = signal_time_A;
67      end
```

A.3 Código do Procedimento 3

```
% get frequency spectrum
function set_of_spectrums = get_frequency_spectrum( ...
set_of_windows_signals, sampling)

    % allocate struct to spectrum
    set_of_spectrums = {};
    sampling = sampling/21;

    for part_signal_iterator = 1:5
        % make downsample to put frequency max in 1050 Hz
        signal = downsample(set_of_windows_signals{ ...
part_signal_iterator}, 21);
        % doing fourier transform
        frequencies=(0:length(signal)-1)*sampling/length(signal);
        module_fft = abs(fft(signal));
        f_round = round(frequencies);
        frequencies_energy(max(f_round)) = 0;
        for slot = 2:length(f_round)
            frequencies_energy(f_round(slot)) = module_fft(slot);
        end
        frequency_spectrum_part = frequencies_energy(1:fix(end/2));
        set_of_spectrums{part_signal_iterator} = frequency_spectrum_part;
    end
```

A.4 Código do procedimento 4

```
function set_of_notes_time = get_energy_notes(set_of_spectrums, ...
set_of_notes_time, time)

    % load data notes
    load_notes;

    for set = 1:5
        respfreq = set_of_spectrums{set};
        notes_time = set_of_notes_time{set};

        % this case works in one case
        respfreq = [respfreq zeros(1, length(notes(1,:)) - ...
length(respfreq))];
        for note = 1:60
            notes_time(time, note) = sum((respfreq.*notes(note,:)).^2);
```

```
16          end
17
18          set_of_notes_time{set} = notes_time;
19      end
```

A.5 Código do procedimento 5

```
1  % binarize set of notes
2      for set = 1:5
3          notes_time = set_of_notes_time{set};
4
5          for time = 1:time_seconds_total
6              for note = 1:60
7                  if notes_time(time, note) < max(max(notes_time))/180
8                      notes_time(time, note) = 0;
9                  else
10                     notes_time(time, note) = 1;
11                 end
12             end
13         end
14
15         set_of_notes_time{set} = notes_time;
16     end
```

A.6 Código do procedimento 6

```
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) ...
```

```
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
```

A.7 Código do Procedimento 7

```
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};
```

A.8 Código do Procedimento 8

```
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
```

A.9 Código do Procedimento 9

```
1  function chords = analyse_set_of_chords(set_of_chords)
2
3      set_of_chords_A = set_of_chords{1};
4      set_of_chords_B = set_of_chords{2};
5      set_of_chords_C = set_of_chords{3};
6      set_of_chords_D = set_of_chords{4};
7      set_of_chords_E = set_of_chords{5};
8
9      total_seconds = length(set_of_chords_A);
10     chords(1:total_seconds) = 0;
11     for time = 1:total_seconds
12         chords_to_analyse = [set_of_chords_A(time) ...
13          set_of_chords_B(time) ...
14          set_of_chords_C(time) set_of_chords_D(time) ...
15           set_of_chords_E(time)];
16         chords(time) = mode(chords_to_analyse);
17     end
18 end
```

A.10 Código do procedimento 10

```
1  function chords_with_bass = get_chords_bass(chords_number, bass_time)
2
3   load_dictionary_chords;
4
5   % build chords with bass to translate to dictionary
6   chords_with_bass_number = {};
7   chord_iterator = 1;
8   for chord = 1:48
9      for bass = 1:12
10         chords_with_bass_number{chord_iterator} = [chord, bass];
11         chord_iterator = chord_iterator + 1;
12     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
```

A.11 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.