Universidade de São Paulo
Instituto de Física de São Carlos
Departamento de Física e Informática
Grupo de Física Computacional e Instrumentação Aplicada

RENATO FABBRI

Música no áudio digital: descrição psicofísica e caixa de ferramentas

São Carlos

2013

RENATO FABBRI

Música no áudio digital: descrição psicofísica e caixa de ferramentas

Dissertação apresentada ao Programa de Pós-graduação em Física do Instituto de Física de São Carlos da Universidade de São Paulo, para a obtenção do título de Mestre em Ciências.

Área de Concentração: Física Aplicada

Orientador: Prof. Dr. Osvaldo Novais de Oliveira Junior
Colaborador: Prof. Dr. Luciano da Fontoura Costa

Versão Original

São Carlos

2013

Ficha catalográfica elaborada pelo Serviço de Biblioteca e Informação do IFSC,
com os dados fornecidos pelo(a) autor(a)

Fabbri, Renato
Música no áudio digital: descrição psicofísica e caixa de ferramentas / Renato Fabbri; orientador Osvaldo Novais Oliveira Junior -- São Carlos, 2013.
213 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Física Aplicada Computacional) -- Instituto de Física de São Carlos, Universidade de São Paulo, 2013.

1. Som. 2. Áudio digital. 3. Música. 4. Psicofísica. 5. Arte e tecnologia. I. Oliveira Junior, Osvaldo Novais, orient. II. Título.

AGRADECIMENTOS

Agradeço ao Prof. Luciano da Fontoura Costa e ao Prof. Osvaldo Novais de Oliveira Junior pela oportunidade e pela orientação deste trabalho.

Agradeço ao corpo de funcionários do IFSC, pela prestatividade e eficiência em todos os momentos que precisei.

Agradeço aos Prof. Rafael Santos Mendes e Prof. Adolfo Maia Junior pelas orientações na gênese deste trabalho.

Agradeço à minha família, em especial à minha esposa Thaís Teixeira Fabbri e ao meu filho Antônio Anzoategui Fabbri.

Agradeço ao meu irmão Ricardo Fabbri e ao amigo Vilson Vieira da Silva Junior pelas profícuas colaborações acadêmicas, artísticas e socialmente engajadas.

Agradeço aos participantes do labMacambira.sf.net pelas colaborações diretas e indiretas nos *software*, apresentações e articulações desde junho de 2011. Em especial agradeço ao Daniel Penalva, Caleb Macarenhas, Chico Simões, Fábio Simões, Daniel Marostegan, Marcos Mendonça, Geraldo Magela Rocha, Guilherme Lunhani, Patrícia Ferraz, Glerm Soares, Danilo Shiga, Edson "Presto" Corrêa, Vanessa Ferreira e Sérgio Teixeira de Carvalho. Agradeço aos demais colaboradores das listas de email, IRC, AA e outros canais.

Agradeço ao Cultura Viva e aos pontos de cultura pelo apoio fundamental a este e outros trabalhos. Agradeço em especial a estes Pontões e Cultura Digital e Ação Griô pelo suporte a este trabalho em ações regionais e nacionais: CDTL (PE), JuntaDados.org (BA), Nós Digitais (SP), Casa dos Meninos (SP), Nina Griô (SP), Pontão da Eco (RJ), Casa de Cultura Tainã (SP) e à Comissão Nacional dos Pontos de Cultura - CNPdC (GO).

Agradeço às comunidades de cultura e software livre por todos os conhecimentos e tecnologias repassados e que compõem esta contribuição.

“Music is a hidden arithmetic exercise of the soul,
which does not know that it is counting.”
— Gottfried Leibniz (1646-1716)

“Music is a hidden metaphysical exercise of the soul,
which does not know that it is philosophizing.”
— Arthur Schopenhauer (1788-1860)

“from Pantheon import Obatalá
from World import sharing, kitten

while True:
if sharing == 0:
Obatalá.kill(kitten)”
— Autoria coletiva e anônima (2010)

RESUMO

FABBRI, R. **Música no áudio digital: descrição psicofísica e caixa de ferramentas**. 2013. 213p. Dissertação (Mestrado) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2013.

A representação dos elementos básicos da música - tais como notas, ornamentos e estruturas intervalares - em termos do som discretizado é bastante utilizada em *software* e rotinas para criação musical e tratamento sonoro. Não há, entretanto, uma abordagem concisa que relacione estes elementos às amostras sonoras. Nesta dissertação, cada elemento musical é descrito por equações que resultam diretamente nas sequências temporais do som em sua representação discretizada. O elemento fundamental, a nota musical básica com duração, volume, altura e timbre, é relacionado quantitativamente às características do sinal digital. As variações internas, como *tremolos*, vibratos e flutuações espectrais, também são contempladas, o que permite sintetizar notas com inspiração nos instrumentos musicais reais além de sonoridades novas. A partir desta representação das notas, dispomos de recursos para a geração de estruturas musicais, como a métrica rítmica, os intervalos de altura e os ciclos. As equações deram origem a uma caixa de ferramentas computacionais: *scripts* que realizam cada equação e geram exemplos sonoros simples. Nomeamos MASSA, Música e Áudio em Sequências e Séries Amostrais, este ferramental e sua eficácia foi comprovada com a síntese de pequenas peças usando notas básicas, notas incrementadas e notas em música. É possível, também, sintetizar álbuns inteiros através de colagens dos *scripts* e parametrização especificada pelo usuário. Com o paradigma de implementação em código aberto, a (caixa de ferramentas) MASSA pode ser expandida em processos de co-autoria e usada livremente por músicos, engenheiros e outros interessados. De fato, o sistema já foi empregado por usuários externos para a produção de músicas, apresentações artísticas, experimentos psico-acústicos e a difusão da linguagem computacional através do apelo lúdico dos artefatos audiovisuais.

Palavras-chave: Som. Áudio digital. Música. Psicofísica. Arte e tecnologia.

ABSTRACT

FABBRI, R. **Music on digital audio: psychophysical description and toolbox**. 2013. 213p. Dissertação (Mestrado) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2013.

The representation of the basic elements of music - such as notes, ornaments and intervalar structures - in terms of discrete audio signal is often used in software for music creation and design. Nevertheless, there is no unified approach that relates these elements to the sound discrete samples. In this dissertation, each musical element is described by equations that represent the sonic time samples, which are then implemented in scripts within a software toolbox, referred to as MASSA (Music and Audio in Sequences and Samples). The fundamental element, the musical note with duration, volume, pitch and timbre, is related quantitatively to the characteristics of the digital signal. Internal variations, such as tremolos, vibratos and spectral fluctuations, are also considered, which enables the synthesis of notes inspired by real instruments and new sonorities. With this representation of notes, resources are provided for the generation of musical structures, such as rhythmic meter, pitch intervals and cycles. The efficacy of MASSA was confirmed by the synthesis of small musical pieces using basic notes, incremented notes and notes in music. It is possible to synthesize whole albums through collage of the scripts and parameterization specified by the user. With the paradigm of open source implementation, MASSA toolbox can be expanded in co-authorship processes and used freely by musicians, engineers and other interested parties. In fact, MASSA has already been employed by external users for diverse purposes which include music production, artistic presentations, psychoacoustic experiments and computer language diffusion where the appeal of audiovisual artifacts is exploited.

Keywords: Sound. Digital audio. Music. Psychophysics. Art and technology.

LISTA DE FIGURAS E TABELAS

Figura 1.1 - Som digital em modulação por código de pulsos (PCM): 25 amostras representadas por 4 bits cada uma. 31

Figura 2.1 - Formas de onda musicais básicas. As formas de onda sintéticas estão em (a) e as formas de onda reais estão em (b). 40

Figura 2.2 - Espectros das ondas sonoras musicais artificiais básicas. As senoide tem o especro puntual, a triangular apresenta somente os harmônicos ímpares, caindo a 6dB por oitava; a onda quadrada tem somente os harmônicos ímpares, caindo a 12dB por oitava; a onda dente de serra apresenta todos os harmônicos, caindo a 6dB por oitava. 41

Figura 2.3 - Espectros das ondas sonoras de uma nota de oboé natural e de período amostrado. O som natural possui flutuações nos harmônicos e ruídos, já o som de período amostrado possui espectro perfeitamente harmônico. 43

Figura 2.4 - Oscilação de 2 amostras (frequência máxima em qualquer f_a). O primeiro coeficiente reflete o deslocamento (*offset* ou *bias*) e o segundo coeficiente especifica a amplitude da oscilação. . . . 45

Figura 2.5 - 3 amostras fixas apresentam uma só frequência não nula. $c_1 = c_2^*$ e $w_1 \equiv w_2$. 46

Figura 2.6 - Componentes frequenciais em 4 amostras. 47

Figura 2.7 - Formas de onda básicas em 4 amostras. 48

Figura 2.8 - Componentes frequenciais em 6 amostras: 3 senoides se somam ao *bias*. 49

Figura 2.9 - Formas de onda básicas em 6 amostras: as ondas triangular e quadrada possuem os harmônicos ímpares, mas em proporções e fases diferentes; a dente de serra possui também o harmônico par. 49

Figura 2.10 - Detecção de localização espacial de fonte sonora: esquema utilizado para cálculo da diferença de tempo interaural (DTI) e da diferença de intensidade interaural (DII). 51

Figura 2.11 - Mixagem de três sequências sonoras. As amplitudes são sobrepostas diretamente. 54

Figura 2.12 - Concatenação de três sequências sonoras através da justaposição temporal de suas amostras. 55

Figura 2.13 - Procedimento de busca em tabela (conhecido como *Lookup Table*) para síntese de sons em frequências diferentes a partir de uma única forma de onda em alta resolução. 58

Figura 2.14 - Transições de intensidade para diferentes valores de α (veja equações 2.39 e 2.40). 61

Figura 2.15 - Interpretação gráfica da convolução. Cada amostra resultante é a soma das amostras anteriores de um sinal uma a uma multiplicadas pelas amostras retrógradas do outro sinal. 64

Figura 2.16 - Convolução com o impulso: deslocamento (a), linhas de delays (b) e síntese granular (c). Dispostos em ordem crescente de densidade de pulsos. 66

Figura 2.17 - Módulos da resposta em frequência (a), (b), (c) e (d) respectivamente dos filtros IIR das equações 2.45, 2.46, 2.48 e 2.49 para diferentes frequências de corte, frequências centrais e larguras de banda. 68

Figura 2.18 - Ruídos coloridos realizados através das equações 2.50, 2.51, 2.52, 2.53, 2.54: espectros e ondas sonoras resultantes. 71

Figura 2.19 - Espectrograma de um som com vibrato senoidal de $3Hz$ e profundidade de uma oitava em uma dente de serra de $1000Hz$ (considerada $f_a = 44.1kHz$). 75

Figura 2.20 - Tremolo de profundidade $V_{dB} = 12dB$ com padrão oscilatório de uma dente de serra em $f' = 1.5Hz$ em uma senoide de $f = 40Hz$ (considerada taxa de amostragem $f_a = 44,1kHz$). 77

Figura 2.21 - Envoltória ADSR (*Attack, Decay, Sustain, Release*) e uma sequência sonora arbitrária submetida à envoltória. A variação linear de amplitude está acima. Abaixo a variação de amplitude é exponencial. 84

Tabela 2.22 - Resumo das funções harmônicas tonais para a escala maior. A tônica é o centro da música, a dominante tende à tônica e a subdominante se distancia da tônica. Os três acordes podem, a princípio, serem substituídos livremente pela relativa ou antirelativa. 92

Figura 2.23 - Movimentos diferenciados pelo contraponto. Visando preservar a independência entre as vozes, são distinguidos 3 tipos de movimentos. 94

Tabela 2.24 - Transição das durações ouvidas individualmente para alturas. . . 95

Figura 2.25 - Divisões e aglomerações do pulso musical para estabelecimento de métrica. Ao lado esquerdo estão as divisões da semínima estabelecida como pulso. Ao lado direito, fórmulas de compasso que especificam as mesmas métricas, mas na escala das aglomerações do pulso musical. 96

Figura 2.26 - Distinções canônicas do clímax musical em uma melodia e outros domínios. 99

Tabela 2.27 - Change Ringing: *Peal* (padrão) com 3 sinos 101

LISTA DE RELAÇÕES ANALÍTICAS DESCRITAS E IMPLEMENTADAS COMPUTACIONALMENTE NO APÊNDICE A

Equação 2.1 - Sequência de amostras de um áudio PCM.

Equação 2.2 - Potência.

Equação 2.3 - Decibels entre dois áudios.

Equação 2.4 - Amplitude dobrada em decibels.

Equação 2.5 - Potência dobrada em decibels.

Equação 2.7 - Variação de amplitude no volume dobrado (10 dBs).

Equação 2.8 - Conversão de decibels em variação de amplitude.

Equação 2.9 - Representação básica de uma sequência periódica.

Equação 2.10 - Sequência infinita senoidal.

Equação 2.11 - Sequência infinita de uma onda 'dente de serra'.

Equação 2.12 - Sequência infinita de uma onda 'triangular'.

Equação 2.13 - Sequência infinita de uma onda 'quadrada'.

Equação 2.14 - Sequência infinita de uma forma de onda *sampleada*.

Equação 2.15 - Recomposição das amostras temporais com base nos coeficientes espectrais.

Equação 2.16 - Recomposição das amostras temporais reais em termos do módulo e fase.

Equação 2.17 - Número de pares de coeficientes relativos à mesma frequência.

Equação 2.18 - Coeficientes equivalentes em termos das frequêcias que representam.

Equação 2.19 - Equivalências dos módulos dos coeficientes espectrais.

Equação 2.20 - Equivalências das fases dos coeficientes espectrais.

Equação 2.21 - Reconstrução das amostras temporais em termos dos coeficientes pareados e independentes.

Equação 2.22 - Nota básica com duração, frequência e altura.

Equação 2.23 - Forma de onda sintética ou amostrada.

Equação 2.24 - Nota básica com forma de onda especificada.

Equação 2.25 - Distância de uma fonte sonora a cada ouvido.

Equação 2.26 - Diferença de Tempo Interaural (DTI).

Equação 2.27 - Diferença de Intensidade Interaural (DII).

Equação 2.28 - Sequência binaural PCM com DTI e DII para localização espacial.

Equação 2.29 - Ângulo resolvido pela implementação da DTI e DII.

Equação 2.30 - Mixagem de N sequências.

Equação 2.31 - Concatenação de N sequências.

Equação 2.32 - Procedimento de busca em tabelas (*Lookup Table*).

Equação 2.33 - Frequências em cada amostra em uma variação linear.

Equação 2.34 - Índices para busca na tabela em uma variação linear de frequência.

Equação 2.35 - Sequência amostral em uma variação linear de frequências.

Equação 2.36 - Frequências em cada amostra em uma variação exponencial.

Equação 2.37 - Índices para busca na tabela em uma variação exponencial de frequência.

Equação 2.38 - Sequência amostral em uma variação exponencial de frequências.

Equação 2.39 - Sequência de amplitudes em uma variação exponencial.

Equação 2.40 - Sequência amostral em uma variação exponencial de amplitude.

Equação 2.41 - Sequência de amplitudes em uma variação linear de amplitude.

Equação 2.42 - Sequência amostral em uma variação exponencial de amplitude dada em decibels.

Equação 2.43 - Convolução de sequências reais e finitas.

Equação 2.44 - Equação a diferenças para aplicação temporal de filtros IIR.

Equação 2.45 - Coeficientes de um filtro passa-baixas bem comportado de primeira ordem com frequência de corte variável.

Equação 2.46 - Coeficientes de um filtro passa-altas bem comportado de primeira ordem com frequência de corte variável.

Equação 2.47 - Variáveis auxiliares de um filtro nó de segunda ordem.

Equação 2.48 - Coeficientes de um filtro passa-banda de segunda ordem com frequência central e largura de banda variáveis.

Equação 2.49 - Coeficientes de um filtro rejeita-banda de segunda ordem com frequência central e largura de banda variáveis.

Equação 2.50 - Coeficientes espectrais para síntese de ruído branco.

Equação 2.51 - Coeficientes espectrais para síntese de ruído rosa.

Equação 2.52 - Coeficientes espectrais para síntese de ruído marrom.

Equação 2.53 - Coeficientes espectrais para síntese de ruído azul.

Equação 2.54 - Coeficientes espectrais para síntese de ruído violeta.

Equação 2.55 - Coeficientes espectrais para síntese de ruído preto.

Equação 2.56 - Índices auxiliares para um vibrato de frequência variável.

Equação 2.57 - Expoentes auxiliares para um vibrato de frequência variável.

Equação 2.58 - Frequências por amostra de um vibrato de frequência e profundidade variáveis.

Equação 2.59 - Índices um vibrato de frequência e profundidade variáveis.

Equação 2.60 - Amostras de um áudio com vibrato de frequência e profundidade variáveis.

Equação 2.61 - Amplitudes por amostra de um tremolo de frequência e profundidade variáveis.

Equação 2.62 - Amostras de um áudio com tremolo de frequência e profundidade variáveis.

Equação 2.63 - Espectro da síntese FM.

Equação 2.65 - Espectro da síntese AM.

Equação 2.66 - Índices auxiliares para síntese FM.

Equação 2.67 - Sequência auxiliar para síntese FM.

Equação 2.68 - Sequência de frequências para cada amostra para síntese FM.

Equação 2.69 - Índices para síntese FM.

Equação 2.70 - Sequência amostral resultante da síntese FM.

Equação 2.71 - Sequência de amplitudes por amostra na síntese AM.

Equação 2.72 - Sequência amostral resultante da síntese AM.

Equação 2.73 - Exemplo de vínculo entre a frequência e parâmetros do tremolo e do vibrato em um som.

Equação 2.74 - Envoltória ADSR com variações lineares e logarítmicas.

Equação 2.75 - Aplicação da envoltória ADSR em uma sequência arbitrária.

Equação 2.76 - Intervalos musicais em número de semitons.

Equação 2.77 - Escalas simétricas na oitava divida em 12 notas.

Equação 2.78 - Escalas diatônicas.

Equação 2.79 - Sucessão de intervalos em uma escala diatônica.

Equação 2.80 - Escalas menores natural, harmônica e melódica.

Equação 2.81 - Tríades maior, menor, diminuta e aumentada.

Subseção 2.3.1 - Relações microtonais através de frações de semitons ou através de quantidades inteiras de divisões arbitrárias da oitava.

Subseção 2.3.2 - Relações básicas de harmonia tonal.

Subseção 2.3.3 - Regras básicas de condução de vozes com independência.

Subseção 2.3.4 - Relações básicas de métrica e rítmica.

Subseção 2.3.5 - Formação de arcos musicais através de estruturas direcionais.

Subseção 2.3.6 - Estruturas cíclicas para a síntese musical.

Sumário

1 Introdução **29**

1.1 Som em áudio digital 29

1.2 Arte sonora e teoria musical 31

1.3 Implementação computacional 32

1.4 Objetivos 32

2 Música no som digitalizado **35**

2.1 Caracterização da nota musical em tempo discreto 35

2.1.1 Duração 36

2.1.2 Volume 36

2.1.3 Altura 38

2.1.4 Timbre 38

2.1.5 O espectro no som amostrado 43

2.1.6 A nota básica 50

2.1.7 Localização espacial 51

2.1.8 Usos musicais 53

2.2 Variações na nota musical básica 57
2.2.1 Tabela de busca 57
2.2.2 Variações incrementais de frequência e intensidade 59
2.2.3 Aplicação de filtros digitais 63
2.2.4 Ruídos 70
2.2.5 Tremolo e vibrato, AM e FM 74
2.2.6 Usos musicais 80
2.3 Organização de notas em música 85
2.3.1 Afinação, intervalos, escalas e acordes 85
2.3.2 Rudimentos de harmonia 91
2.3.3 Contraponto 94
2.3.4 Ritmo 95
2.3.5 Estruturas direcionais 98
2.3.6 Estruturas cíclicas 100
2.3.7 Idioma musical? 102
2.3.8 Usos musicais 103

3 Conclusões e trabalhos futuros **105**

REFERÊNCIAS **107**

Apêndice A – Código computacional dos procedimentos expostos no capítulo 2 **113**

A.1 Código Python das relações descritas na seção 2.1 113

A.2 Código Python das relações descritas na seção 2.2 118

A.3 Código Python das relações descritas na seção 2.3 128

Apêndice B – Código Computacional das Peças Musicais **135**

B.1 Peças referentes à seção 2.1 136

B.1.1 Quadros sonoros 136

B.1.2 Reduced-fi 139

B.2 Peças referentes à seção 2.2 141

B.2.1 Transita para metro 141

B.2.2 Vibra e treme 145

B.2.3 Tremolos, vibratos e a frequência 145

B.2.4 Trenzinho de caipiras impulsivos 145

B.2.5 Ruidosa faixa 145

B.2.6 Bela Rugosi 151

B.2.7 Chorus infantil 154

B.2.8 ADa e SaRa 154

B.3 Peças referentes à seção 2.3 155

B.3.1 Intervalos entre alturas 155

B.3.2 Cristais 155

B.3.3 Micro tom 158

B.3.4 Acorde cedo 161
B.3.5 Conta ponto 161
B.3.6 Poli Hit Mia 161
B.3.7 Dirracional 161

Apêndice C – *Finite Groups in Granular and Unit Synthesis* e a síntese de um EP 165

C.1 FIGGUS 165
C.1.1 FIGGUS.py 165
C.1.2 tables.py 165
C.1.3 __init__.py 165
C.2 PPEPPS: músicas de um EP solvente 166
C.2.1 RUNME make EP MUSIC.py 166
C.2.2 Éter 167
C.2.3 Benzina 169
C.2.4 Clorofórmio 172

Apêndice D – Síntese FM e AM em escala logarítmica 177

Apêndice E – Música digital em domínios não digitais 179

E.1 Experimentos abertos em áudio: LADSPAs, Wavelets e Redes Complexas . . . 179
E.2 Áudio e música 182
E.2.1 Música em tempo diferido: *minimum-fi* e FIGGUS 182

E.2.2 Música em tempo real: *Livecoding* e ABeatTracker (ABT) 193

E.2.3 Música na matéria: EKP e AHT 200

E.2.4 Música no tecido social: Sabrina Kawahara, Audioexperiments, EstudioLivre.org, CDTL, juntaDados.org, Devolts.org, MSST, LabMacambira.sf.net 202

E.3 Materiais didáticos 205

E.3.1 Tutoriais em texto e código: python, filtros e nyquist, plugins lv2, metrics, carta mídias livres, contra-cultura digital 205

E.3.2 *Screencasts* e outros materiais em video 208

E.4 Web 209

E.4.1 Tecnologias sociais: Sítios, Conteúdos e Articulação 209

E.5 Momento atual e previsões 212

1 Introdução

"Tradicionalmente a notação musical é vista como um código através do qual sons, ideias musicais ou indicações para execução musical são registrados sob forma escrita."

Edson S. Zampronha.(1)

Representar estruturas e artifícios musicais através das características do som discretizado é a proposta deste trabalho. Os resultados são relações matemáticas e suas implementações computacionais. Uma descrição teórica está no capítulo 2 e o conjunto de *scripts* disponibilizados no Apêndice A e *online*.(2) A caixa de ferramentas (*toolbox*) resultante recebeu o nome MASSA (música e áudio em sequências e séries amostrais) e foi utilizada para fazer pequenas peças e montagens focadas nos princípios expostos. O Apêndice B possui uma relação destas montagens assim como o diretório *exemplos_de_uso* da MASSA.

1.1 Som em áudio digital

O som é uma onda mecânica longitudinal de pressão. A banda de frequências compreendida entre $20Hz$ e $20kHz$ é apreciada pelo aparelho auditivo humano com variações dependentes da pessoa, das condições climáticas e do som em si. Considerada a velocidade do som no ar

$\approx 343.2 m/s$, estes limites correspondem respectivamente aos comprimentos de onda $\frac{343.2}{20} = 17.16 m$ e $\frac{343.2}{20000} = 17.16 mm$.(3)

A percepção humana do som envolve captações pelos ossos, estômago e orelha, funções de transferência da cabeça e dorso e processamento pelo sistema nervoso. Além disso, o ouvido é um órgão dedicado à captura destas ondas. Seu funcionamento decompõe o som em seu espectro senoidal e passa para o sistema nervoso.(3) Estas componentes senoidais são cruciais para os fenômenos musicais, como se pode observar tanto na composição dos sons de interesse para a música quanto nas afinações e escalas.(4) A subseção 2.1 expõe a presença de senoides no som discretizado e caracteriza a nota musical básica.

A representação do som é o áudio[1] e este pode provir da captura do som por microfones ou da síntese. Muitas vezes, o áudio digital é especificado através de protocolos que facilitam o armazenamento e transferência dos arquivos. A representação digital do som pode consistir em amostras igualmente espaçadas no tempo e cujas amplitudes individuais são registradas com um mesmo número de *bits*. Estas amostras separadas por intervalos regulares λ_a constituem a forma padrão de representação do som em tempo discreto, chamada de modulação por código de pulsos (PCM do inglês *Pulse Code Modulation*). Um som digital PCM é caracterizado pela frequência de amostragem $f_a = \frac{1}{\lambda_a}$, também chamada de taxa de amostragem, e a profundidade de *bit* que é o número de *bits* utilizados para representar a amplitude de cada amostra. A figura 1.1 exibe 25 amostras de um áudio PCM com 4 *bits* cada. Os $2^4 = 16$ grados para a amplitude de cada amostra junto ao espaçamento regular λ_a introduzem um erro de quantização. O ruído causado por estes erros diminuem com a diminuição destes espaçamentos.(6)

Pelo teorema de Nyquist, constata-se que a metade da frequência de amostragem é a frequência máxima do sinal. Assim, para apreender as frequências audíveis, é necessária uma taxa de amostragem que seja ao menos o dobro da frequência mais aguda $f_a \geq 2 \times 20kHz = 40kHz$. Este

[1] Os termos som e áudio são muitas vezes usados de forma intercambiável.(5)

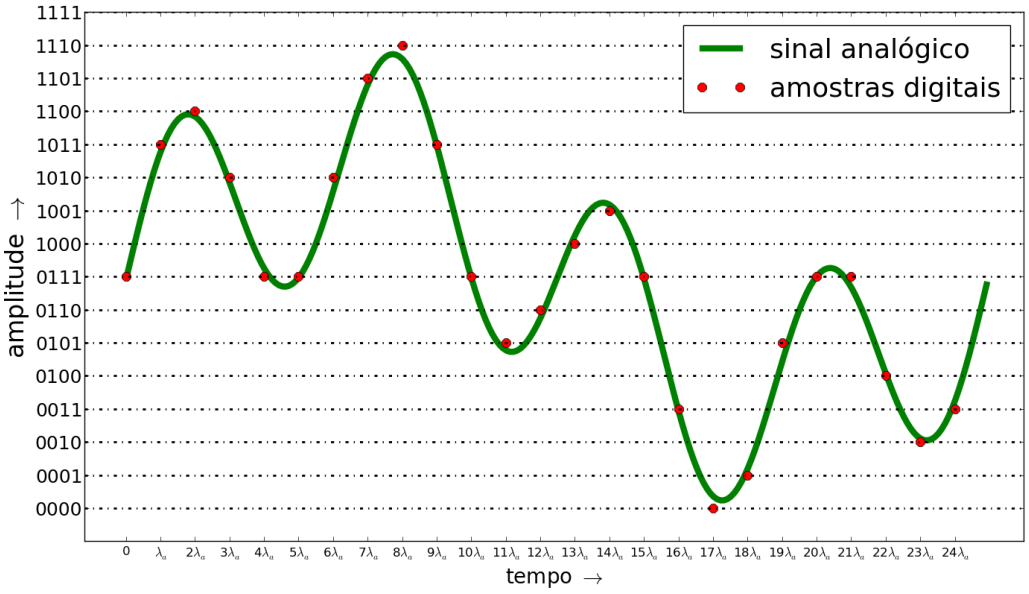

Figura 1.1 *– Som digital em modulação por código de pulsos (PCM): 25 amostras representadas por 4 bits cada uma.*

raciocínio está na base da utilização das frequências de amostragem $f_a = 44.1kHz$ e $f_a = 48kHz$, ambas padrão em *Compact Disks* (CDs) e em sistemas de Rádio e TV, respectivamente.(6)

1.2 Arte sonora e teoria musical

A música é definida como a arte manifesta pelos sons e silêncios. Para um ouvinte comum - e boa parte dos especialistas - uma 'música que seja música' pressupõe também uma métrica rítmica e organizações de alturas que formem melodias e harmonias como explicadas na seção 2.3. A música do século XX ampliou esta concepção tradicional de música. Isso ocorreu na música de concerto, especialmente nas correntes concreta, eletrônica e eletroacústica. Já na década de 90 era evidente que também a música popular, especialmente as músicas eletrônicas de dança, tinham incorporado sons sem altura definida e organizações temporais fora de métricas simples. Mesmo assim, a nota permanece paradigmática como 'unidade fundamental' das estruturas musicais e, na prática, pode se desdobrar em sons que contemplam estes desenvolvi-

mentos recentes. A nota musical e sua expansão são abordadas na seção 2.2.(7–10)

A teoria musical engloba assuntos tão diversos quanto psico-acústica, manifestações culturais e formalismos. O texto do capítulo 2 aborda estes assuntos mediante necessidade e assinala complementos externos.(11–13)

1.3 Implementação computacional

Os resultados apresentados desta dissertação incluem *scripts*, i.e. pequenos programas para melhor disponibilidade e validação das tecnologias, que constituem a caixa de ferramentas MASSA. Este conjunto de códigos está disponibilizado em domínio público através de repositórios Git públicos.(14) Os *scripts* estão em Python e fazem uso das bibliotecas externas Numpy e Scikits/Audiolab que realizam chamadas à linguagem Fortran para maior eficiência computacional. Parte deste código foi transcrito para JavaScript e Python nativos com facilidade, o que aponta para um uso destas contribuições em navegadores como o Firefox e o Chromium.(15–18)

Estas tecnologias são todas abertas, i.e. estão publicadas em licenças que permitem o uso, cópia, distribuição e utilização de quaisquer partes para estudo e geração de produtos derivados. Desta forma, o trabalho aqui descrito está disponível e facilita os processos de co-autoria[2].

1.4 Objetivos

O objetivo principal desta dissertação é apresentar de forma unificada relações entre elementos básicos da música e as sequências amostrais do áudio PCM. O capítulo seguinte é um texto

[2] A comunidade e movimento chamada *'Open Source'* entende a publicação de código computacional (e outras tecnologias) em licenças abertas como uma vantagem pragmática que facilita o desenvolvimento de *software* e apresenta vantagens pedagógicas e mercadológicas. A comunidade e movimento chamada *'Free Software'* engloba este entendimento, mas adiciona a abordagem filosófica da liberdade e compartilhamento, dando ênfase a isso. Ambas as correntes reforçam o entendimento de que o código computacional é o 'bem mais precioso produzido atualmente' pois consiste em tecnologia condensada, reativa (executa, processa ou gera resultados), modular (partes são copiadas e reutilizadas eficientemente) e replicada sem custo adicional (a cópia de texto tem custo baixíssimo).(19, 20)

conciso em que os elementos musicais são apresentados junto às amostras temporais resultantes. Para validação e compartilhamento, as implementações em código computacional destas relações e de pequenas peças musicais[3] foram reunidas em uma *toolbox* chamada MASSA e disponibilizadas *online* e parcialmente nos Apêndices A e B.

Dos objetivos secundários, destaca-se a difusão da compreensão do código computacional através de práticas lúdicas, no caso a música. Outro objetivo considerado é a apresentação de um arcabouço de síntese sonora e musical com controle amostral, para o qual há potenciais usos em experimentos psico-acústicos e síntese em alta definição (*hi-fi*). Também é considerada a apresentação destes conteúdos de forma didática, quase um tutorial, o que possibilita compreensão e uso facilitados. Esta exposição amistosa faz-se significativa pois os assuntos tratados são de reconhecida complexidade: processamento de sinais, música, psico-acústica, para citar somente alguns exemplos. Deste ponto de vista pedagógico, também se presta a apresentação destes resultados na forma de hipertexto, em que cada *script* e exemplo sonoro/musical seja acessível junto ao material teórico.

[3] Ou, de forma menos pretenciosa, montagens musicais, sequências musicais.

2 *Música no som digitalizado*

> 'The increasing dominance of graphic interfaces for music software obscured the continuing presence of the command-line tradition, the code writer, the hacker. The code writing of deferred time computer programming may be assembled out of time order, debugged and optimized.'
>
> *Simon Emmerson, Living electronic music.(21)*

2.1 Caracterização da nota musical em tempo discreto

Em diversos contextos artísticos e teóricos, a música é pensada através de unidades chamadas notas e estas unidades compreendidas como "átomos" constituintes da música.(7, 8, 22) Atualmente, estas notas são tidas como um paradigma de proposta musical e, de um ponto de vista cognitivo, como discretizações que facilitam e enriquecem o fluxo de informação através da música.(3, 23) Canonicamente, as notas possuem ao menos duração, volume, altura e timbre.(23) Estas são qualidades tratáveis quantitativamente.(3) Assim, as características da nota musical digital básica são dadas pelas amostras igualmente espaçadas no tempo da onda mecânica que representam (veja seção 1.1 sobre áudio PCM).

Todas as relações desta seção estão no Apêndice A.1, as montagens musicais *Quadros sonoros* e *Reduced-fi* estão nos Apêndices B.1.1 e B.1.2. Estas implementações estão também disponíveis online como parte do *toolbox* MASSA.(2)

2.1.1 Duração

A frequência (ou taxa) de amostragem f_a é definida como o número de amostras por segundo. Seja a sequência $T_i = \{t_i\}$ um conjunto ordenado de amostras reais separadas por $\delta_a = 1/f_a$ segundos. Uma nota musical de duração Δ se apresenta como uma sequência de $\lfloor \Delta.f_a \rfloor$ amostras[1]:

$$T_i^{\Delta} = \{t_i\}_{i=0}^{\lfloor \Delta.f_a \rfloor - 1} \tag{2.1}$$

Seja $\Lambda = \lfloor \Delta.f_a \rfloor$ o número de amostras da sequência, de forma que $T_i = \{t_i\}_0^{\Lambda-1}$.

2.1.2 Volume

A sensação de volume sonoro depende da reverberação e distribuição dos harmônicos, dentre outras características trabalhadas na seção 2.2. Pode-se obter variações do volume através da potência da onda (24):

$$pot(T_i) = \frac{\sum_{i=0}^{\Lambda-1} t_i^2}{\Lambda} \tag{2.2}$$

O volume final dependerá sempre da amplificação do sinal nos alto-falantes, assim o crucial é a potência relativa de uma nota em relação às outras ou de um trecho da música em relação ao resto. As diferenças de volume são medidas em decibels, e estes são calculados diretamente com as amplitudes através das energias ou potências[2]:

$$V_{dB} = 10 log_{10} \frac{pot(T_i^{'})}{pot(T_i)} \tag{2.3}$$

[1] O limite superior de uma sequência é um número natural, mas $\Delta.f_a$ só satisfaz esta condição em casos muito excepcionais. É necessário escolher um inteiro próximo de $\Delta.f_a$ e admitir algum erro. Por simplicidade, será considerada sempre a parte inteira da multiplicação, descrita por $\lfloor \Delta.f_a \rfloor$ e aceito o erro de até $-\delta_a$ segundos. Por exemplo, $\delta_a = 1/44100 \approx 2,3.10^{-5}$, por volta de 23 microssegundos, o que é razoável para usos musicais.

[2] Lembrando que, devido à percepção logarítmica, em um som de volume v a redução da potência p para uma mesma fração $v.p$ com $v \in [0,1]$ é sentido como a mesma diminuição κ do volume $v - \kappa$ com $\kappa \geq 0$.

A quantidade V_{dB} possui a unidade decibel (dB). A cada 10 dB se atribui a sensação de "volume dobrado". Importantes resultados de referência são os equivalentes em dB de se dobrar a amplitude ou a potência:

$$se \quad t_i^{'} = 2.t_i \quad \Rightarrow \quad pot(T_i^{'}) = 4.pot(T_i) \quad \Rightarrow \quad V_{dB}^{'} = 10log_{10}4 \quad \approx \quad 6dB \tag{2.4}$$

$$se \quad pot(T_i^{'}) = 2pot(T_i) \quad \Rightarrow \quad V_{dB}^{'} = 10log_{10}2 \quad \approx \quad 3dB \tag{2.5}$$

Outro valor de referência é o ganho de amplitude necessário para que uma sequência tenha o volume dobrado (10dB a mais):

$$10log_{10}\frac{pot(T_i^{'})}{pot(T_i)} = 10 \quad \Rightarrow \quad \sum_{i=0}^{\lfloor \Delta.f_a \rfloor - 1} t_i^{'2} = 10\sum_{i=0}^{\Lambda-1} t_i^2 = \sum_{i=0}^{\Lambda-1} (\sqrt{10}.t_i)^2 \tag{2.6}$$

$$\therefore \quad t_i^{'} = \sqrt{10}t_i \quad \Rightarrow \quad t_i^{'} \approx 3,16t_i \tag{2.7}$$

Ou seja, é necessário pouco mais que triplicar a amplitude para um volume drobrado. Estes valores servem de guia para os aumentos e diminuições dos valores absolutos que compõem as sequências de amostras sonoras com propósitos musicais. A conversão direta de decibels em ganho ou atenuação de amplitude se dá da seguinte forma:

$$A = 10^{\frac{V_{dB}}{20}} \tag{2.8}$$

Onde A é o fator multiplicativo que relaciona as amplitudes do sinal antes e depois da amplificação.

2.1.3 Altura

Recapitulando, a partícula musical (nota) é uma sequência T_i cuja duração e volume correspondem ao tamanho da sequência e amplitude de suas amostras. A altura é especificada pela frequência fundamental f_0 com ciclo de duração $\delta_{f_0} = 1/f_0$. Esta duração multiplicada pela frequência de amostragem f_a resulta no número de amostras do ciclo $\lambda_{f_0} = f_a.\delta_{f_0} = f_a/f_0$.

Por motivos didáticos, seja f_0 tal que divida f_a e λ_{f_0} resulte inteiro. Se T_i^f é uma sequência sonora de frequência fundamental f, então:

$$T_i^f = \left\{t_i^f\right\} = \left\{t_{i+\lambda_f}^f\right\} = \left\{t_{i+\frac{fa}{f}}^f\right\} \tag{2.9}$$

Na seção seguinte serão contempladas frequências f que não dividem f_a e esta restrição não implica na perda de generalidade do conteúdo desta seção.

2.1.4 Timbre

Enquanto o período da onda corresponde a uma frequência fundamental, o percurso da onda sonora dentro do período - chamado de forma de onda - define um espectro harmônico e portanto um timbre[3]. Musicalmente, importa que espectros sonoros com diferenças mínimas resultam em timbres com diferenças expressivamente cruciais e que, portanto, pode-se produzir timbres diferentes através de espectros diferentes.(3)

O caso mais simples (e mais importante, como mostra o texto que segue) é o do espectro que consiste somente em sua própria fundamental f. Este é o caso da senoide, frequência em

[3] O timbre é uma característica subjetiva e complexa. Fisicamente, o timbre é multidimensional e dado pelo comportamento temporalmente dinâmico de energias em componentes espectrais tanto harmônicas quanto ruidosas. Além disso, a palavra *timbre* é utilizada para designar coisas diferentes: uma mesma nota possui diferentes timbres, um mesmo instrumento possui diferentes timbres, dois instrumentos da mesma família possuem o mesmo timbre que a caracteriza mas possuem timbres diferentes porque são instrumentos diferentes. Vale salientar que nem tudo o que se atribui ao timbre se acha manifesto em diferenças espectrais e que até aspectos culturais ou circunstanciais alteram nossa percepção do timbre.

movimento oscilatório puro chamado movimento harmônico simples. Seja S_i^f uma sequência cujas amostras s_i^f descrevem uma senoide de frequência f:

$$S_i^f = \{s_i^f\} = \left\{\sin(2\pi\frac{i}{\lambda_f})\right\} = \left\{\sin(2\pi f\frac{i}{f_a})\right\} \tag{2.10}$$

Onde $\lambda_f = \frac{f_a}{f} = \frac{\delta_f}{\delta_a}$ é o número de amostras do período[4].

De forma semelhante, outras formas de onda são utilizadas na música por suas qualidades espectrais e simplicidade. Enquanto a senoide é um ponto isolado no espectro, estas ondas apresentam cadeias de componentes harmônicas. As formas de onda especificadas nas equações 2.10, 2.11, 2.12 e 2.13 estão na figura 2.1. São as formas de onda artificiais tradicionalmente usadas na música para síntese e controle oscilatório de variáveis e apresentam diversos usos também fora da música.(25)

A dente de serra apresenta todas as componentes da série harmônica com energia decrescente de $-6dB/oitava$. A sequência de amostras temporais pode ser descrita da seguinte forma:

$$D_i^f = \left\{d_i^f\right\} = \left\{2\frac{i\%\lambda_f}{\lambda_f} - 1\right\} \tag{2.11}$$

A forma de onda triangular apresenta somente os harmônicos ímpares caindo a $-12dB/oitava$:

$$T_i^f = \left\{t_i^f\right\} = \left\{1 - \left|2 - 4\frac{i\%\lambda_f}{\lambda_f}\right|\right\} \tag{2.12}$$

A onda quadrada apresenta somente os harmônicos ímpares caindo a $-6dB/oitava$:

$$Q_i^f = \left\{q_i^f\right\} = \begin{cases} 1 & \text{para } (i\%\lambda_f) < \lambda_f/2 \\ -1 & \text{caso contrário} \end{cases} \tag{2.13}$$

A dente de serra é um ponto de partida comum para a síntese subtrativa pois possui ambos os

[4] Neste ponto já se tem toda a base para música *Reduced-fi* do Apêndice B.1.2.

harmônicos pares e ímpares e em grande quantidade. Para fins musicais, estas formas de onda são excessivamente ricas em harmônicos agudos e uma filtragem atenuante nos médios e agudos é útil para que o som ganhe naturalidade e fique mais agradável. Os harmônicos relativamente atenuados da onda triangular a faz a mais funcional - dentre as citadas - para ser usada sem nenhum tratamento na síntese de notas musicais.

Para citar também uma utilidade da onda quadrada, ela pode ser usada em uma síntese subtrativa que vise a imitar um clarinete. Este instrumento também só apresenta os componentes ímpares do espectro harmônico e a onda quadrada convém com sua energia abundante nas altas frequências.

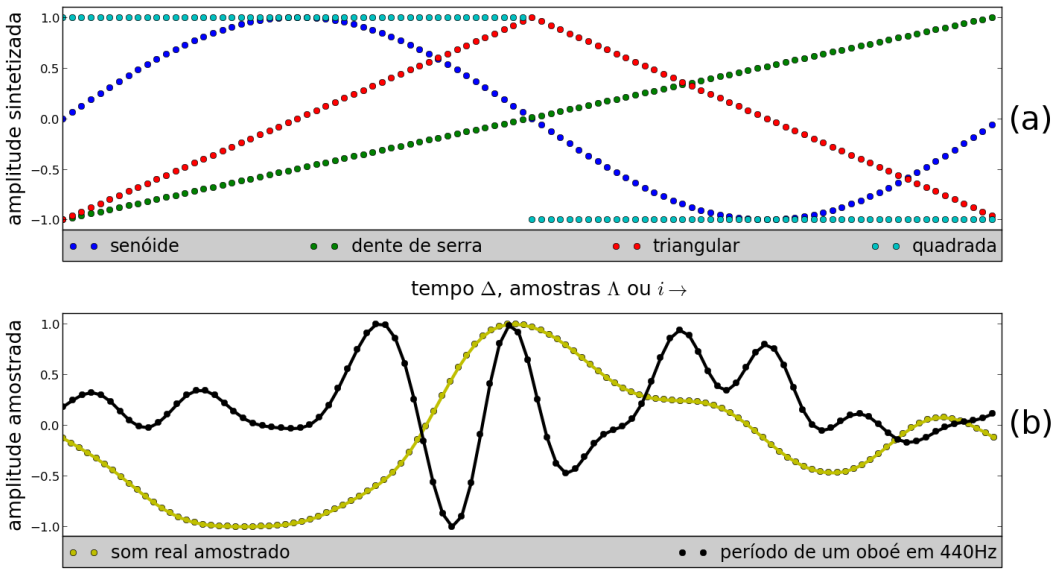

Figura 2.1 *– Formas de onda musicais básicas. As formas de onda sintéticas estão em (a) e as formas de onda reais estão em (b).*

A figura 2.1 apresenta as formas de onda descritas nas equações 2.10, 2.11, 2.12 e 2.13 para $\lambda_f = 100$ (período de 100 amostras). Se $t_a = 44,1kHz$, como no padrão PCM de *Compact Disks*, a onda possui frequência fundamental $f = \frac{f_a}{\lambda_f} = \frac{44100}{100} = 441$ *Herz*. Um lá[5], independente

[5] Um lá 4, logo acima do dó central, no segundo espaço do pentagrama na clave de sol comum.

da forma de onda.

O espectro de cada forma de onda básica está na figura 2.2. As componentes isoladas e exatamente harmônicas dos espectros correspondem a um período rigorosamente fixo. A senoide consiste de um nódulo único no espectro, frequência pura. A dente de serra é a única com a série harmônica completa (pares e ímpares). Já as ondas triangular e quadrada possuem as mesmas componentes espectrais, mas com decaimentos de $-12dB/oitava$ e $-6dB/oitava$ respectivamente.

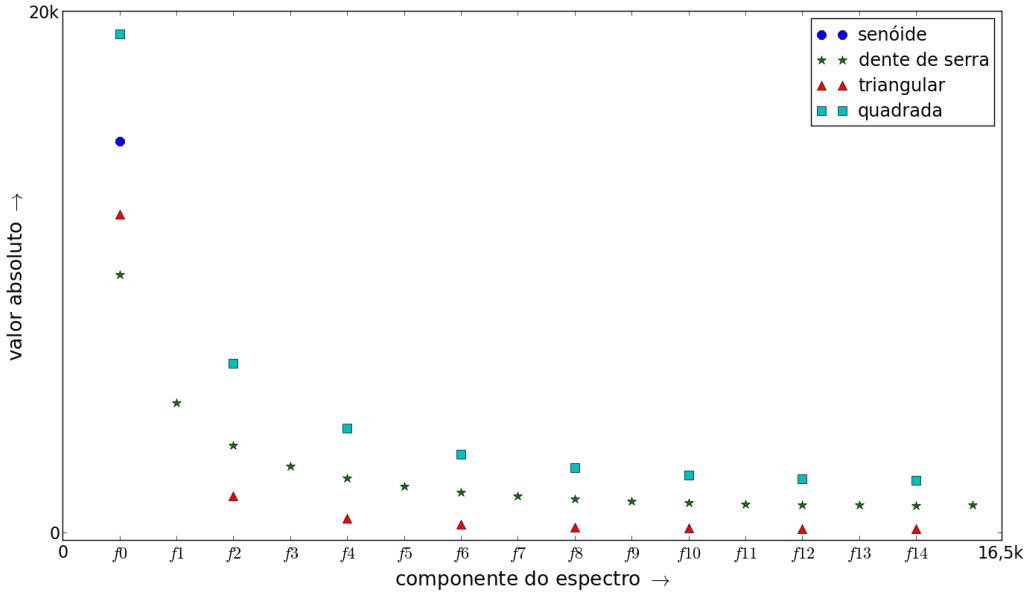

Figura 2.2 – *Espectros das ondas sonoras musicais artificiais básicas. As senoide tem o especro puntual, a triangular apresenta somente os harmônicos ímpares, caindo a 6dB por oitava; a onda quadrada tem somente os harmônicos ímpares, caindo a 12dB por oitava; a onda dente de serra apresenta todos os harmônicos, caindo a 6dB por oitava.*

O espectro harmônico é formado pelas frequências múltiplas da frequência fundamental $f_n = (n+1).f_0$. Como a percepção humana de altura segue uma progressão geométrica de frequências, o espectro possui notas diferentes da frequência fundamental. Além disso, o número de harmônicos será limitado pela frequência máxima $f_a/2$ (pelo Teorema de Nyquist).

Musicalmente crucial aqui é internalizar que a presença de energia em uma componente de

frequência f_n significa uma oscilação na constituição do som, puramente harmônica e naquela frequência f_n. Esta energia concentrada especificamente na frequência f_n é separada pelo ouvido para adentrar em um nível cognitivo de processamento[6]. As componentes senoidais são geralmente as principais responsáveis pela qualidade chamada timbre. Caso não se apresentem em proporções harmônicas (relações de pequenos números), o som é percebido como ruidoso ou dissonante e não com uma sonoridade de frequência fundamental estabelecida de forma unívoca. Além disso, a noção de altura absoluta em um complexo sonoro é baseada na semelhança do espectro com a série harmônica.(3)

No caso de uma forma de onda fixa e de tamanho fixo, o espectro é sempre harmônico e estático. Cada forma de onda é composta de proporções específicas das componentes harmônicas e quanto maior a curvatura do trecho na forma de onda, maior a contribuição do trecho para a concentração de energia nos harmônicos agudos. Pode-se constatar isso em sons reais. A onda rotulada como "som real amostrado" na figura 2.1 é um período de $\lambda_f = 114$ amostras extraído de um som real relativamente comportado. A onda de oboé foi amostrada de um lá 4 também em $44,1kHz$. O período escolhido para a amostragem é relativamente curto, com 98 amostras corresponde a uma frequência de $\frac{44100}{98} = 450Hz$. Pode-se perceber, através das curvaturas, o espectro rico em frequências agudas do oboé e o espectro mais grave do som real.

A sequência $R_i = \{r_i\}_0^{\lambda_f - 1}$ de amostras do som real da figura 2.1 pode ser tomada como base para um som T_i^f da seguinte forma:

$$T_i^f = \{t_i^f\} = \left\{r_{(i \% \lambda_f)}\right\} \tag{2.14}$$

O som resultante possui o espectro momentâneo do som original. Por ser repetido de forma idêntica, seu espectro é perfeitamente harmônico, sem os ruídos e variações típicas do fenô-

[6] Esta separação em frequência é realizada por diversas espécies através de mecanismos similares à cóclea humana.(3)

meno natural. Isso pode ser visto na figura 2.3, que mostra os espectros da nota original do oboé e de uma nota artificial de mesma duração e cujas amostras consistem no mesmo período da figura 2.1. O espectro natural possui variações nas frequências dos harmônicos, nas suas intensidades e uma quantidade de ruído. Já a nota cujo período foi amostrado possui espectro perfeitamente harmônico.

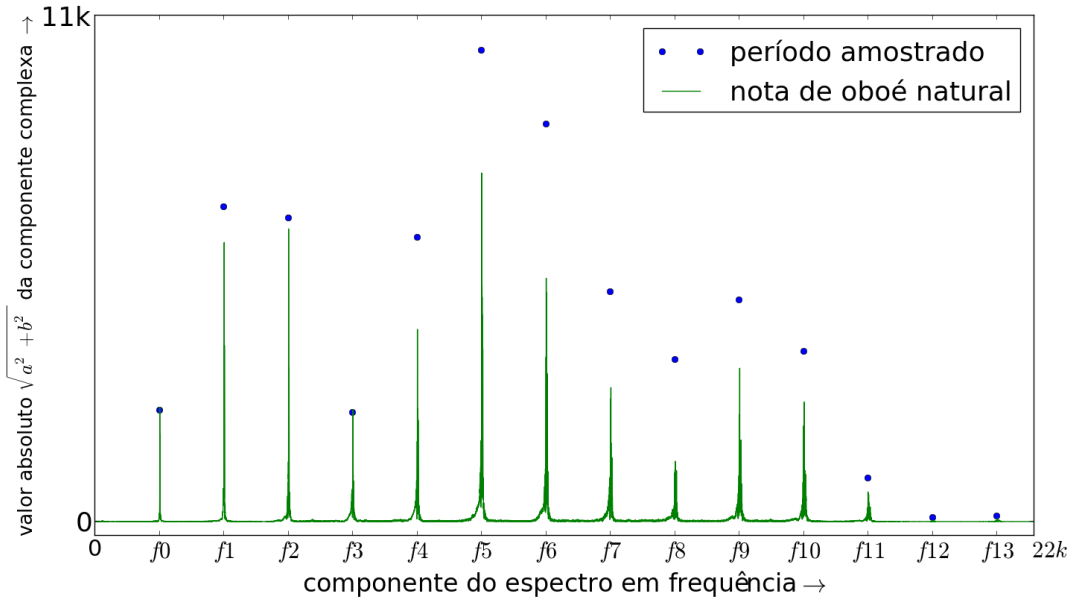

Figura 2.3 *– Espectros das ondas sonoras de uma nota de oboé natural e de período amostrado. O som natural possui flutuações nos harmônicos e ruídos, já o som de período amostrado possui espectro perfeitamente harmônico.*

2.1.5 O espectro no som amostrado

A presença e comportamento destas componentes senoidais no som discretizado possui particularidades. Considere um sinal T_i e sua decomposição de Fourier $\mathcal{F}\langle T_i\rangle = C_i = \{c_i\}_0^{\Lambda-1}$. A recomposição é a soma das componentes frequenciais em amostras temporais[7]:

[7] Lembrando que o fator $\frac{1}{\Lambda}$ pode ser distribuído dentre a transformada e a reconstrução como preferir.

$$t_i = \frac{1}{\Lambda}\sum_{k=0}^{\Lambda-1} c_k e^{j\frac{2\pi k}{\Lambda}i} = \frac{1}{\Lambda}\sum_{k=0}^{\Lambda-1}(a_k + j.b_k)\left[cos(w_k i) + j.sen(w_k i)\right] \tag{2.15}$$

Onde $c_k = a_k + j.b_k$ dita a amplitude e fase de cada frequência: $w_k = \frac{2\pi}{\Lambda}k$ em radianos ou $f_k = w_k\frac{f_a}{2\pi} = \frac{f_a}{\Lambda}k$ em Herz,com atenção para os respectivos limites em π e em $\frac{f_a}{2}$ dados pelo Teorema de Nyquist.

No caso específico de um sinal sonoro, as amostras t_i são reais e dadas pela parte real da equação 2.15:

$$\begin{aligned} t_i &= \frac{1}{\Lambda}\sum_{k=0}^{\Lambda-1}\left[a_k cos(w_k i) - b_k sen(w_k i)\right] \\ &= \frac{1}{\Lambda}\sum_{k=0}^{\Lambda-1}\sqrt{a_k^2 + b_k^2}\, cos\left[w_k i - tg^{-1}\left(\frac{b_k}{a_k}\right)\right] \end{aligned} \tag{2.16}$$

A equação 2.16 mostra que o termo imaginário de c_k acrescenta uma fase à senoide real, i.e. os termos imaginários b_k da decomposição espectral por Fourier proporcionam a varredura de fase $\left[-\frac{\pi}{2}, +\frac{\pi}{2}\right]$ dada pelo termo $tg^{-1}\left(\frac{b_k}{a_k}\right)$ que possui esta imagem. O sinal de a_k especifica o lado direito ou esquerdo do circulo trigonométrico, o que completa a varredura de fase: $\left[-\frac{\pi}{2}, +\frac{\pi}{2}\right] \cup \left[\frac{\pi}{2}, \frac{3\pi}{2}\right] \equiv [2\pi]$.

A figura 2.4 exibe duas amostras e as componentes espectrais que contêm. A decomposição de Fourier possui neste caso um único par de coeficientes $\{c_k = a_k - j.b_k\}_0^{\Lambda-1=1}$ relativos às frequências $\{f_k\}_0^1 = \left\{w_k\frac{f_a}{2\pi}\right\}_0^1 = \left\{k\frac{f_a}{\Lambda=2}\right\}_0^1 = \left\{0, \frac{f_a}{2} = f_{\text{máx}}\right\}$ com energias $e_k = \frac{(c_k)^2}{\Lambda=2}$. O papel das amplitudes a_k é nítido com $\frac{a_0}{2}$ o deslocamento fixo[8] e $\frac{a_1}{2}$ a amplitude da oscilação em si, dada pela relação $f_k = k\frac{f_a}{\Lambda=2}$. Este caso é de especial importância pois o mínimo necessário para representar uma oscilação são 2 amostras e disso resulta a frequência de Nyquist $f_{\text{máx}} = \frac{f_a}{2}$.

[8] Chamado de *bias* ou *offset*.

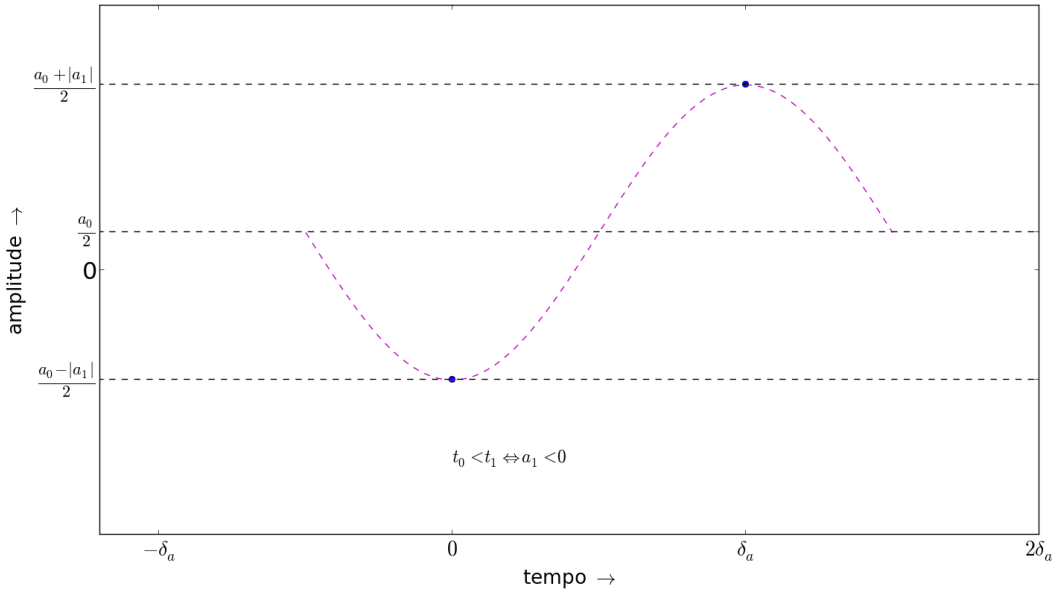

Figura 2.4 – *Oscilação de 2 amostras (frequência máxima em qualquer f_a). O primeiro coeficiente reflete o deslocamento (*offset *ou* bias*) e o segundo coeficiente especifica a amplitude da oscilação.*

Esta é a frequência máxima presente em um som amostrado com f_a amostras por segundo[9].

Todas as sequências fixas T_i de apenas 3 amostras também apresentam somente 1 frequência, pois sua primeira harmônica usaria 1.5 amostras e ultrapassa o limite inferior de 2 amostras mínimas, i.e. a frequência da harmônica ultrapassaria a de Nyquist pois: $\frac{2.f_a}{3} > \frac{f_a}{2}$. Os coeficiêntes $\{c_k\}_0^{\Lambda-1=2}$ apresentam-se em 3 componentes frequenciais. Uma delas é relativa à frequência zero (c_0), as outras duas (c_1 e c_2) contribuem de forma igual na reconstrução da senoide com $f = f_a/3$.

Λ amostras reais t_i resultam em Λ coeficientes complexos $c_k = a_k + j.b_k$. Os coeficientes c_k se equivalem dois a dois correspondendo às mesmas frequências e com contribuições idênticas[10]. Lembrando que $f_k = k\frac{f_a}{\Lambda}$, $k \in \left\{0, ..., \left\lfloor \frac{\Lambda}{2} \right\rfloor\right\}$. Quando $k > \frac{\Lambda}{2}$, a frequência f_k é espelhada em $\frac{f_a}{2}$ da seguinte forma $f_k = \frac{f_a}{2} - (f_k - \frac{f_a}{2}) = f_a - f_k = f_a - k\frac{f_a}{\Lambda} = (\Lambda - k)\frac{f_a}{\Lambda} \quad \Rightarrow \quad f_k \equiv f_{\Lambda-k}\ ,\ \forall\ k < \Lambda.$

[9] Qualquer sinal amostrado possui esta característica, não somente o som digitalizado.

[10] Parte real igual e imaginária com sinal trocado: $a_{k1} = a_{k2}$ e $b_{k1} = -b_{k2}$. Como consequência os módulos são iguais e as fases possuem sinais opostos

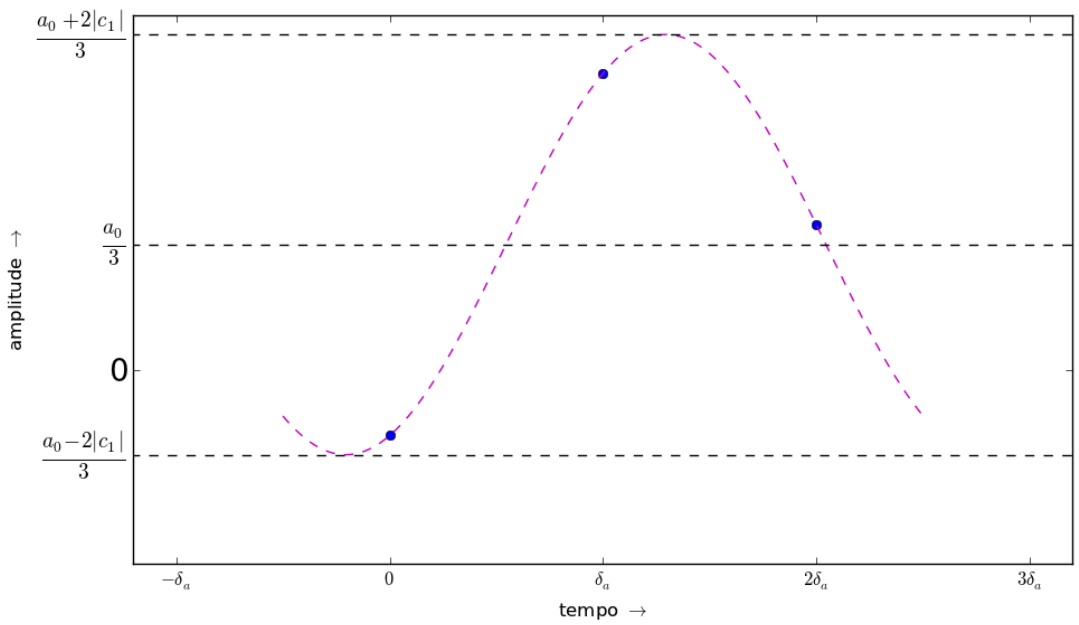

Figura 2.5 *– 3 amostras fixas apresentam uma só frequência não nula.* $c_1 = c_2^*$ *e* $w_1 \equiv w_2$.

O mesmo pode ser observado com $w_k = f_k.\frac{2\pi}{f_a}$ e lembrando da periodicidade 2π, que resulta em $w_k = -w_{\Lambda-k}$. Como o coseno é uma função par e a tangente inversa é impar, as componentes em w_k e $w_{\Lambda-k}$ se somam na equação de reconstrução das amostras reais disposta na equação 2.15.

Ou seja, em uma decomposição de Λ amostras, as Λ componentes frequenciais $\{c_i\}_0^{\Lambda-1}$ resultantes são equivalentes em pares. Exceção para f_0 e, no caso de Λ ser par, de $f_{\Lambda/2} = f_{\text{máx}} = \frac{f_a}{2}$, ambas as componentes são isoladas, i.e. não existe outra componente na frequência f_0 ou $f_{\Lambda/2}$ (se Λ par) além dela mesma. Pois $f_{\Lambda/2} = f_{(\Lambda-\Lambda/2)=\Lambda/2}$ e $f_0 = f_{(\Lambda-0)=\Lambda} = f_0$. Além disso, estas duas frequências (a frequência zero e a frequência máxima) não são representadas com variação de fase e, portanto, são estritamente reais. Assim, pode-se concluir que o número τ de pares de coeficientes equivalentes é:

$$\tau = \frac{\Lambda - \Lambda\%2}{2} + \Lambda\%2 - 1 \tag{2.17}$$

e ficam evidentes as equivalências 2.18, 2.19 e 2.20:

$$f_k \equiv f_{\Lambda-k}\ ,\ \ w_k \equiv -w_{\Lambda-k}\quad ,\qquad \forall\quad 1 \leq k \leq \tau \tag{2.18}$$

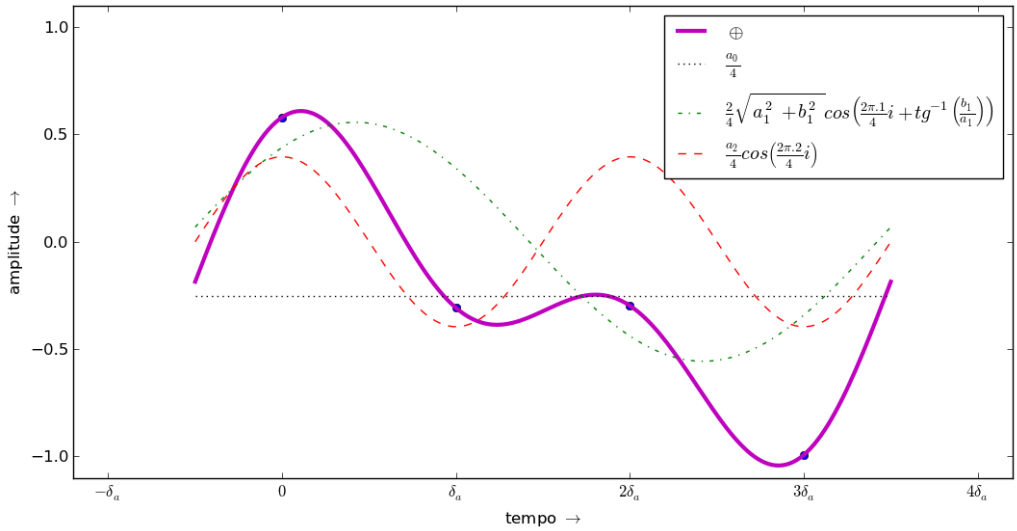

Figura 2.6 *– Componentes frequenciais em 4 amostras.*

Como $a_k = a_{\Lambda-k}$ e $b_k = -b_{\Lambda-k}$:

$$\sqrt{a_k^2+b_k^2} = \sqrt{a_{\Lambda-k}^2+b_{\Lambda-k}^2}\ ,\qquad \forall\quad 1 \leq k \leq \tau \tag{2.19}$$

$$tg^{-1}\left(\frac{b_k}{a_k}\right) = -tg^{-1}\left(\frac{b_{\Lambda-k}}{a_{\Lambda-k}}\right)\ ,\qquad \forall\quad 1 \leq k \leq \tau \tag{2.20}$$

Com $k \in \mathbb{N}$.

A observação da equação de reconstrução para o sinal real 2.16 em conjunto com as equivalências dos módulos e fases 2.19 e 2.20, o número de coeficientes pareados 2.17 e equivalência de pares de frequências 2.18 expõe o caso geral da combinação das componentes em cada amostra t_i:

$$t_i = \frac{a_0}{\Lambda} + \frac{2}{\Lambda}\sum_{k=1}^{\tau}\sqrt{a_k^2 + b_k^2}\, cos\left[w_k i - tg^{-1}\left(\frac{b_k}{a_k}\right)\right] + \frac{a_{\Lambda/2}}{\Lambda}.(1 - \Lambda\%2) \tag{2.21}$$

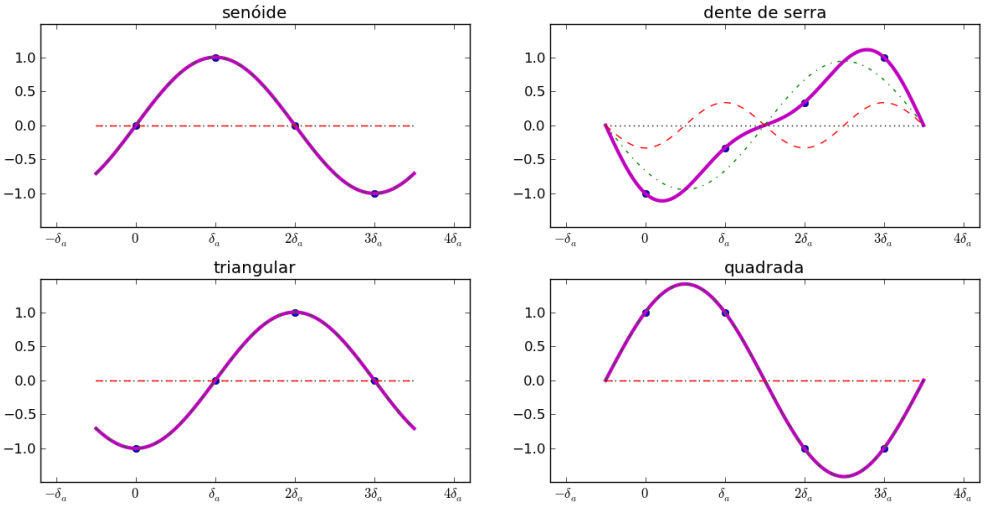

Figura 2.7 *– Formas de onda básicas em 4 amostras.*

Assim, a exemplo da figura 2.5, a transformada de Fourier de 3 amostras possui 2 coeficientes frequenciais com quantidades iguais de energia na mesma frequência.

Com 4 amostras, pode-se representar 1 ou 2 frequências em proporções quaisquer. A figura 2.6 mostra uma forma de onda de 4 amostras e suas duas componentes. As contribuições individuais se somam na forma de onda original, e uma breve inspeção revela que as curvaturas maiores são fruto da frequência mais aguda, enquanto um deslocamento fixo da somatória das componentes advém da componente na frequência zero.

A figura 2.7 explicita os harmônicos em 4 amostras nas formas de onda básicas das equações 2.10, 2.11, 2.12 e 2.13. Todas consistem em apenas 1 senoide, com exceção da dente de serra que possui os harmônicos pares.

A figura 2.8 mostra uma decomposição senoidal para o caso de 6 amostras e a figura 2.9 decompõe as formas de onda básicas. Neste caso todas as ondas se diferenciam no espectro:

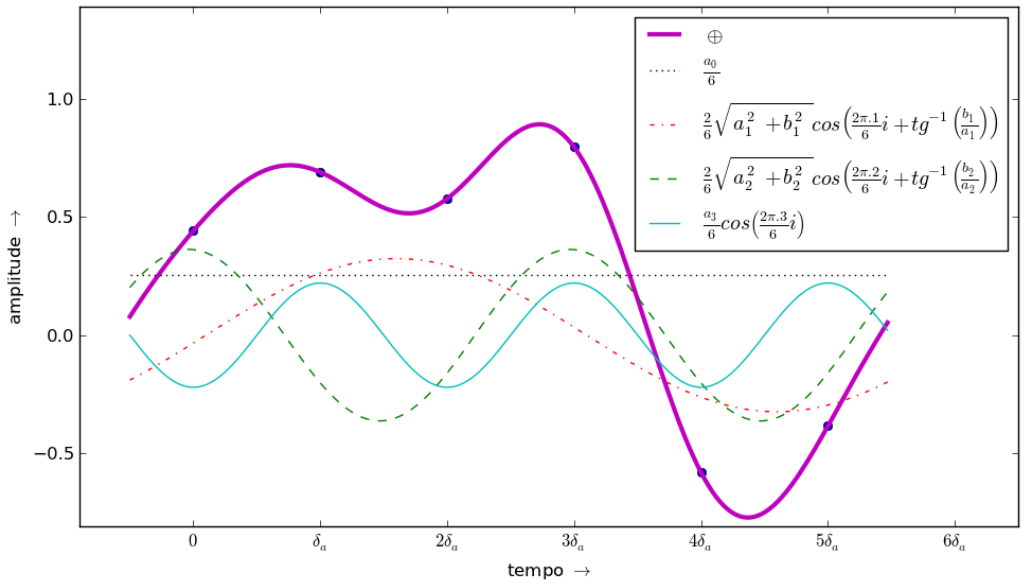

Figura 2.8 *– Componentes frequenciais em 6 amostras: 3 senoides se somam ao* bias*.*

as quadrada e triangular possuem as mesmas componentes, mas em proporções diferentes, já a dente de serra possui uma componente a mais.

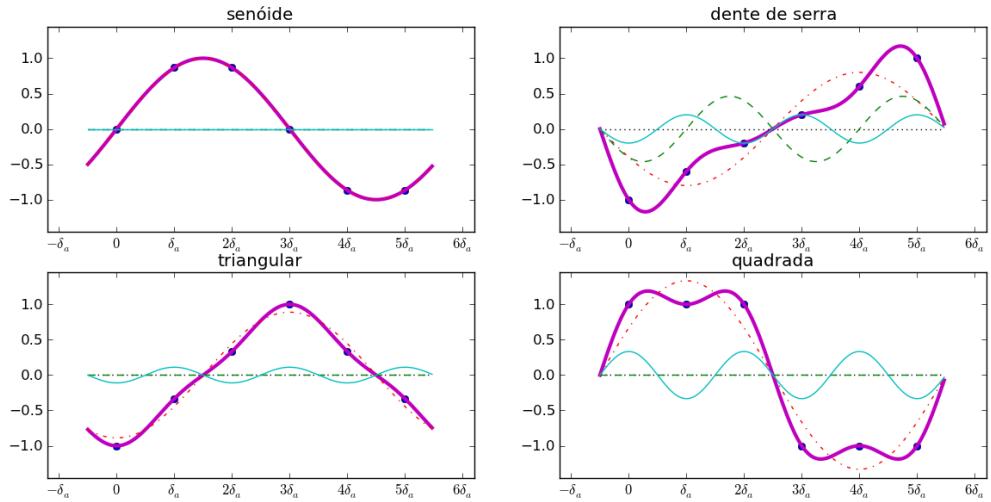

Figura 2.9 *– Formas de onda básicas em 6 amostras: as ondas triangular e quadrada possuem os harmônicos ímpares, mas em proporções e fases diferentes; a dente de serra possui também o harmônico par.*

2.1.6 A nota básica

Seja f tal que f divida f_a[11]. Uma sequência T_i de amostras sonoras separadas por $\delta_a = 1/f_a$ descreve uma nota musical de frequência f Herz e duração Δ segundos se, e somente se, possuir a periodicidade $\lambda_f = f_a/f$ e tamanho $\Lambda = \lfloor f_a.\Delta \rfloor$:

$$T_i^{f,\,\Delta} = \{t_i \%_{\lambda_f}\}_0^{\Lambda-1} = \left\{ t^f_{i\,\%\left(\frac{f_a}{f}\right)} \right\}_0^{\Lambda-1} \tag{2.22}$$

A nota por si só não especifica um timbre. Mesmo assim, faz-se necessária a escolha de uma forma de onda para que as amostras t_i tenham um valor estabelecido individualmente. Um único período dentre as ondas básicas pode ser utilizado para a especificação da nota da seguinte forma:

$\lambda_f = \frac{f_a}{f}$ é o número de amostras do período. Seja L_i^{f,δ_f} a sequência que descreve um período da onda $L_i^f \in \{S_i^f, Q_i^f, T_i^f, D_i^f, R_i^f\}$ de duração $\delta_f = 1/f$, dadas pelas equações 2.10, 2.11, 2.12 e 2.13 e onde R_i^f é uma onda real amostrada:

$$L_i^{f,\delta_f} = \left\{ l_i^f \right\}_0^{\delta_f . f_a - 1} = \left\{ l_i^f \right\}_0^{\lambda_f - 1} \tag{2.23}$$

Então a sequência T_i consistirá em uma nota de duração Δ e frequência f se:

$$T_i^{f,\,\Delta} = \left\{ t_i^f \right\}_0^{\lfloor f_a.\Delta \rfloor - 1} = \left\{ l^f_{i\,\%\left(\frac{f_a}{f}\right)} \right\}_0^{\Lambda-1} \tag{2.24}$$

[11] Como apontado anteriormente, esta limitação facilita a exposição sem perda de generalidade. A limitação será superada no início da próxima seção.

2.1.7 Localização espacial

Embora não seja uma das quatro qualidades básicas tradicionais de uma nota musical, esta possui sempre uma localização espacial. Além disso, a espacialização é atributo bastante valorizado por audiófilos e pela indústria fonográfica.(4) Acredita-se que a percepção da localização espacial do som se dê em nosso sistema nervoso através destas três informações: o atraso de chegada do som entre um ouvido e o outro, a diferença de intensidade do som direto em cada ouvido e a filtragem realizada pelo corpo, incluindo toráx, cabeça e orelhas.(3, 26, 27)

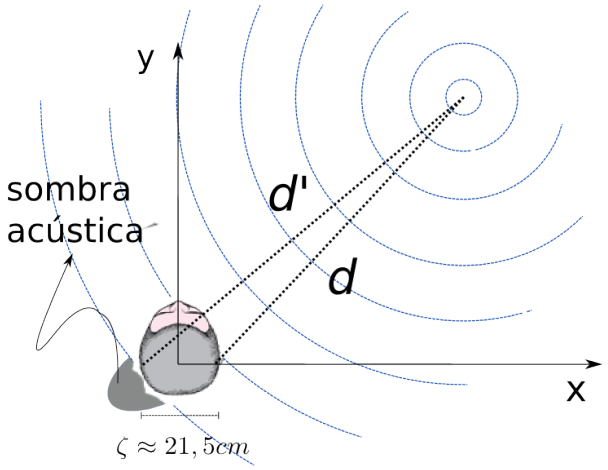

Figura 2.10 *– Detecção de localização espacial de fonte sonora: esquema utilizado para cálculo da diferença de tempo interaural (DTI) e da diferença de intensidade interaural (DII).*

Se consideradas somente as incidências diretas em cada ouvido, as equações são simples. Dada a separação ζ entre os ouvidos[12], um objeto localizado em (x,y) conforme a figura 2.10 está distante de cada ouvido:

$$\begin{aligned} d &= \sqrt{\left(x-\frac{\zeta}{2}\right)^2+y^2} \\ d' &= \sqrt{\left(x+\frac{\zeta}{2}\right)^2+y^2} \end{aligned} \tag{2.25}$$

[12] Constata-se que $\zeta \approx 21,5cm$ para um humano adulto.

e cálculos imediatos resultam na Diferença de Tempo Interaural:

$$DTI = \frac{d' - d}{v_{som\ no\ ar} \approx 343.2} \quad \text{segundos} \tag{2.26}$$

e na Diferença de Intensidade Interaural:

$$DII = 20\log_{10}\left(\frac{d}{d'}\right) \quad decibels \tag{2.27}$$

Convertendo para amplitude, obtem-se $DII_a = \frac{d}{d'}$. A DII_a pode ser utilizada como constante multiplicativa do canal direito de um sinal sonoro estéreo: $\{t'_i\}_0^{\Lambda-1} = \{DII_a.t_i\}_0^{\Lambda-1}$. Pode-se utilizar a DII junto à DTI como adiantamento no tempo do canal direito com relação ao esquerdo, vínculo crucial para a localização em sons graves e em sonoridades percussivas.(27) Considerando $\Lambda_{DTI} = \lfloor DTI.f_a \rfloor$:

$$\begin{aligned} \Lambda_{DTI} &= \left\lfloor \frac{d' - d}{343,2} f_a \right\rfloor \\ DII_a &= \frac{d}{d'} \\ \left\{t'_{(i+\Lambda_{DTI})}\right\}_{\Lambda_{DTI}}^{\Lambda+\Lambda_{DTI}-1} &= \{DII_a.t_i\}_0^{\Lambda-1} \\ \left\{t'_i\right\}_0^{\Lambda_{DTI}-1} &= 0 \end{aligned} \tag{2.28}$$

Com t_i o canal direito e t'_i o canal esquerdo. Caso $\Lambda_{DTI} < 0$, basta trocar t_i por t'_i e utilizar $\Lambda'_{DTI} = |\Lambda_{DTI}|$.

Embora consideravelmente simples até aqui, a localização espacial depende drasticamente de outras pistas. Pela DTI e DII especifica-se somente o ângulo horizontal (azimutal) θ dado por:

$$\theta = \tan^{-1}\left(\frac{y}{x}\right) \tag{2.29}$$

com x, y tais como representados na figura 2.10. Mesmo assim, há dificuldades quando θ incide sobre o chamado "cone de confusão" em que um mesmo par de especificações DTI, DII resultam de vários dos pontos do cone. Nestes pontos, a inferência do ângulo azimutal depende especialmente da filtragem atenuante nos agudos, pois a cabeça interfere um tanto mais nas ondas mecânicas agudas do que nas graves.(26, 27)

A figura 2.10 mostra também esta sombra acústica do crânio, importante para a percepção do ângulo azimutal da fonte no cone de confusão. O cone em si não foi disposto na figura pois não é exatamente um cone e suas dimensões precisas não foram encontradas na literatura visitada, mas pode ser entendido como um cone com o ápice no meio da cabeca e saindo por cada uma das orelhas.(26)

Já a localização completa, incluindo distância e elevação da fonte sonora, é dada pela função de transferência de cabeça (HRTF - do inglês *Head Related Transfer Function*).(26) Exitem bases abertas e conhecidas de HRTF como a CIPIC e pode-se aplicar estas funções de transferência em um som por convolução (veja equação 2.43).(28) O corpo do indivíduo altera bastante as filtragens realizadas e existem técnicas para gerar HRTFs que sejam - como proposta - utilizáveis de forma universal.(29)

2.1.8 Usos musicais

A partir da nota básica, cabe realizar estruturas musicais com sequências destas partículas. A soma dos elementos de mesmo índice de N sequências $T_{k,i} = \{t_{k,i}\}_{k=0}^{N-1}$ de mesmo tamanho Λ resulta em seus conteúdos espectrais sobrepostos em um processo de mixagem sonora:

$$\{t_i\}_0^{\Lambda-1} = \left\{ \sum_{k=0}^{N-1} t_{k,i} \right\}_0^{\Lambda-1} \tag{2.30}$$

A figura 2.11 ilustra este processo de superposição de ondas sonoras discretizadas. A figura

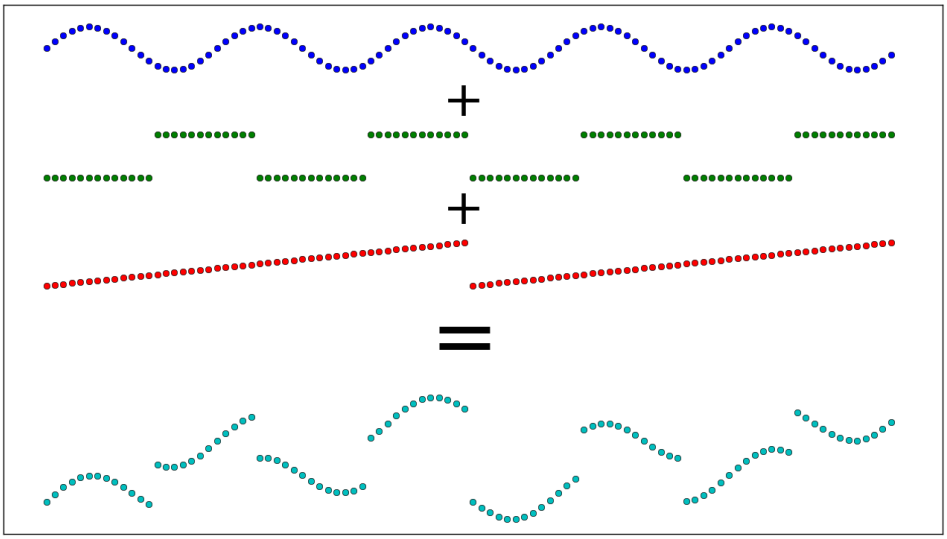

Figura 2.11 *– Mixagem de três sequências sonoras. As amplitudes são sobrepostas diretamente.*

dispõe 100 amostras, de onde pode-se concluir que, se $f_a = 44.1kHz$, as frequências da dente de serra, da onda quadrada e da senoide são, respectivamente, $\frac{f_a}{100/2} = 882Hz$, $\frac{f_a}{100/4} = 1764Hz$ e $\frac{f_a}{100/5} = 2205Hz$. A duração do trecho é bastante curto $\frac{f_a=44.1kHz}{100} \approx 2$ milisegundos. Basta completar com zeros para somar sequências de tamanhos diferentes.

As notas mixadas são em grande parte separadas pelo ouvido por leis físicas de ressonância e pelo sistema nervoso.(3) O resultado da mixagem de notas musicais é a harmonia musical, cujos intervalos entre as frequências e os acordes de notas simultâneas regem aspectos subjetivos e abstratos da música e sua apreciação.(30)

As sequências podem também ser concatenadas no tempo. Caso as sequências $\{t_{k,i}\}_0^{\Lambda_k-1}$ de tamanhos Λ_k representem k notas musicais, sua concatenação em uma única sequência T_i é em uma sequência musical simples ou melodia:

$$\{t_i\}_0^{\sum \Delta_k-1} = \{t_{l,i}\}_0^{\sum \Delta_k-1}, \ l \text{ menor inteiro } : \quad \Lambda_l > i - \sum_{j=0}^{l-1} \Lambda_j \tag{2.31}$$

Este mecanismo é demonstrado de forma ilustrativa na figura 2.12 com as mesmas sequências da figura 2.11. As sequências são curtas para as taxas de amostragem usuais, mas pode-se observar a concatenação de sequências sonoras. Além disso, cada nota tem a duração maior que $100ms$ se $f_a < 1kHz$.

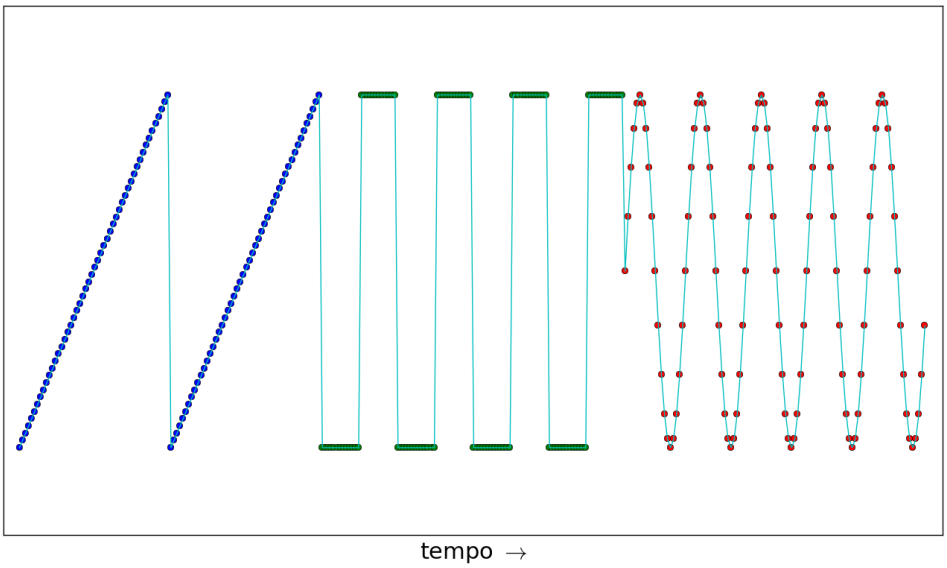

Figura 2.12 – *Concatenação de três sequências sonoras através da justaposição temporal de suas amostras.*

A montagem musical *reduced-fi* explora de forma isolada este uso de justaposição temporal das notas, resultando em uma peça homofônica. O princípio vertical está demonstrado nos *quadros sonoros*, sons estáticos com espectros peculiares. Ambas as peças estão em código Python nos Apêndices B.1.1 e B.1.2 e estão disponíveis como parte da *toolbox* MASSA.(2)

Está descrita a nota musical digital básica e a seção a seguinte desenvolve a evolução temporal de seus conteúdos, como nos glissandos e nas envoltórias de volume. A filtragem de componentes espectrais e a geração dos ruídos completam a constituição da nota musical como unidade isolada e se desdobra na seção 2.3, dedicada à estruturação destas notas em música através de métricas e trajetórias.

2.2 Variações na nota musical básica

A nota musical digital básica foi definida na seção 2.1 com os parâmetros: duração, altura, intensidade (volume) e timbre. Esta é uma modelagem útil e paradigmática, mas não esgota o que se entende por uma nota musical.

Em primeiro lugar, as características da nota se modificam no decorrer da própria nota.(24) Por exemplo, uma nota de piano de 3 segundos tem a intensidade com início abrupto e decaimento progressivo, além de variações do espectro, com harmônicos que decaem antes dos outros e alguns que aparecem com o tempo. Estas variações não são obrigatórias e sim orientações da síntese sonora para usos musicais, pois é como os sons se apresentam na natureza[13]. Explorar todas as formas pelas quais estas variações ocorrem está fora do escopo de qualquer trabalho dada a considerável sensibilidade do ouvido humano e a complexidade da nossa cognição sonora. A seguir, serão apontados recursos primários para estas variações das características na nota básica. Todas as relações descritas nesta seção estão implementadas em Python no Apêndice A.2. As montagens musicais *Tansita para metro*, *Vibra e treme*, *Tremolos, vibratos e a frequência*, *Trenzinho de caipiras impulsivos*, *Ruidosa faixa*, *Bela rugosi*, *Chorus infantil*, *ADa e SaRa* estão nos Apêndices B.2.1, B.2.2, B.2.3, B.2.4, B.2.5, B.2.6, B.2.7 e B.2.8. Estes códigos são parte da caixa de ferramentas MASSA, disponível online.(2)

2.2.1 Tabela de busca

Mais conhecida pelo termo em inglês, a *Lookup Table* (ou simplesmente LUT), é uma estrutura de dados para consultas indexadas usada frequentemente para reduzir a complexidade computacional e por permitir o uso de funções sem possibilidade de cálculo direto, como amostras recolhidas da natureza. Na música seu uso transcende estes primeiros, facilitando as operações

[13] A regra de ouro aqui é: para que um som isolado desperte interesse por si só, faça como que tenha variações internas.(3)

e permitindo que um único período de onda possa ser usado para sintetizar sons em toda a banda de frequências audíveis, qualquer que seja a forma de onda amostrada.

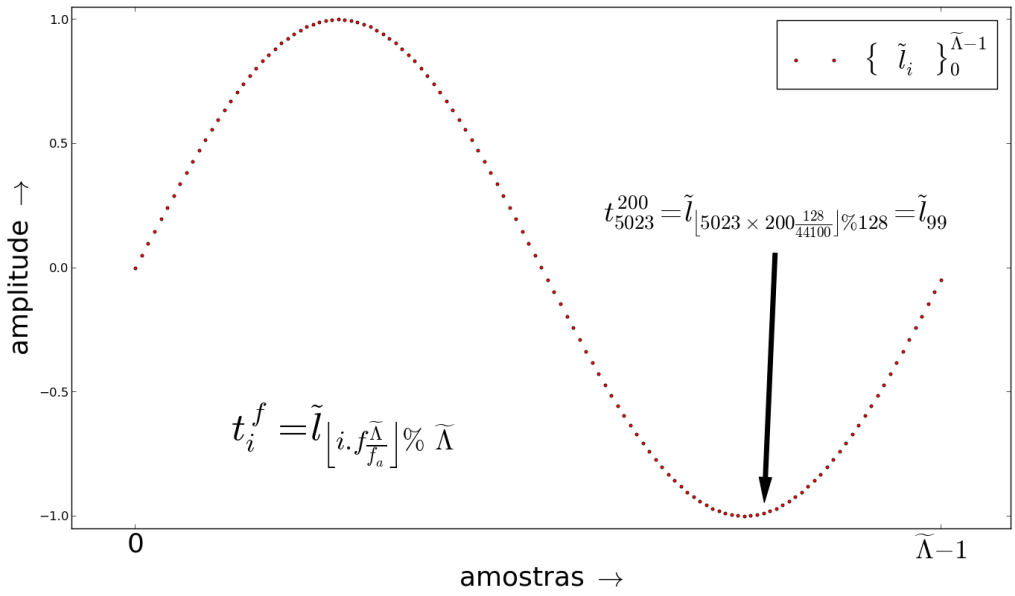

Figura 2.13 *– Procedimento de busca em tabela (conhecido como* Lookup Table*) para síntese de sons em frequências diferentes a partir de uma única forma de onda em alta resolução.*

Seja $\widetilde{\Lambda}$ o tamanho do período e $\widetilde{L_i} = \left\{\widetilde{l_i}\right\}_0^{\widetilde{\Lambda}-1}$ os elementos $\widetilde{l_i}$ de um período de onda qualquer (veja equação 2.23). Uma sequência $T_i^{f,\Delta}$ com amostras de um som de frequência f e duração Δ pode ser obtida a partir de $\widetilde{L_i}$ da seguinte forma:

$$T_i^{f,\Delta} = \left\{t_i^f\right\}_0^{\lfloor f_a.\Delta \rfloor - 1} = \left\{\widetilde{l}_{\gamma_i \% \widetilde{\Lambda}}\right\}_0^{\Lambda-1}, \quad \text{onde } \gamma_i = \left\lfloor i.f\frac{\widetilde{\Lambda}}{f_a} \right\rfloor \tag{2.32}$$

Ou seja, com os índices corretos ($\gamma_i \% \widetilde{\Lambda}$) da LUT, pode-se sintetizar o som em qualquer frequência. A figura 2.13 ilustra o cálculo de uma amostra de $\{t_i\}$ a partir de $\left\{\widetilde{l_i}\right\}$ para uma frequência de $f = 200Hz$, $\widetilde{\Lambda} = 128$ e considerada a taxa de amostragem em $f_a = 44.1kHz$. Esta não é uma configuração praticável, como assinalado abaixo, mas possibilita uma disposição gráfica do procedimento.

O cálculo do inteiro γ_i introduz um ruído, e este diminui com o aumento de $\widetilde{\Lambda}$. Para fins de

síntese, em $f_a = 44.1kHz$ o padrão é usar $\widetilde{\Lambda} = 1024$ amostras, pois já não gera ruído relevante no espectro audível. O método de arredondamento ou interpolação não é decisivo.(31)

A expressão que define a variável γ_i pode ser compreendida da seguinte forma: i é acrescida de f_a a cada 1 segundo. Caso seja dividida pela frequência de amostragem, resulta $\frac{i}{f_a}$, que é acrescida de 1 a cada 1 segundo. Multiplicada pelo comprimento do período, resulta $i\frac{\widetilde{\Lambda}}{f_a}$ que varre o período em 1 segundo. Por fim, com a frequência f, resulta $i.f\frac{\widetilde{\Lambda}}{f_a}$ que completa f varreduras do período $\widetilde{\Lambda}$ em 1 segundo, i.e. a sequência resultante apresenta a frequência fundamental f.

Importantes considerações: f é qualquer, só há limitantes nas frequências graves quando o tamanho da tabela $\widetilde{\Lambda}$ não é suficientemente grande para a taxa de amostragem f_a. O procedimento de busca em tabela é computacionalmente bastante barato, substituindo cálculos por buscas simples (por isso geralmente é entendido como um processo de otimização). Salvo quando assinalado, no texto que usará este procedimento para todos os casos cabíveis pois simplifica as rotinas e é computacionalmente coerente.

O uso de LUTs é bastante difundido nas implementações computacionais voltadas para música e um uso clássico que explora com ênfase as LUTs na síntese sonora musical, é a chamada *Wavetable Synthesis* que consiste em várias LUTs utilizadas em conjunto através da mixagem para gerar uma nota musical quasi-periódica. (10, 32).

2.2.2 Variações incrementais de frequência e intensidade

Segundo a lei de Weber e Fechner, a percepção humana tem uma relação logarítmica com o estímulo que a causa.(33) Em outras palavras, um estímulo em progressão exponencial é percebido como linear. Por razões didáticas e dado o uso nas AM e FM (veja subseção 2.2.5), a variação linear será abordada primeiro.

Em uma nota de duração $\Delta = \frac{\Lambda}{f_a}$, a frequência $f = f_i$ varia de f_0 até $f_{\Lambda-1}$ linearmente. Pode-se escrever:

$$F_i = \{f_i\}_0^{\Lambda-1} = \left\{ f_0 + (f_{\Lambda-1} - f_0)\frac{i}{\Lambda - 1} \right\}_0^{\Lambda-1} \tag{2.33}$$

$$\Delta_{\gamma_i} = f_i \frac{\widetilde{\Lambda}}{f_a} \quad \Rightarrow \quad \gamma_i = \left\lfloor \sum_{j=0}^{i} f_j \frac{\widetilde{\Lambda}}{f_a} \right\rfloor = \left\lfloor \sum_{j=0}^{i} \frac{\widetilde{\Lambda}}{f_a} \left[f_0 + (f_{\Lambda-1} - f_0)\frac{j}{\Lambda - 1} \right] \right\rfloor \tag{2.34}$$

$$\left\{ t_i^{\overline{f_0, f_{\Lambda-1}}} \right\}_0^{\Lambda-1} = \left\{ \widetilde{l}_{\gamma_i \% \widetilde{\Lambda}} \right\}_0^{\Lambda-1} \tag{2.35}$$

Onde $\Delta_{\gamma_i} = f_i \frac{\widetilde{\Lambda}}{f_a}$ é o incremento da LUT entre duas amostras dada a frequência do som na primeira amostra.

Desta forma, pode-se calcular os elementos $t_i^{\overline{f_0, f_{\Lambda-1}}}$ com base no período $\left\{ \widetilde{l}_i \right\}_0^{\Lambda-1}$.

As equações 2.33, 2.34 e 2.35 são relativas à progressão linear da frequência. Como assinalado para o caso geral, também aqui uma progressão de frequência *percebida* como linear segue uma progressão exponencial[14]. Pode-se escrever que: $f_i = f_0 . 2^{\frac{i}{\Lambda-1} n_8}$ onde $n_8 = \log_2 \frac{f_{\Lambda-1}}{f_0}$ é o número de oitavas entre f_0 e $f_{\Lambda-1}$. De forma que $f_i = f_0 . 2^{\frac{i}{\Lambda-1} \log_2 \frac{f_{\Lambda-1}}{f_0}} = f_0 . 2^{\log_2 \left(\frac{f_{\Lambda-1}}{f_0} \right)^{\frac{i}{\Lambda-1}}} = f_0 \left(\frac{f_{\Lambda-1}}{f_0} \right)^{\frac{i}{\Lambda-1}}$. Portanto, as equações de transições de frequência lineares para o ouvido são:

$$F_i = \{f_i\}_0^{\Lambda-1} = \left\{ f_0 \left(\frac{f_{\Lambda-1}}{f_0} \right)^{\frac{i}{\Lambda-1}} \right\}_0^{\Lambda-1} \tag{2.36}$$

$$\Delta_{\gamma_i} = f_i \frac{\widetilde{\Lambda}}{f_a} \quad \Rightarrow \quad \gamma_i = \left\lfloor \sum_{j=0}^{i} f_j \frac{\widetilde{\Lambda}}{f_a} \right\rfloor = \left\lfloor \sum_{j=0}^{i} f_0 \frac{\widetilde{\Lambda}}{f_a} \left(\frac{f_{\Lambda-1}}{f_0} \right)^{\frac{j}{\Lambda-1}} \right\rfloor \tag{2.37}$$

[14] Ou, dito ainda de outra forma, uma progressão geométrica da frequência é percebida como uma progressão aritmética de alturas.

$$\left\{t_i^{\overline{f_0, f_{\Lambda-1}}}\right\}_0^{\Lambda-1} = \left\{\widetilde{l}_{\gamma_i \% \widetilde{\Lambda}}\right\}_0^{\Lambda-1} \tag{2.38}$$

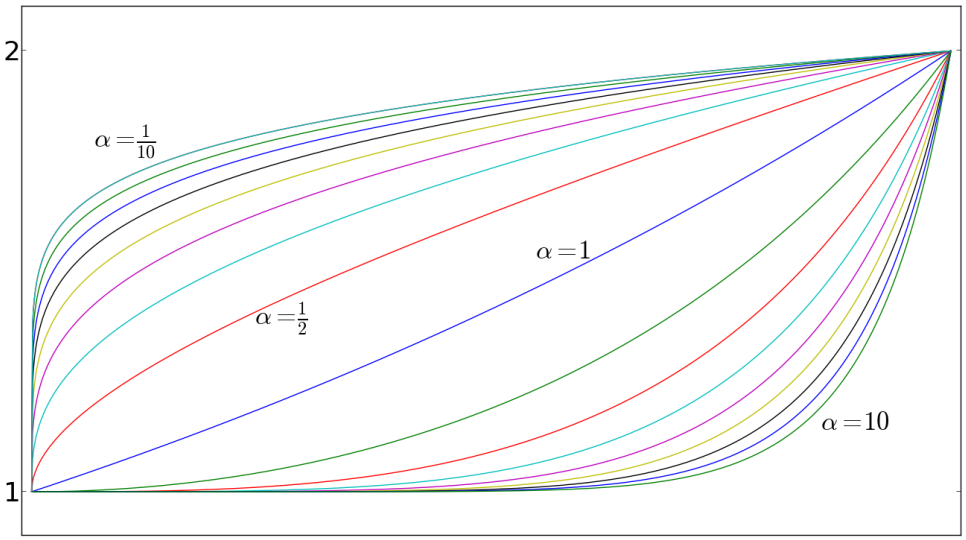

Figura 2.14 *– Transições de intensidade para diferentes valores de α (veja equações 2.39 e 2.40).*

O termo $\frac{i}{\Lambda-1}$ varre o intervalo $[0,1]$ e pode-se elevá-lo a uma potência para que o início da transição seja mais suave ou abrupto. Este procedimento é útil para variações de energia da onda vibratória para alteração do volume[15]. Basta multiplicar a sequência original (seja ela gerada ou pré-estabelecida) pela sequência $a_{\Lambda-1}^{\left(\frac{i}{\Lambda-1}\right)^{\alpha}}$ onde α é o coeficiente citado e $a_{\Lambda-1}$ é fração da amplitude original que se visa atingir ao final da transição.

Assim, para variações de amplitude:

$$\{a_i\}_0^{\Lambda-1} = \left\{a_0\left(\frac{a_{\Lambda-1}}{a_0}\right)^{\left(\frac{i}{\Lambda-1}\right)^{\alpha}}\right\}_0^{\Lambda-1} = \left\{(a_{\Lambda-1})^{\left(\frac{i}{\Lambda-1}\right)^{\alpha}}\right\}_0^{\Lambda-1} \text{ com } a_0 = 1 \tag{2.39}$$

[15] A mudança do volume (qualidade psicofísica) ocorre através de diferentes características do som, como a reverberação e a concentração de harmônicos agudos, dentre as quais está a energia da onda. A manipulada com mais facilidade é a energia da onda (veja equação 2.2) e esta também pode variar de diferentes formas. Uma forma mais simples é variar a amplitude através da multiplicação da sequência toda por um número real. O aumento de energia sem variação de amplitude é a *compressão sonora*, útil na produção musical atual.(34)

$$T_i^{'} = T_i \odot A_i = \{t_i.a_i\}_0^{\Lambda-1} = \left\{t_i.(a_{\Lambda-1})^{\left(\frac{i}{\Lambda-1}\right)^{\alpha}}\right\}_0^{\Lambda-1} \tag{2.40}$$

Pode-se tomar $a_0 = 1$ para iniciar a nova sequência com a amplitude original e então ir modificando com o decorrer das amostras. Esta restrição faz com que o termo $a_{\Lambda-1}$ seja a variação da amplitude. Caso $\alpha = 1$, a variação de amplitude segue exatamente a progressão geométrica que caracteriza a percepção linear. A figura 2.14 exibe as transições para diferentes valores de α e para a transição entre os valores 1 e 2, um ganho de $\approx 6dB$ segundo a equação 2.4.

Algum cuidado é necessário para lidar com $a = 0$. Na equação 2.39, se $a_0 = 0$ há divisão por zero e se $a_{\Lambda-1} = 0$, há uma multiplicação por zero. Ambos os casos tornam o procedimento inútil pois nenhum número diferente de zero pode ser representado como uma proporção com relação ao zero. Pode-se resolver isso escolhendo um número suficientemente pequeno como $-80dB \Rightarrow a = 10^{\frac{-80}{20}} = 10^{-4}$ como o volume mínimo no caso de um *fade in* ($a_0 = 10^{-4}$) ou de um *fade out* ($a_{\Lambda-1} = 10^{-4}$).

Para uma amplificação linear, mas não linear para a percepção, basta usar uma sequência $\{a_i\}$ adequada:

$$a_i = a_0 + (a_{\Lambda-1} - a_0)\frac{i}{\Lambda - 1} \tag{2.41}$$

Aqui convém a conversão de decibels para amplitude. Assim, as equações 2.8 e 2.40 especificam a transição de V_{dB} decibels:

$$T_i^{'} = \left\{t_i 10^{\frac{V_{dB}}{20}\left(\frac{i}{\Lambda-1}\right)^{\alpha}}\right\}_0^{\Lambda-1} \tag{2.42}$$

para o caso geral de variações de amplitude segundo a progressão geométrica. Quanto maior o valor de α, mais suave é a introdução do som e mais intenso o final da transição. $\alpha > 1$ resulta

em transições de volume muitas vezes chamadas de *slow fade* enquanto $\alpha < 1$ resulta em *fast fade*.(34)

As transições lineares serão usadas para as sínteses AM e FM e a aplicação das transições logarítmicas para os tremolos e vibratos. Uma exploração não oscilatória destas variações está na montagem musical *Transita para metro*, cujo código está no Apêndice B.2.1 e online na MASSA.(2)

2.2.3 Aplicação de filtros digitais

Esta subseção limita-se a uma descrição do processamento das sequências, por convolução e equação a diferenças, e em aplicações imediatas, pois a complexidade facilmente foge ao escopo[16]. A aplicação de filtros pode ser parte constituinte da síntese ou feita posteriormente como parte dos processos tipicamente chamados de tratamento sonoro.

- Convolução e filtros de resposta ao impulso finita (FIR)

  Os filtros aplicados por convolução são conhecidos pela sua sigla FIR (do inglês Finite Impulse Response) e são caracterizados por possuirem uma representação amostral finita no tempo. Esta representação amostral é chamada de 'resposta ao impulso' $\{h_i\}$. Os filtros FIR são aplicados no domínio temporal ao som digitalizado pela convolução do som com a resposta ao impulso do filtro[17]. Para os fins deste trabalho, a convolução fica definida como:

[16] A elaboração de filtros constitui uma área reconhecidamente complexa, com literatura e pacotes de software dedicados. Recomendamos ao leitor interessado uma visita à nossa bibliografia.(25, 35)

[17] Pode-se aplicar o filtro do domínio espectral através da multiplicação das transformadas de Fourier de ambos o som e a resposta ao impulso, e então realizada a transformada inversa de Fourier do espectro resultante.(25)

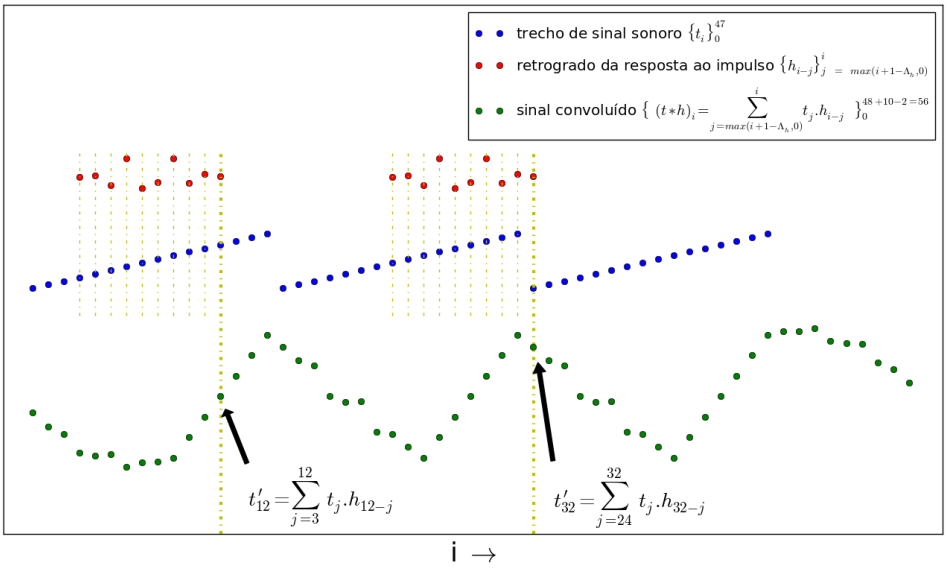

Figura 2.15 – *Interpretação gráfica da convolução. Cada amostra resultante é a soma das amostras anteriores de um sinal uma a uma multiplicadas pelas amostras retrógradas do outro sinal.*

$$\left\{t'_i\right\}_0^{\Lambda_t+\Lambda_h-2=\Lambda_{t'}-1} = \{(T_j * H_j)_i\}_0^{\Lambda_{t'}-1} = \left\{ \sum_{j=0}^{min(\Lambda_h-1,i)} h_j.t_{i-j} \right\}_0^{\Lambda_{t'}-1} = \left\{ \sum_{j=max(i+1-\Lambda_h,0)}^{i} t_j.h_{i-j} \right\}_0^{\Lambda_{t'}-1} \tag{2.43}$$

Onde $t_i = 0$ para as amostras não definidas de antemão. Ou seja, o som $\{t'_i\}$ resultante da convolução de $\{t_i\}$ com a resposta ao impulso $\{h_i\}$ tem cada i-ésima amostra t_i substituída pela soma de suas últimas Λ_h amostras $\{t_{i-j}\}_{j=0}^{\Lambda_h-1}$ multiplicadas uma a uma pelas amostras da resposta ao impulso $\{h_i\}_0^{\Lambda_h-1}$. Este procedimento está ilustrado na figura 2.15, onde a resposta ao impulso $\{h_i\}$ é percorrida na forma retrógrada e t'_{12} e t'_{32} são duas amostras calculadas pela convolução $(T_j * H_j)_i = t'_i$. O sinal resultante possui sempre o tamanho $\Lambda_t + \Lambda_h - 1 = \Lambda_{t'}$.

Com este procedimento pode-se aplicar reverberadores, equalizadores, delays e vários

outros tipos de filtros para fins de tratamento sonoro ou efeitos musicais/artísticos.

A resposta ao impulso pode provir de medições físicas ou da síntese. Uma resposta ao impulso para a aplicação de reverberação pode resultar da gravação sonora em um ambiente ao disparar um estalo que se assemelhe a um impulso ou de uma varredura em senoide, que transformada se aproxima da resposta em frequência. Ambas são respostas ao impulso que, convoluídas com a sequência sonora, resultam na própria sequência com uma reverberação que se assemelha àquela do ambiente em que ocorreu a medição.(10)

A transformada inversa de Fourier de uma envoltória par e real é uma resposta ao impulso de um FIR. Este realiza uma filtragem em frequência com a envoltória. Quanto maior o número de amostras maior a resolução da envoltória e também o processamento computacional, pois a convolução é cara.

Uma propriedade importante é o deslocamento temporal causado pela convolução com o impulso deslocado. Embora caro computacionalmente, pode-se criar linhas de *delays* através da convolução do som com uma resposta ao impulso que possui um impulso para cada reincidência do som. Na figura 2.16 pode-se observar o deslocamento causado pela convolução com o impulso. Dependendo da densidade dos impulsos, o resultado é de caráter rítmico (20 impulsos por segundo ou menos) ou de amálgama sonoro (20-40 impulsos por segundo ou mais). Neste último caso, ocorrem processos tipicamente vinculados à síntese granular, delays, reverbs e equalizações.

- Filtros de resposta ao impulso infinita (IIR)

Esta classe de filtros é conhecida pela sigla IIR (do inglês Infinite Impulse Response) e é caracterizada por possuir uma representação temporal infinita, i.e. a resposta ao impulso não converge para zero. Sua aplicação é usualmente feita pela equação:

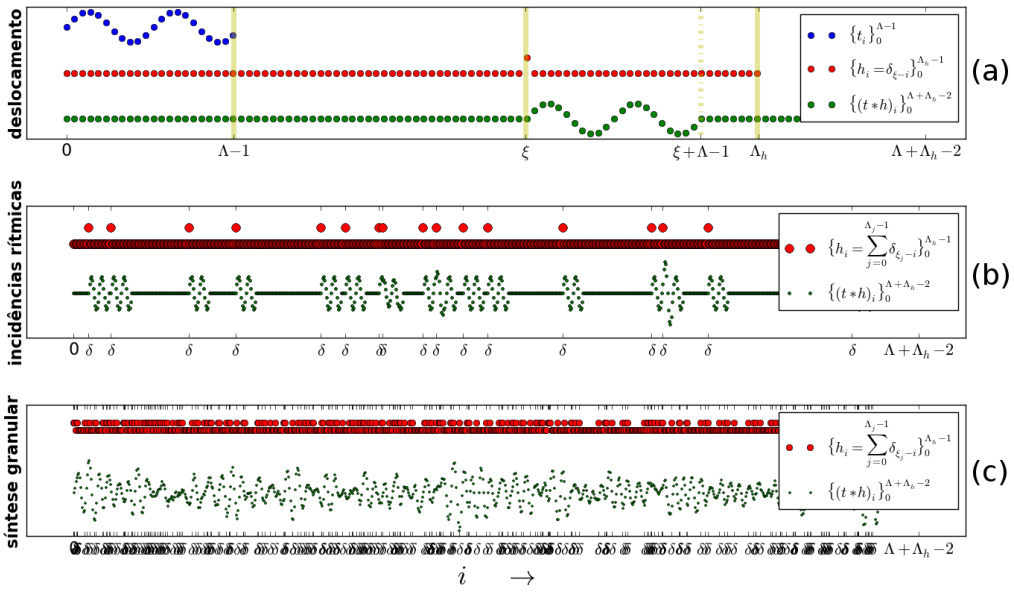

Figura 2.16 *– Convolução com o impulso: deslocamento (a), linhas de delays (b) e síntese granular (c). Dispostos em ordem crescente de densidade de pulsos.*

$$t'_i = \frac{1}{b_0}\left(\sum_{j=0}^{J} a_j.t_{i-j} + \sum_{k=1}^{K} b_k.t'_{i-k}\right) \tag{2.44}$$

com $b_0 = 1$ na grande maioria dos casos pois pode-se normalizar as variáveis: $a'_j = \frac{a_j}{b_0}$ e $b'_k = \frac{b_k}{b_0} \Rightarrow b'_0 = 1$. A equação 2.44 é chamada 'equação a diferenças' por exibir as amostras resultantes $\left\{t'_i\right\}$ através das diferenças entre as amostras originais $\{t_i\}$ e as amostras resultantes anteriores $\left\{t'_{i-k}\right\}$.

Existem diversos métodos e ferrametas para a elaboração de filtros IIR e segue abaixo uma seleção com fins didáticos e para consulta futura por utilidade. São filtros bem comportados e cujas filtragens estão na figura 2.17.

No caso dos filtros de ordem simples, a frequência de corte f_c é onde o filtro realiza uma atenuação de $-3dB \approx 0.707$ da amplitude original. No caso dos filtros passa e rejeita banda, esta mesma atenuação é resultado de duas especificações: f_c (neste caso mais bem compreendida como 'frequência central') e a largura de banda bw, em ambas as frequên-

cias $f_c \pm bw$ há uma atenuação de ≈ 0.707 da amplitude original. Existe amplificação do som no caso dos filtros passa e rejeita banda quando a frequência de corte é baixa e a largura de banda é grande o suficiente. Nos agudos, estes filtros apresentam somente um desvio do perfil esperado, expandindo a envoltória para o lado grave da banda em evidência.

Para filtros cujas respostas em frequência possuem outras envoltórias (para o módulo), pode-se realizar cascatas destes filtros aplicando-os sucessivamente. Outra possibilidade é utilizar alguma receita de filtro biquad[18] ou rotinas para cálculo de coeficientes de filtros Chebichev[19]. Ambas as possibilidades são exploradas por títulos em nossas referências, em especial (35, 36) e a coleção de filtros da comunidade *Music-DSP*, da Universiade de Columbia.(25, 37)

1. Passa-baixas de polo simples com módulo da resposta em frequência no canto superior esquerdo da figura 2.17. A fórmula geral tem por referência da frequência de corte $f_c \in (0, \frac{1}{2})$, fração da frequência de amostragem f_a em que há aproximadamente uma atenuação de $3dB$. Os coeficientes do filtro IIR a_0 e b_1 são dados através da variável intermediária $x \in [e^{-\pi}, 1]$:

$$\begin{aligned} x &= e^{-2\pi f_c} \\ a_0 &= 1 - x \\ b_1 &= x \end{aligned} \tag{2.45}$$

2. Passa-altas de polo simples com o módulo da resposta em frequência no canto superior direito da figura 2.17. A fórmula geral, com frequência de corte $f_c \in (0, \frac{1}{2})$, é calculada

[18] Abreviação de 'biquadrado' pois sua função de transferência possui dois polos e dois zeros, i.e. sua forma normal consiste em dois polinômios quadráticos formando uma fração: $\mathbb{H}(z) = \frac{a_0 + a_1.z^{-1} + a_2.x^{-2}}{1 - b_1.z^{-1} - b_2.z^{-2}}$.

[19] Filtros Butterworth e Elípticos podem ser considerados como casos específicos dos Filtros do tipo Chebichev.(25, 35)

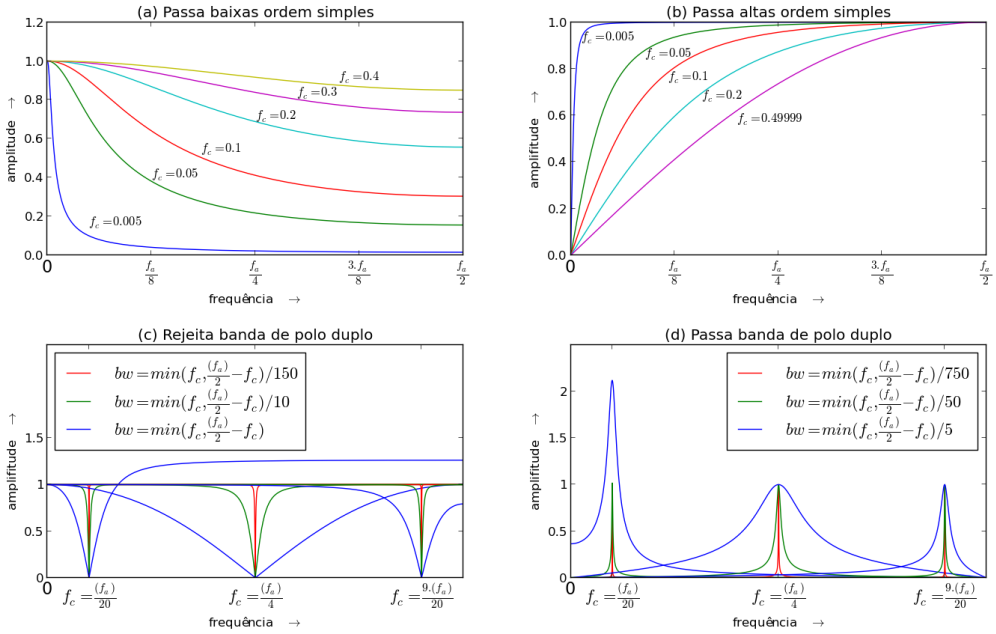

Figura 2.17 – *Módulos da resposta em frequência (a), (b), (c) e (d) respectivamente dos filtros IIR das equações 2.45, 2.46, 2.48 e 2.49 para diferentes frequências de corte, frequências centrais e larguras de banda.*

através da variável intermediária $x \in [e^{-\pi}, 1]$:

$$\begin{aligned} x &= e^{-2\pi f_c} \\ a_0 &= \frac{x+1}{2} \\ a_1 &= -\frac{x+1}{2} \\ b_1 &= x \end{aligned} \tag{2.46}$$

3. Nó (*notch filter*). Este filtro é parametrizado pela frequência central[20] f_c e a largura de banda bw - $f_c \pm bw$, que resultam em 0.707 da amplitude, i.e. atenuação de $3dB$ - ambos dados como frações de f_a, portanto f, $bw \in (0, 0.5)$.

   Por facilidade, sejam as variáveis auxiliares K e R:

[20] Atenção com a frequência de corte também f_c nos filtros passa baixas e passa altas.

$$
\begin{aligned}
R &= 1 - 3bw \\
K &= \frac{1 - 2R\cos(2\pi f_c) + R^2}{2 - 2\cos(2\pi f_c)}
\end{aligned}
\tag{2.47}
$$

O filtro passa banda do canto inferior esquerdo da figura 2.17 possui os seguintes coeficientes para a equação 2.44:

$$
\begin{aligned}
a_0 &= 1 - K \\
a_1 &= 2(K - R)\cos(2\pi f_c) \\
a_2 &= R^2 - K \\
b_1 &= 2R\cos(2\pi f_c) \\
b_2 &= -R^2
\end{aligned}
\tag{2.48}
$$

Os coeficientes do filtro rejeita banda são:

$$
\begin{aligned}
a_0 &= K \\
a_1 &= -2K\cos(2\pi f_c) \\
a_2 &= K \\
b_1 &= 2R\cos(2\pi f_c) \\
b_2 &= -R^2
\end{aligned}
\tag{2.49}
$$

com o módulo de sua resposta em frequência disposto na parte inferior esquerda da figura 2.17.

2.2.4 Ruídos

De forma geral, os sons sem altura definida são chamados ruídos.(23) Estes são constituintes importantes dos sons musicais de altura definida, como os ruídos presentes nas notas do piano, do violino, etc. Além disso, os instrumentos de percussão, em grande parte, não possuem altura definida e seus sons são em geral compreendidos como ruídos.(3) Na música eletrônica, incluindo a eletroacústica e gêneros de pista de dança, os ruídos possuem usos diversificados e comumente característicos do estilo musical.(10)

A ausência de uma altura definida é fruto da ausência de uma organização harmônica perceptível nas componentes senoidais que formam o som. Assim, são incontáveis as possibilidades de gerar ruídos. A utilização de valores aleatórios para a geração da sequência sonora T_i é um método atraente, mas os resultados não são tão úteis, tendendo geralmente ao ruído branco.(10)

Outra possibilidade é a geração de ruído através do espectro desejado, a partir do qual executamos a transformada inversa de Fourier. A distribuição espectral deve ser feita com cuidado pois caso se utilize a mesma fase, ou fases com forte correlação, o som sintetizado possuirá energia bastante concentrada em alguns trechos de sua duração.

Abaixo elencamos alguns ruídos de espectro estático. São chamados *coloridos* por terem sido associados a cores. A figura 2.18 mostra lado a lado o perfil espectral e a sequência sonora. Os ruídos foram gerados com a mesma fase, então pode-se observar o resultado das contribuições em diferentes regiões do espectro.

- O ruído branco deve seu nome por possuir energia distribuída igualmente por todas as frequências. Pode-se realizar o ruído branco com a transformada inversa dos seguintes coeficientes:

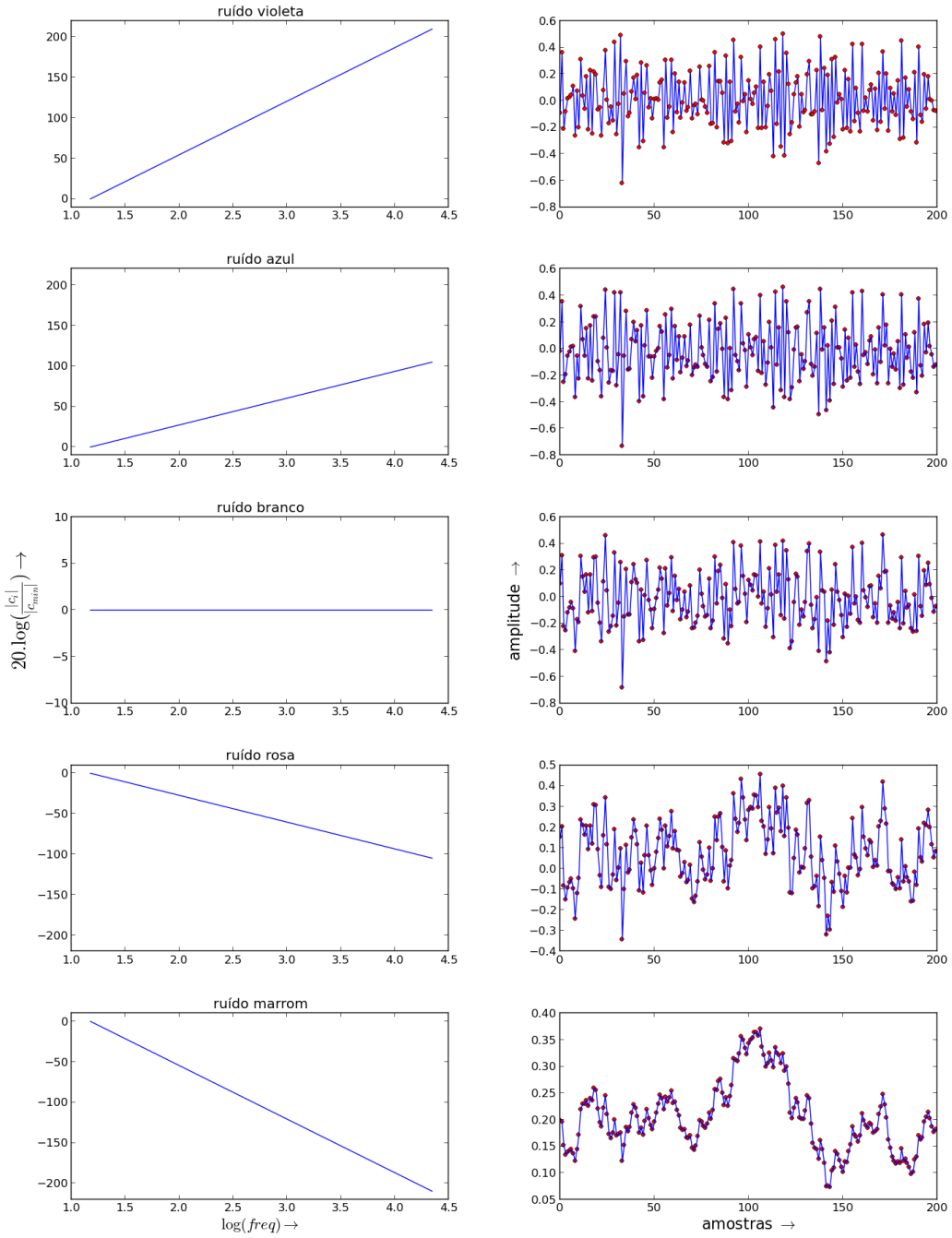

Figura 2.18 *– Ruídos coloridos realizados através das equações 2.50, 2.51, 2.52, 2.53, 2.54: espectros e ondas sonoras resultantes.*

$$
\begin{aligned}
& c_0 = 0 \quad \text{pois evita-se bias} \\
& c_i = e^{j.x}\ ,\ \ j^2 = -1\ ,\ \ x \text{ randômico } \in [0, 2\pi]\ ,\ \ i \in \left[1, \frac{\Lambda}{2} - 1\right] \\
& c_{\Lambda/2} = 1 \qquad (\text{se } \Lambda \text{ par}) \\
& c_i = c^*_{\Lambda - i}\ ,\ \ \text{para } i > \frac{\Lambda}{2}
\end{aligned}
\tag{2.50}
$$

O valor de c_i calculado pela exponencial é apenas um artifício para resultar em módulo unitário e fase aleatória. Já $c_{\Lambda/2}$ é sempre puramente real (como vimos na seção anterior).

- O ruído rosa possui uma queda de $3dB$ por oitava. Este ruído é muito usual no teste de equipamentos e montagens de aparelhos além de presença destacada na natureza (3).

$$
\begin{aligned}
& f_{\min} \approx 15Hz \\
& f_i = i\frac{f_a}{\Lambda}\ , \qquad i \leq \frac{\Lambda}{2},\ i \in \mathbb{N} \\
& \alpha_i = \left(10^{-\frac{3}{20}}\right)^{\log_2\left(\frac{f_i}{f_{\min}}\right)} \\
& c_i = 0\ ,\ \ \forall\, i\ : f_i < f_{\min} \\
& c_i = e^{j.x}.\alpha_i\ ,\ \ j^2 = -1\ ,\ \ \ x \text{ randômico } \in [0, 2\pi]\ ,\ \ \forall\, i\ : f_{\min} \leq f_i < f_{\lceil \Lambda/2 - 1 \rceil} \\
& c_{\Lambda/2} = \alpha_{\Lambda/2} \qquad (\text{se } \Lambda \text{ par}) \\
& c_i = c^*_{\Lambda - i}\ ,\ \ \text{para } i > \Lambda/2
\end{aligned}
\tag{2.51}
$$

A frequência mínima $f_{\min}$ pode ser escolhida com base no limite da audição, pois não se escuta como altura uma componente sonora cuja frequência esteja abaixo de $\approx\ 20Hz$.

Os ruídos restantes podem ser feitos com base no procedimento descrito para o ruído rosa, bastando que modificar detalhes, em especial a equação que define α_i.

- O ruído marrom deve seu nome a Robert Brown, que descreveu o movimento browniano.

Embora esta origem seja um tanto díspar do que pode-se considerar motivo para uma associação com a cor marrom, o ruído sonoro ficou consagrado com este nome. De qualquer forma, é bastante comum declarar satisfatória a associação do ruído com a cor marrom, uma vez que os ruídos branco e rosa são mais estridentes e relacionados a cores mais intensas (10, 34).

O que caracteriza este ruído é a queda de $6dB$ por oitava. Desta forma, α_i no conjunto 2.51 fica:

$$\alpha_i = (10^{-\frac{6}{20}})^{\log_2\left(\frac{f_i}{f_{\text{min}}}\right)} \tag{2.52}$$

- No ruído azul há ganho de $3dB$ por oitava em uma banda limitada pela frequência mínima f_{min} e a frequência máxima $f_{\text{máx}}$. Assim, também com base no conjunto de equações 2.51:

$$\begin{aligned} \alpha_i &= (10^{\frac{3}{20}})^{\log_2\left(\frac{f_i}{f_{\text{min}}}\right)} \\ c_i &= 0\,, \ \forall\, i \ : f_i < f_{\text{min}} \ \text{ ou } \ f_i > f_{\text{máx}} \end{aligned} \tag{2.53}$$

- O ruído violeta é similar ao ruído azul, mas o ganho é de $6dB$ por oitava:

$$\alpha_i = (10^{\frac{6}{20}})^{\log_2\left(\frac{f_i}{f_{\text{min}}}\right)}\,, \quad f_{\text{min}} \approx 15Hz \tag{2.54}$$

- O ruído preto possui perdas maiores que $6dB$ por oitava, assim:

$$\alpha_i = (10^{-\frac{\beta}{20}})^{\log_2\left(\frac{f_i}{f_{\text{min}}}\right)}\,, \quad \beta > 6 \tag{2.55}$$

- O ruído cinza é definido como um ruído branco sujeito a uma das curvas iso-audíveis. Estas curvas são resultados experimentais e necessárias para a obtenção de α_i. Uma implementação da ISO 226, a última revisão destas curvas, está na toolbox MASSA.(2)

Foram expostos somente ruídos com espectro estático. Existem classificações de ruídos com variações do espectro no decorrer do tempo. Existem também ruídos que são fundamentalmente transientes, como os clicks e os chirps. O primeiro é modelado facilmente por um impulso relativamente isolado, enquanto o segundo não é um ruído, mas uma varredura rápida de alguma banda de frequência.(10)

Os ruídos das equações 2.50, 2.51, 2.52, 2.53, 2.54 estão na figura 2.18. Os espectros foram feitos com a mesma fase em cada coeficiente de mesma frequência, de forma que se pode observar a contribuição dos harmônicos agudos e das frequências graves.

2.2.5 Tremolo e vibrato, AM e FM

O vibrato é uma variação periódica de altura (frequência) e o tremolo é uma variação periódica de volume (intensidade).[21] Para o caso geral, o vibrato é descrito da seguinte forma:

$$\gamma'_i = \left\lfloor if' \frac{\widetilde{\Lambda}_M}{f_a} \right\rfloor \tag{2.56}$$

$$t'_i = \widetilde{m}_{\gamma'_i \,\%\widetilde{\Lambda}_M} \tag{2.57}$$

$$f_i = f\left(\frac{f+\mu}{f}\right)^{t'_i} = f.2^{t'_i \frac{\nu}{12}} \tag{2.58}$$

$$\Delta_{\gamma_i} = f_i \frac{\widetilde{\Lambda}}{f_a} \quad \Rightarrow \quad \gamma_i = \left\lfloor \sum_{j=0}^{i} f_j \frac{\widetilde{\Lambda}}{f_a} \right\rfloor = \left\lfloor \sum_{j=0}^{i} \frac{\widetilde{\Lambda}}{f_a} f\left(\frac{f+\mu}{f}\right)^{t'_j} \right\rfloor = \left\lfloor \sum_{j=0}^{i} \frac{\widetilde{\Lambda}}{f_a} f.2^{t'_j \frac{\nu}{12}} \right\rfloor \tag{2.59}$$

[21] Alguns instrumentos e contextos musicais usam nomenclaturas diferentes. Por exemplo, no piano, o chamado tremolo é um vibrato e um tremolo segundo a classificação aqui utilizada. As definições presentes neste trabalho tem por base uma literatura mais abrangente do que a utilizada para um único instrumento, prática ou tradição musical e comum em contextos de teoria musical e musica eletrônica.(23, 30)

$$T_i^{f,vbr(f',\nu)} = \left\{t_i^{f,vbr(f',\nu)}\right\}_0^{\Lambda-1} = \left\{\widetilde{l}_{\gamma_i \% \widetilde{\Lambda}}\right\}_0^{\Lambda-1} \tag{2.60}$$

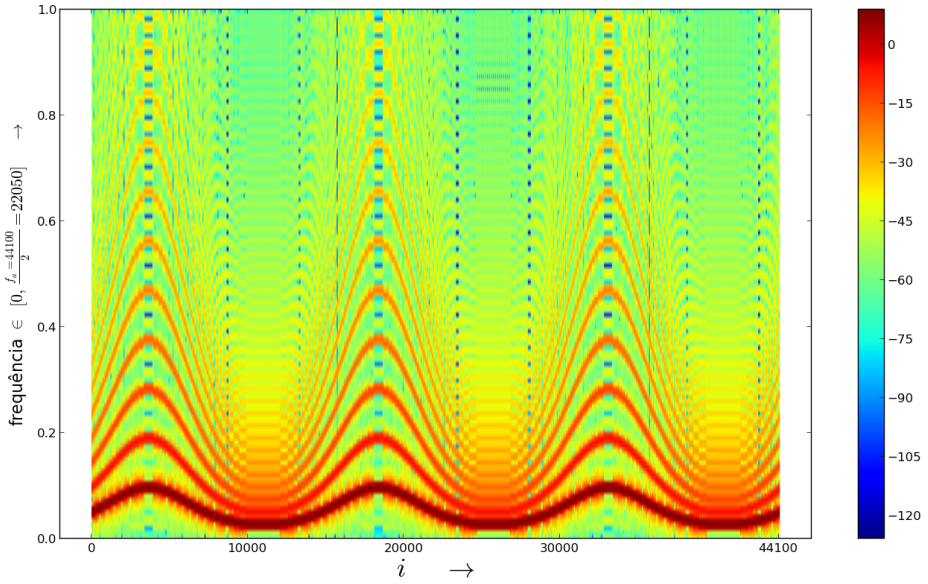

Figura 2.19 *– Espectrograma de um som com vibrato senoidal de* $3Hz$ *e profundidade de uma oitava em uma dente de serra de* $1000Hz$ *(considerada* $f_a = 44.1kHz$*).*

Para a correta realização do vibrato, é importante atenção para as duas tabelas e sequências. A tabela $\widetilde{M}_i$ de tamanho $\widetilde{\Lambda}_M$ e a sequência de índices γ'_i formam a sequência t'_i que é o padrão da oscilação da frequência enquanto a tabela $\widetilde{L}_i$ de tamanho $\widetilde{\Lambda}$ e a sequência de índices γ_i formam t_i que é o som em si. As variáveis μ e ν quantificam a intensidade do vibrato: μ é uma medida direta da quantidade de Herz envolvidos no limite superior da oscilação e ν é a medida direta de semitons envolvidos na oscilação (2ν é o número de semitons entre os picos superiores e inferiores de oscilação da frequência do som $\{t_i\}$ causada pelo vibrato). É conveniente $\nu = \log_2 \frac{f+\mu}{f}$ neste caso pois o aumento máximo de frequência não equivale à diminuição máxima, mas a variação de semitons se mantém.

A Figura 2.19 é o espectrograma de um vibrato artificial de uma nota em $1000Hz$ (entre um si e um dó) e cujo desvio da frequência atinge uma oitava para cima e para baixo. Qualquer forma

de onda pode ser utilizada para gerar o som e o padrão de oscilação do vibrato em quaisquer frequência de oscilação e desvio de altura envolvidos[22]. Estas qualidades não são praticáveis em instrumentos musicais tradicionais, introduzindo novidade nas possibilidades artísticas.

O caso do tremolo é semelhante: f', γ'_i e t'_i permanecem os mesmos. A sequência de amplitudes a serem multiplicadas pela sequência original t_i fica:

$$a_i = 10^{\frac{V_{dB}}{20} t'_i} = a_{\text{máx}}^{t'_i} \tag{2.61}$$

$$T_i^{tr(f')} = \left\{ t_i^{tr(f')} \right\}_0^{\Lambda-1} = \{t_i.a_i\}_0^{\Lambda-1} = \left\{ t_i.10^{t'_i \frac{V_{dB}}{20}} \right\}_0^{\Lambda-1} = \left\{ t_i.a_{\text{máx}}^{t'_i} \right\}_0^{\Lambda-1} \tag{2.62}$$

Onde V_{dB} é a profundidade da oscilação em decibels do tremolo e $a_{\text{máx}} = 10^{\frac{V_{dB}}{20}}$ é o ganho máximo de amplitude envolvido. A medição em decibels é pertinente pois o aumento máximo de amplitude não equivale à diminuição máxima relacionada, enquanto a diferença em decibels se mantém.

A figura 2.20 mostra a amplitude das sequências $\{a_i\}_0^{\Lambda-1}$ e $\{t'_i\}_0^{\Lambda-1}$ para três oscilações de um tremolo com forma da dente de serra. A curvatura é devida à progressão logarítmica de intensidade. A frequência do tremolo é de $1,5Hz$ pois $f_a = 44,1kHz \Rightarrow$ duração $= \frac{i_{\text{máx}}=82000}{f_a} = 2s \Rightarrow \frac{3\text{oscilações}}{2s} = 1,5$ oscilações por segundo (Hz).

A montagem musical *Vibra e treme* explora estes recursos dos tremolos e vibratos em associação e isoladamente com frequências f' e profundidades (ν e V_{dB}) diferentes, variações progressivas dos parâmetros[23]. A peça desenvolve também uma comparação entre os vibratos e tremolos em escala logarítmica e em escala linear para uma apreciação qualitativa. Seu código está no Apêndice B.2.2 e disponível online como parte da *toolbox* MASSA.

[22] O desvio de altura é chamado profundidade do vibrato e é geralmetne dado por conveniência em semitons ou cents.

[23] Os tremolos e vibratos ocorrem muitas vezes juntos em instrumentos tradicionais e na voz.

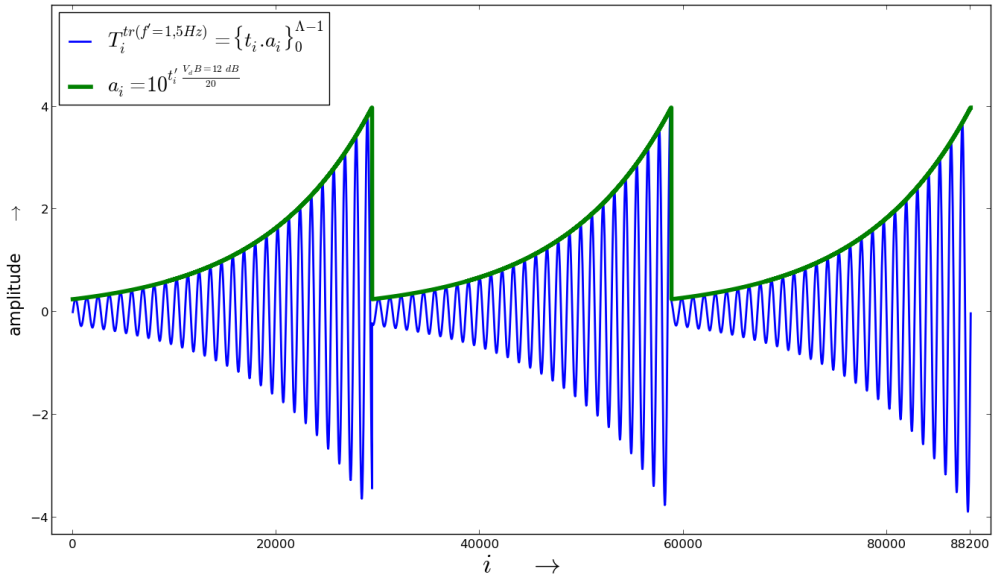

Figura 2.20 – *Tremolo de profundidade $V_{dB} = 12dB$ com padrão oscilatório de uma dente de serra em $f' = 1.5Hz$ em uma senoide de $f = 40Hz$ (considerada taxa de amostragem $f_a = 44,1kHz$).*

No aumento progressivo de f', a aproximação do limiar de frequência para audição do fenômeno sonoro como altura ($\approx 20Hz$) gera rugosidades para ambos tremolos e vibratos. Estas rugosidades são muito apreciadas tanto na tradição erudita quanto na música eletrônica atual, especialmente no *Dubstep*. A rugosidade também é gerada através de conteúdos espectrais que geram batimentos.(38, 39) A sequência *Bela Rugosi* explora este limiar com concomitâncias de tremolos e vibratos na mesma voz, com intensidades e formas de onda diferentes. Seu código está o Apêndice B.2.6 e disponível online como parte da MASSA.

Aumentando ainda mais a frequência, estas oscilações deixam de ser eventos identificáveis. Neste caso, as oscilações são audíveis como altura. Assim, f', μ e a forma de onda realizam alterações espectrais no som original T_i de formas diferentes para os tremolos e para os vibratos. São as chamadas sínteses AM (*Amplitude Modulation*) e FM (*Frequency Modulation*), respectivamente. Estas são técnicas conhecidas, com aplicações em sintetizadores como o *Yamaha DX7*, e com aplicações fora da música, como em telecomunicações para transmissão de

informação via ondas eletromagnéticas (ex. rádios AM e FM).

Para fins musicais e em resumo, pode-se entender a síntese FM através do caso entre senoides e decompor os sinais em seus espectros de Fourier (i.e. senoidais) para casos mais complexos. Assim, a síntese FM realizada com um 'vibrato' senoidal de frequência f' e profundidade μ em um som também senoidal T_i de frequência f gera bandas centradas em f e distantes f' entre si:

$$\begin{aligned}\{t'_i\} &= \left\{\cos\left[f.2\pi\frac{i}{f_a-1}+\mu.sen\left(f'.2\pi\frac{i}{f_a-1}\right)\right]\right\} = \\ &= \left\{\sum_{k=-\infty}^{+\infty} J_k(\mu)\cos\left[f.2\pi\frac{i}{f_a-1}+k.f'.2\pi\frac{i}{f_a-1}\right]\right\} = \\ &= \left\{\sum_{k=-\infty}^{+\infty} J_k(\mu)\cos\left[(f+k.f').2\pi\frac{i}{f_a-1}\right]\right\}\end{aligned} \tag{2.63}$$

onde

$$J_k(\mu) = \frac{2}{\pi}\int_0^{\frac{\pi}{2}}\left[cos\left(\overline{k}\,\frac{\pi}{2}+\mu.\sin w\right).cos\left(\overline{k}\,\frac{\pi}{2}+k.w\right)\right]dw \quad , \quad \overline{k} = k\%2 \ , \ k\in\mathbb{N} \tag{2.64}$$

é a função de Bessel (36, 40) que na FM especifica a amplitude de cada componente.

Nestas equações, a variação de frequência introduzida por $\{t'_i\}$ não respeita a progressão geométrica de frequência que acompanha a percepção de altura, mas sim a equação 2.33. A utilização das equações 2.58 para a FM está no Apêndice D, onde é calculado o conteúdo espectral da síntese FM obtida com oscilações na escala logarítmica. De fato, o comportamento simples que torna a FM atraente é obtido somente com as variações lineares em 2.63.

O caso da modulação de amplitude (AM) é mais simples:

$$\{t'_i\}_0^{\Lambda-1} = \{(1+a_i).t_i\}_0^{\Lambda-1} = \left\{\left[1+M.\sin\left(f'.2\pi\frac{i}{f_a-1}\right)\right].P.\sin\left(f.2\pi\frac{i}{f_a-1}\right)\right\}_0^{\Lambda-1} =$$
$$= \left\{P.\sin\left(f.2\pi\frac{i}{f_a-1}\right)+\frac{P.M}{2}\left[\sin\left((f-f').2\pi\frac{i}{f_a-1}\right)+\sin\left((f+f').2\pi\frac{i}{f_a-1}\right)\right]\right\}_0^{\Lambda-1} \tag{2.65}$$

Ou seja, o som resultante é o original e a reprodução de seu conteúdo espectral acima e abaixo da frequência original, distantes f' de f. Novamente, isso é obtido com a variação na escala linear de amplitude. No Apêndice D está uma exposição do espectro da AM realizada com a oscilação na escala logarítmica de amplitude. Esta também perde o comportamento simples.

A sequência T_i de frequência f, chamada portadora, é modulada pela f', chamada moduladora. No jargão de FM e AM, μ e $\alpha = 10^{\frac{V_{dB}}{20}}$ são chamados índices de modulação. Assim, para o padrão de vibração da moduladora $\{t'_i\}$ as equações:

$$\gamma'_i = \left\lfloor if'\frac{\widetilde{\Lambda}_M}{f_a}\right\rfloor \tag{2.66}$$

$$t'_i = \widetilde{m}_{\gamma'_i \,\%\widetilde{\Lambda}_M} \tag{2.67}$$

Para aplicação da moduladora $\{t'_i\}$ na portadora $\{t_i\}$ por FM:

$$f_i = f + \mu.t'_i \tag{2.68}$$

$$\Delta_{\gamma_i} = f_i\frac{\widetilde{\Lambda}}{f_a} \quad \Rightarrow \quad \gamma_i = \left\lfloor\sum_{j=0}^{i} f_j\frac{\widetilde{\Lambda}}{f_a}\right\rfloor = \left\lfloor\sum_{j=0}^{i}\frac{\widetilde{\Lambda}}{f_a}(f+\mu.t'_j)\right\rfloor \tag{2.69}$$

$$T_i^{f,FM(f',\mu)} = \left\{t_i^{f,FM(f',\mu)}\right\}_0^{\Lambda-1} = \left\{\widetilde{l}_{\gamma_i\%\,\widetilde{\Lambda}}\right\}_0^{\Lambda-1} \tag{2.70}$$

Onde $\widetilde{l}$ é um período da forma de onda de comprimento $\widetilde{\Lambda}$ da portadora.

Para realizar a AM, basta modular $\{t_i\}$ com $\{t'_i\}$ através das equações:

$$a_i = 1 + \alpha.t'_i \tag{2.71}$$

$$T_i^{f,AM(f',\alpha)} = \left\{t_i^{f,AM(f',\alpha)}\right\}_0^{\Lambda-1} = \{t_i.a_i\}_0^{\Lambda-1} = \{t_i.(1+\alpha.t'_i)\}_0^{\Lambda-1} \tag{2.72}$$

2.2.6 Usos musicais

A este ponto as possibilidades musicais explodiram. As características de altura (dada pela frequência), timbre (dado pela forma de onda e filtragens), volume (dado pela intensidade) e duração (dada pelo número de amostras) podem ser consideradas de forma absoluta ou tratadas ao longo de sua duração, com a única exceção da duração em si.

Desta forma, os usos musicais aqui dispostos são uma coleção de possibilidades com o objetivo de exemplificar manipulações sonoras que resultem algo musical, por razões variadas e aprofundadas na próxima seção.

Uma primeira possibilidade interessante é usar vínculos entre os parâmetros do tremolo e do vibrato e algum parâmetro da nota básica, como a frequência. Assim, pode-se estabelecer que a frequência do vibrato é diretamente proporcional à altura, e a profundidade do tremolo é inversamente proporcional à altura. Desta forma, com as equações 2.56, 2.58 e 2.61 pode-se escrever:

$$\begin{aligned} f^{vbr} = f^{tr} &= func_a(f) \\ \nu &= func_b(f) \\ V_{dB} &= func_c(f) \end{aligned} \tag{2.73}$$

Com f^{vbr} e f^{tr} como f' nas equações de referência, ou seja, a frequência de oscilação do vibrato e do tremolo da equação 2.56. Já ν e V_{dB} são as profundidades do vibrato e do tremolo, respectivamente. As funções $func_a$, $func_b$ e $func_c$ são arbitrárias e dependentes das intenções musicais. A montagem *Tremolos, vibratos e a frequência* explora recursos como este e variações da forma de onda da oscilação com vínculos, de modo a formar um *idioma musical*[24]. Seu código está no Apêndice B.2.3 e também disponível online como parte da caixa de ferramentas MASSA.

Com relação à convolução, pode-se estabelecer uma duração como pulso musical - a exemplo de um pulso BPM - e distribuir impulsos no decorrer deste pulso, de forma a estabelecer métricas e ritmos[25]. Por exemplo, 2 impulsos igualmente espaçados fazem uma divisão binária básica do pulso. Dois sinais, um com 2 pulsos e outro com 3 pulsos, ambos com os impulsos igualmente espaçados na duração do pulso, resultam na manutenção do pulso, com uma marcação rítmica usada tanto em divisões binárias quanto ternárias em diversos estilos de música étnica e tradicional.(41) Os próprios valores absolutos destes impulsos resultam em proporções entre as amplitudes dos sinais convoluídos. Este recurso da métrica estabelecida pela convolução com impulsos é explorado na montagem *Trenzinho de caipiras impulsivos*. Os recursos explorados incluem a criação de amálgamas sonoros provenientes da síntese granular e esta montagem é já uma ponte para a próxima seção. Veja especialmente a figura 2.25. O código da peça está na seção B.2.4 e disponível como parte da toolbox MASSA.

[24] Veja na próxima seção.
[25] Lembrando que a convolução com o impulso resulta no som deslocado ao instante de ocorrência do impulso.

Com os filtros as possibilidades explodem ainda mais vertiginosamente. Pode-se convoluir um sinal para reverberá-lo, para remover algum ruído, para gerar distorções ou para tratamento com intuito estético mesmo. Por exemplo, a aplicação de um filtro passa banda em que deixa passar somente entre $1kHz$ e $3kHz$, resulta em sons que parecem de telefone ou de televisão antiga. Ao remover com alguma precisão somente a frequência de oscilação da rede elétrica (usualmente $50Hz$ ou $60Hz$) e harmônicas, pode-se remover ruídos introduzidos por equipamentos de áudio. Um uso mais incrementado e propriamente musical é realizar filtragens em bandas específicas e usar estas bandas pré-estabelecidas como um parâmetro adicional das notas.

Inspirado em nos instrumentos tradicionais, pode-se aplicar uma a filtragem dependente do tempo (3). Cascatas destes filtros podem realizar filtragens complexas e mais precisas. A montagem *Ruidosa faixa* explora este recursos, realizando filtragens em ruídos diversos e síntese de ruídos diversos. O código da peça está no Apêndice B.2.5 e disponível online como parte da MASSA.

Estes recursos todos utilizados em conjunto podem incidir na realização de um efeito chamado *chorus*. A exemplo do que ocorre com um coro de cantores, neste efeito o som é realizado com diversas pequenas modificações, potencialmente aleatórias, em paramêtros como frequência central, presença (ou ausência) de vibrato e tremolo e suas características, equalizações, volume etc. Para o resultado final, estas versões do som inicial são então mixadas (ver equação 2.30). A peça *Chorus infantil* realiza chorus de formas diferentes em sons diferentes e seu código está no Apêndice B.2.7. A *toolbox* MASSA disponibiliza online este *script*.

A variação de volume no decorrer de um som é crucial para nossa percepção de timbre. A envoltória de volume chamada ADSR (sigla de *Atack-Decay-Sustain-Release*) possui numerosas implementações em sintetizadores em hardware e software. Uma implementação pioneira pode ser encontrada no Hammond Novachord de 1938 e algumas variantes são citadas logo abaixo.(42)

A envoltória ADSR escolástica é caracterizada por 4 parâmetros: duração do ataque (tempo que o som demora para atingir seu volume máximo), duração do decaimento (segue ao ataque imediatamente), nível de volume de sustentação (em que o volume fica estável após o decaimento) e duração de soltura (após a sustentação, nesta duração o volume decai a zero). Note que o tempo de sustentação não é especificado, pois é resultante da duração em si menos os tempos de ataque, decaimento e soltura.

A aplicação da envoltória ADSR com durações Δ_A, Δ_D e Δ_R, duração total Δ e nível de sustentação a_S, dado como fração da amplitude máxima, em uma sequência sonora t_i qualquer, pode ser feita da seguinte forma:

$$
\begin{aligned}
\{a_i\}_{0}^{\Lambda_A-1} &= \left\{ \xi \left(\frac{1}{\xi}\right)^{\frac{i}{\Lambda_A-1}} \right\}_{0}^{\Lambda_A-1} \qquad \text{ou} \qquad \left\{ \frac{i}{\Lambda_A-1} \right\}_{0}^{\Lambda_A} \\
\{a_i\}_{\Lambda_A}^{\Lambda_A+\Lambda_D-1} &= \left\{ a_S^{\frac{i-\Lambda_A}{\Lambda_D-1}} \right\}_{\Lambda_A}^{\Lambda_A+\Lambda_D-1} \qquad \text{ou} \qquad \left\{ 1-(1-a_S)\frac{i-\Lambda_A}{\Lambda_D-1} \right\}_{\Lambda_A}^{\Lambda_A+\Lambda_D-1} \\
\{a_i\}_{\Lambda_A+\Lambda_D}^{\Lambda-\Lambda_R-1} &= \{a_S\}_{\Lambda_A+\Lambda_D}^{\Lambda-\Lambda_R-1} \\
\{a_i\}_{\Lambda-\Lambda_R}^{\Lambda-1} &= \left\{ a_S \left(\frac{\xi}{a_S}\right)^{\frac{i-(\Lambda-\Lambda_R)}{\Lambda_R-1}} \right\}_{\Lambda-\Lambda_R}^{\Lambda-1} \qquad \text{ou} \qquad \left\{ a_S - a_S \frac{i+\Lambda_R-\Lambda}{\Lambda_R-1} \right\}_{\Lambda-\Lambda_R}^{\Lambda-1}
\end{aligned}
\tag{2.74}
$$

Com $\Lambda_X = \lfloor \Delta . f_a \rfloor \ \forall \ X \in (A, D, R,)$ e ξ um valor pequeno que torne o *fade in* e o *fade out* satisfatórios, e.g. $\xi = 10^{\frac{-80}{20}} = 10^{-4}$ ou $\xi = 10^{\frac{-40}{20}} = 10^{-2}$. Quanto menor for ξ mais lento é o *fade*, a exemplo de α da figura 2.14. Já os termos do lado direito de 2.74 podem realizar as entrada e saída do som a partir da intensidade zero, por serem lineares. Esquematicamente, a figura 2.2.6 mostra esta envoltória ADSR, uma implementação clássica que comporta variações diversas. Por exemplo, entre o ataque e o decaimento, pode-se adicionar uma partição adicional em que a amplitude máxima perdura. Outro exemplo comum é o uso de traçados mais elaborados para o ataque ou para o decaimento. A montagem musical *ADa e SaRa*, disponível no Apêndice B.2.8

e na MASSA, explora diversas destas configurações da envoltória ADSR

$$\left\{t_i^{ADSR}\right\}_0^{\Lambda-1} = \{t_i.a_i\}_0^{\Lambda-1} \tag{2.75}$$

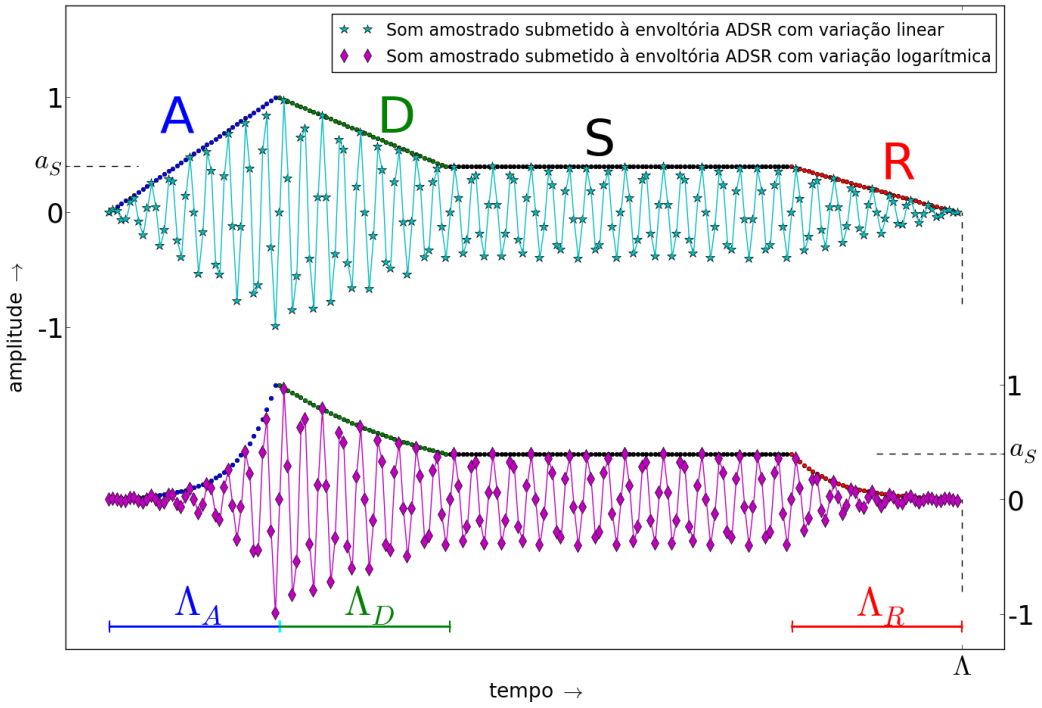

Figura 2.21 – *Envoltória ADSR (*Attack, Decay, Sustain, Release*) e uma sequência sonora arbitrária submetida à envoltória. A variação linear de amplitude está acima. Abaixo a variação de amplitude é exponencial.*

2.3 Organização de notas em música

Seja $S_j = \{T_{i,1}, T_{i,2}...T_{i,N}\} = \bigcup_{j\in[1,N]} T_{i,j}$ uma sequência de N eventos musicais. Para simplificar, sejam notas estes eventos musicais e S_j uma estrutura musical. Esta seção é dedicada às técnicas que tornam S_j interessante e agradável na audição.

Os elementos de S_j podem ser sobrepostos por mixagem, como na equação 2.30 e figura 2.11, formando intervalos e acordes. Resultam no 'pensamento vertical' da música. A concatenação de S_j, como na equação 2.31 e na figura 2.12, forma sequências melódicas e ritmos, associados ao 'pensamento horizontal' na música. A frequência fundamental f e o momento de início (ataque) são as características mais importantes dos elementos de S_j, possibilitando, grosso modo, a presença de músicas de alturas (harmonia e melodia) e a presença da métrica temporal e ritmos, respectivamente.

2.3.1 Afinação, intervalos, escalas e acordes

O dobro da frequência é uma oitava ascendente ($f = 2f_0$). Doze semitons equidistantes para o ouvido formam uma oitava, portanto, se $f = 2^{\frac{1}{12}} f_0$, há um semitom entre f_0 e f. O fator $\varepsilon = 2^{\frac{1}{12}}$, de um semitom, forma uma grade de notas no espectro audível. Fixada uma frequência f, as frequências fundamentais possíveis estão separadas por intervalos múltiplos de ε. Esta precisão absoluta é característica de implementações computacionais, pois os intrumentos reais possuem desvios destas frequências para mais bem compatibilizar os harmônicos de suas notas.(3)

Esta divisão da oitava em doze notas é o cânone da música ocidental clássica, além de incidir em usos cerimoniais/religiosos e étnicos.(7) Notados em proporções de ε entre as notas (i.e. um semitom), estes intervalos são, junto a suas notações abreviadas usuais:

$$
\begin{aligned}
\text{uníssono} &= 1J = 0 \\
\text{semitom ou segunda menor} &= 2m = 1 \\
\text{tom, tom inteiro ou segunda maior} &= 2M = 2 \\
\text{terça menor} &= 3m = 3 \\
\text{terça maior} &= 3M = 4 \\
\text{quarta justa} &= 4J = 5 \\
\text{quarta aumentada, quinta diminuta ou trítono} &= 4aum = 5dim = Tri = 6 \\
\text{quinta justa} &= 5J = 7 \\
\text{sexta menor} &= 6m = 8 \\
\text{sexta maior} &= 6M = 9 \\
\text{sétima menor} &= 7m = 10 \\
\text{sétima maior} &= 7M = 11 \\
\text{oitava} &= 8J = 12
\end{aligned}
\tag{2.76}
$$

Nesta nomenclatura, um dos padrões presentes no Brasil, um intervalo justo (uníssono, quarta, quinta e oitava), ou um intervalo maior (segunda maior, terça maior, sexta maior ou sétima maior), acrescido de um semitom, resulta em um intervalo aumentado. Um intervalo justo, ou um intervalo menor (segunda menor, terça menor, sexta menor ou sétima menor), decrescido de um semitom, resulta em um intervalo diminuto. Um intervalo maior, descrescido de um semitom, resulta em um intervalo menor. Um intervalo menor, acrescido de um semitom, resulta em um intervalo maior. Um intervalo aumentado, acrescido de um semitom, resulta em um intervalo 'mais que aumentado' e um intervalo diminuto, descrescido de um semitom, resulta

em um intervalo 'mais que diminuto'[26].(23)

Caso as notas sejam dispostas sequencialmente, o intervalo é melódico. A ordem das notas, primeiro a nota mais grave ou a mais aguda, resulta em um intervalo ascendente ou descendente, respectivamente. Passando a nota mais grave para a oitava acima, ou a nota mais aguda uma oitava para baixo, o intervalo é invertido. Um intervalo, somado à sua inversão, resulta 9 (7m inverte para 2M: $7+2=9$). Um intervalo maior invertido resulta em um intervalo menor e vice-versa. Um intervalo aumentado invertido resulta diminuto e vice-versa, assim como o mais que aumentado resulta em mais que diminuto e vice-versa. Um intervalo justo invertido resulta igualmente justo. Um intervalo maior que a oitava é classificado como intervalo composto e leva o nome do intervalo entre as mesmas notas, mas na mesma oitava.

Caso as notas soem simultaneamente, o intervalo é harmônico. Os intervalos de uníssono, oitava e quinta justa são consonâncias perfeitas. As terças e sextas são consonâncias imperfeitas. As segundas, sétimas e trítonos são considerados dissonantes. Esta classificação não é absoluta. A discrepância mais comum é com relação à quarta justa, que é tida como consonante perfeita ou dissonante de acordo com o contexto e arcabouço teórico. Pode-se considerá-la consonante como regra geral. A sétima menor por vezes é considerada consonante, assim como algumas culturas entoam o trítono como intervalo consonante. Por vezes, as segundas menores, sétimas maiores e trítonos são consideradas dissonãncias forte, as segundas maiores, sétimas menores consideradas dissonâncias fracas. As oitavas, quintas e uníssonos consonâncias fortes, terças e sextas consonâncias fracas.(3, 7)

Esta descrição resume a teoria tradicional dos intervalos musicais.(23) A montagem *Intervalos entre alturas* explora estes intervalos de formas isoladas e diversas. O código está no Apêndice B.3.1 e disponível online com a *toolbox* MASSA.(2)

[26] A necessidade da nomenclatura, notação e uso de intervalos aumentados/diminutos e mais que aumentados/diminutos é consequência do sistema tonal, em que os graus da escala (veja abaixo) são realmente notas diferentes, com funções e usos específicos. Assim, em uma escala de dó bemol maior, a tônica - primeiro grau - é dó bemol, não si, e a sensível - sétimo grau - é si bemol, não lá sustenido ou dó duplo bemol.

Na escala cromática de temperamento igual, existem 5 divisões perfeitamente simétricas, consideradas escalas pelas simetrias internas e usos fáceis que disso resultam. Como sequências de inteiros aos quais ε é elevado para multiplicar f_0, as escalas são:

$$\begin{aligned}
\text{cromática} &= E_i^c = \{e_i^c\}_0^{11} = \{0,1,2,3,4,5,6,7,8,9,10,11\} = \{i\}_0^{11} \\
\text{tons inteiros} &= E_i^t = \{e_i^t\}_0^5 = \{0,2,4,6,8,10\} = \{2.i\}_0^5 \\
\text{terças menores} &= E_i^{tm} = \{e_i^{tm}\}_0^3 = \{0,3,6,9\} = \{3.i\}_0^3 \\
\text{terças maiores} &= E_i^{tM} = \{e_i^{tM}\}_0^2 = \{0,4,8\} = \{4.i\}_0^2 \\
\text{trítonos} &= E_i^{tt} = \{e_i^{tt}\}_0^1 = \{0,6\} = \{6.i\}_0^1
\end{aligned} \tag{2.77}$$

Assim, a terceira nota da escala em tons inteiros com $f_0 = 200Hz$ é $f_3 = \varepsilon^{e_3^t}.f_0 = 2^{\frac{6}{12}}.200 \approxeq 282.843Hz$. Estas 'escalas' ou padrões, geram estruturas estáveis pelas simetrias internas, e podem ser repetidas de forma eficiente e sustentada. Abaixo há mais sobre simetrias e sobre estas divisões da oitava por igual. A montagem *Cristais* expõe cada uma destas escalas tanto melódica quanto harmonicamente e seu código está no Apêndice B.3.2. Como parte da MASSA, este código é disponibilizado online.

As escalas diatônicas:

$$
\begin{aligned}
\text{menor natural} = \text{modo eólico} = E_i^m &= \{e_i^m\}_0^6 = \{0,2,3,5,7,8,10\} \\
\text{modo lócrio} = E_i^{mlo} &= \{e_i^{mlo}\}_0^6 = \{0,1,3,5,6,8,10\} \\
\text{maior} = \text{modo jônico} = E_i^M &= \{e_i^M\}_0^6 = \{0,2,4,5,7,9,11\} \\
\text{modo dórico} = E_i^{md} &= \{e_i^{md}\}_0^6 = \{0,2,3,5,7,9,10\} \\
\text{modo frígio} = E_i^{mf} &= \{e_i^{mf}\}_0^6 = \{0,1,3,5,7,8,10\} \\
\text{modo lídio} = E_i^{ml} &= \{e_i^{ml}\}_0^6 = \{0,2,4,6,7,9,11\} \\
\text{modo mixolídio} = E_i^{mmi} &= \{e_i^{mmi}\}_0^6 = \{0,2,4,5,7,9,10\}
\end{aligned}
\tag{2.78}
$$

só possuem intervalos maiores, menores e justos. Única exceção para o trítono, que se apresenta como quarta aumenta ou quinta diminuta.

Todas as escalas diatônicas seguem o padrão de intervalos sucessivos tom, tom, semitom, tom, tom, tom, semitom. De forma que pode-se escrever:

$$
\begin{aligned}
\{d_i\} &= \{2,2,1,2,2,2,1\} \\
e_0 &= 0 \\
e_i &= d_{(i+\kappa)\%7} + e_{i-1} \quad \textit{para } \; i > 0
\end{aligned}
\tag{2.79}
$$

Com $\kappa \in \mathbb{N}$. Para cada modo, há um único valor de κ em qualquer intervalo $\in [x, x+6], \forall x \in \mathbb{N}$.

Por exemplo, uma breve inspeção revela que $e_i^{ml} = d_{(i+2)\%7} + e_{i-1}^{ml}$. Portanto, $\kappa = 2$ para o modo lídio.

A escala menor possui duas formas adicionais, melódica e harmônica:

$$\begin{aligned}
\text{menor natural (igual acima)} = E_i^m = \{e_i^m\}_0^6 = \{0,2,3,5,7,8,10\} \\
\text{menor harmônica} = E_i^{mh} = \{e_i^{mh}\}_0^6 = \{0,2,3,5,7,8,11\} \\
\text{menor melódica} = E_i^{mm} = \{e_i^{mm}\}_0^{14} = \{0,2,3,5,7,9,11,12,10,8,7,5,3,2,0\}
\end{aligned} \tag{2.80}$$

O traçado ascendente e descendente da escala menor melódica é necessário para a presença da sensível (sétimo e último grau separado por um semitom da oitava) no trajeto ascendente, o que não é necessário quando descende, retomando a forma natural. Já a escala harmônica apresenta a sensível, mas não evita o intervalo de segunda aumentada entre o sexto e sétimo graus, pois não precisa contemplar trajetória melódica, apenas apresentar a sensível tão cara ao sistema tonal (a sensível tende à tônica, afirmando-a).(30) Outras escalas podem ser representadas da mesma forma, como as pentatônicas e os modos de transposição limitados de Messiaen.(43)

A ocorrência simultânea de notas é observada através dos acordes. Destes, a base na música tonal são as tríades. Estas constituem-se de duas terças sucessivas, em 3 notas: fundamental, terça e quinta. Os acordes triádicos sem inversão e na forma fechada[27], com a fundamental em 0 são escritos;

$$\begin{aligned}
\text{tríade maior} = A_i^M = \{a_i^M\}_0^2 = \{0,4,7\} \\
\text{tríade menor} = A_i^m = \{a_i^m\}_0^2 = \{0,3,7\} \\
\text{tríade diminuta} = A_i^d = \{a_i^d\}_0^2 = \{0,3,6\} \\
\text{tríade aumentada} = A_i^a = \{a_i^a\}_0^2 = \{0,4,8\}
\end{aligned} \tag{2.81}$$

Basta acrescentar 10 ao final para ter o acorde (uma tétrade) com sétima menor ou 11 para resultar em sétima maior. As inversões e posições abertas podem ser obtidas contando o número

[27] Um acorde invertido é aquele que apresenta no grave outra nota que não a fundamental. A posição fechada é aquela em que não cabe nota alguma do acorde entre quaisquer duas notas consecutivas.(23)

de semitons de cada nota a partir da nota mais grave ou somando 12 à fundamental, terça ou quinta.

Para a realização de microtonalidade[28], pode-se usar reais não inteiros para descrever a sequência de alturas, ou modificar o fator ε e continuar usando inteiros. Por exemplo, uma afinação bastante próxima da série harmônica em si é proposta na forma da divisão da oitava em 53 notas: $\varepsilon = 2^{\frac{1}{53}}$.(44) As notas continuam inteiras com o novo ε. Note que se S_i é uma sequência de alturas relacionadas por ε_1, um mapeamento para notas em ε_2 resulta em uma nova sequência S'_i com $s'_i = s_i \frac{\varepsilon_1}{\varepsilon_2}$. A montagem musical *Micro tom* explora recursos microtonais e seu código está no Apêndice B.3.3, assim como na MASSA online.

2.3.2 Rudimentos de harmonia

Acordes como os listados em 2.81 formados em cada grau das escalas em 2.78 formam campos harmônicos das escalas. As progressões de acordes utilizadas e regras para seus encadeamentos é o objeto de estudo da harmonia musical. Mesmo uma melodia monofônica gera campos harmônicos.

Não respeitar os encadeamentos básicos do sistema tonal resulta em harmonias modal e atonal. Caso as notas coincidam com algum modo diatônico 2.78 ou sejam em número pequeno, pode-se dizer que a harmonia é modal. Caso contrário, atonal.(45).

Na harmonia tonal tradicional, as escalas maior e menor são base para tríades em cada grau da escala: $\hat{1}, \hat{2}, \hat{3}, \hat{4}, \hat{5}, \hat{6}, \hat{7}$. Pode-se anotar $\hat{1}, \hat{3}, \hat{5}$ para o acorde do primeiro grau, formado pelo primeiro grau da escala e central para uma música tonal. Secundários, o acorde do quinto grau $\hat{5}, \hat{7}, \hat{2}$ ($\hat{7}$ sustenido no caso da escala menor) e o acorde do quarto grau $\hat{4}, \hat{6}, \hat{1}$. Depois são

[28] O uso de intervalos menores que o semitom é chamado microtonalidade e tem usos ornamentais e estruturantes da música. A divisão da oitava em 12 notas possui fundamentos físicos mas não deixa de ser uma *convenção* adotada inclusive pela música erudita clássica de origem européia. Outras afinações são incidentes. Para citar somente um exemplo, a música tradicional tailandesa utiliza uma divisão da oitava em sete notas igualmente espaçadas ($\varepsilon = 2^{\frac{1}{7}}$), resultando em intervalos que pouco se assemelham aos intervalos na divisão de doze notas.(7)

considerados os outros graus. As convenções e técnicas apresentadas ao longo da história são objeto de estudo da harmonia tradicional.(30)

A chamada harmonia funcional atribui funções a estes três acordes centrais e busca compreender seus usos através destas funções. O acorde formado sobre o primeiro grau é o acorde de tônica e tem a função de manter um centro, um "chão" na música. O acorde formado sobre o quinto grau é a dominante e tem a função de tender à tônica, direcionar a música para ela. A tríade formada sobre o quarto grau é a subdominante e tem a função de se distanciar da tônica. O sistema se baseia em afirmar a tônica através de encadeamentos tônica-dominante-tônica expandidos com outros acordes das formas mais diversas.

A estes três acordes, são associadas as outras tríades. Na escala maior, a associada relativa (tônica relativa, subdominante relativa e dominante relativa) é a tríade formada uma terça abaixo e a associada anti-relativa (tônica anti-relativa, subdominante anti-relativa e a dominante anti-relativa) é a tríade formada na terça acima. Na escala menor ocorre o mesmo, mas a tríade a uma terça abaixo é chamada anti-relativa e a tríade a uma terça acima é chamada de relativa. As exatas funções e efeitos musicais destes acordes é motivo de bastante controvérsia. A tabela 2.22 mostra a relação entre as tríades formadas em cada grau da escala.

Tabela 2.22 *– Resumo das funções harmônicas tonais para a escala maior. A tônica é o centro da música, a dominante tende à tônica e a subdominante se distancia da tônica. Os três acordes podem, a princípio, serem substituídos livremente pela relativa ou antirelativa.*

relativa	acorde principal da função	antirelativa
$\hat{6},\hat{1},\hat{3}$	tônica: $\hat{1},\hat{3},\hat{5}$	$\hat{3},\hat{5},\hat{7}$
$\hat{3},\hat{5},\hat{7}$	dominante: $\hat{5},\hat{7},\hat{2}$	[$\hat{7},\hat{2},\hat{4}$#]
$\hat{2},\hat{4},\hat{6}$	subdominante: $\hat{4},\hat{6},\hat{1}$	$\hat{6},\hat{1},\hat{3}$

A dominante anti-relativa forma um acorde menor, exigindo a alteração do quarto grau um semitom para cima $\hat{7}$#. O acorde diminuto $\hat{7},\hat{2},\hat{4}$, geralmente é considerado uma 'dominante com sétima e sem fundamental'.(46)

Na escala menor, a relativa é a tríade iniciada uma terça acima e a antirelativa uma terça abaixo,

ou seja, se invertem com relação à tabela 2.22. Outro detalhe destas tríades em modo menor é a alteração do $\hat{7}$ um semitom acima para que fique somente a um semitom da tônica, permitindo a dominante (que deve ser maior e tender à tônica). Assim, a dominante é sempre maior, tanto em escalas maior ou menor. Por este motivo, a dominante relativa permanece a uma terça abaixo e a antirelativa a uma terça acima na escala menor.

Cada um destes acordes pode ser confirmado e se desenvolver com uma dominante e uma subdominante individuais (acorde com base na tríade formada a uma quinta acima e uma quinta abaixo respectivamente).

As medianas[29] são acordes formados também a uma terça, mas com alterações com relação ao acorde de origem. Caso haja duas alterações cromáticas, i.e. duas notas alteradas por um semitom cada com relação ao acorde original, a mediana é chamada 'duplamente cromática'. Esta relação entre acordes é considerada de tonalismo avançado, por vezes até de dissolução do tonalismo, e tem efeitos fortes e marcantes embora perfeitamente consonantes. As medianas foram utilizadas a partir do final do romantismo por Wagner, Lizt, Richard Strauss dentre outros e é bastante simples de ser realizada.(30, 47)

A modulação é a troca de tonalidade e se dão principalmente através de: uma dominante individual que afirma a mudança da tônica; alterações cromáticas para um acorde da nova tonalidade; e 'enarmonia' que é a mudança de função de um acorde com as mesmas notas. A importância da dominante torna ela o pivô natural das modulações, resultando no círculo das quintas.(30, 46–48)

A montagem musical *Acorde cedo* explora estas relações entre acordes. Seu código está disponível do Apêndice B.3.4 e online como parte da MASSA.(2)

[29] Também chamadas de medianas cromáticas.

2.3.3 Contraponto

A condução de linhas melódicas simultâneas, chamadas vozes, é matéria do campo de estudo chamado contraponto. A bibliografia geralmente percorre formas sistemáticas de condução de vozes e desemboca em gêneros escolásticos como cânones, invenções e fugas. É possível resumir regras principais do contraponto e é dito que o próprio Beethoven - dentre outros - esboçou uma síntese deste tipo.

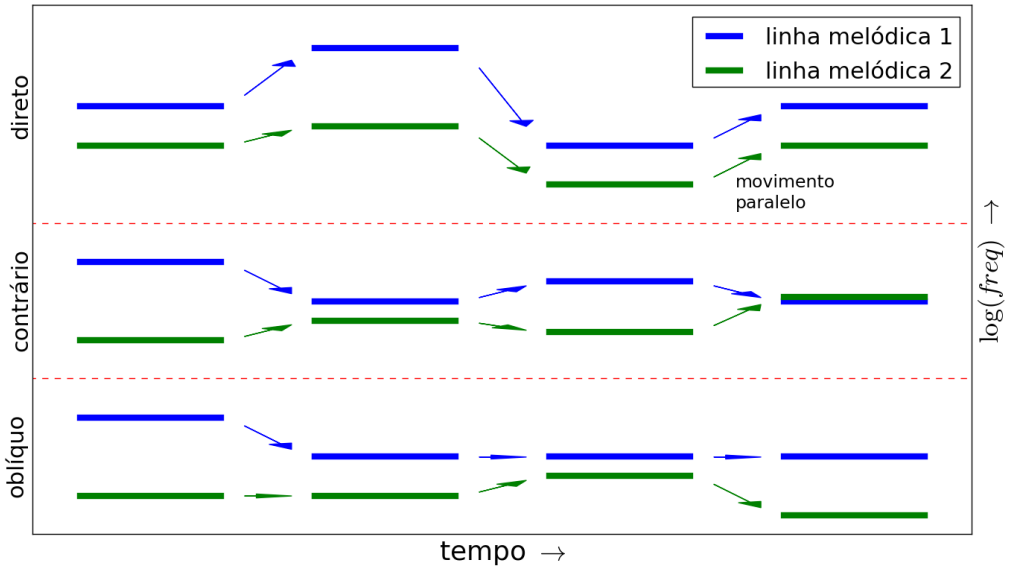

Figura 2.23 *– Movimentos diferenciados pelo contraponto. Visando preservar a independência entre as vozes, são distinguidos 3 tipos de movimentos.*

O propósito do contraponto é conduzir as vozes de forma que soem independentes. Cruciais para isso são as movimentações relativas, categorizadas em: movimento direto, oblíquo e contrário conforme a figura 2.23. O movimento paralelo é um movimento oblíquo. A regra de ouro é cuidar que os movimentos diretos não terminem em consonância perfeita. O movimento paralelo deve só ocorrer entre consonâncias imperfeitas e não mais do que três vezes consecutivas. As dissonâncias podem ser não admitidas ou usadas seguidas e precedidas de consonâncias em graus conjuntos. Os movimentos que levam a nota a uma vizinha soam mais coerentes. Na

presença de 3 ou mais vozes, a importância melódica recai sobre as vozes mais aguda e mais grave.(49–51)

Estas regras foram usadas na montagem *Conta ponto*, o código está no Apêndice B.3.5 e disponível online junto à MASSA.

2.3.4 Ritmo

A noção rítmica é depende de eventos separados por durações.(23) Estes eventos podem ser ouvidos individualmente se separados por ao menos $50 - 63ms$. Para que a separação entre eles possa ser apreciada como duração, ela deve ser maior, por volta de $100ms$.(13) Pode-se sumarizar a transição de durações ouvidas como alturas para a apreciação em ritmo da seguinte forma:(13, 52)

Tabela 2.24 *– Transição das durações ouvidas individualmente para alturas.*

	zona de percepção de durações em ritmo ←									transição		-					
duração (s)	**...**	**32,**	**16,**	**8,**	**4,**	**2,**	**1,**	**1/2,**	**1/4,**	**1/8,**	$\frac{1}{16} = 62,5ms$,	$\frac{1}{20} = 50ms$	1/40	1/80	1/160	1/320	1/640	...
frequência (Hz)	...	1/32,	1/16,	1/8,	1/4,	1/2,	1,	2,	4,	8,	16,	20	**40**	**80**	**160**	**320**	**640**	**...**
					-						transição		**zona de percepção de durações em altura →**					

A banda de durações marcada como transição está minimizada pois os limites não são bem definidos: a duração em que se começa a perceber uma frequência fundamental ou uma separação entre as ocorrências é dependente da pessoa e de características do som.(3, 13)

A métrica rítmica costuma se basear em uma duração básica chamada pulso. O pulso tipicamente compreende durações entre $0.25 - 1.5s$ (respectivamente 240 e $40BPM$). Na educação musical e estudos cognitivistas, costuma-se associar esta gama de frequências de pulsação às durações entre batidas do coração ou entre os passos ao caminhar.(3, 23)

O pulso é subdividido em partes iguais e também é repetido sequencialmente. Estas relações (de divisão e de concatenação) costumam seguir relações de números inteiros de baixa ordem

[30]. Uma exposição esquemática está na figura 2.25.

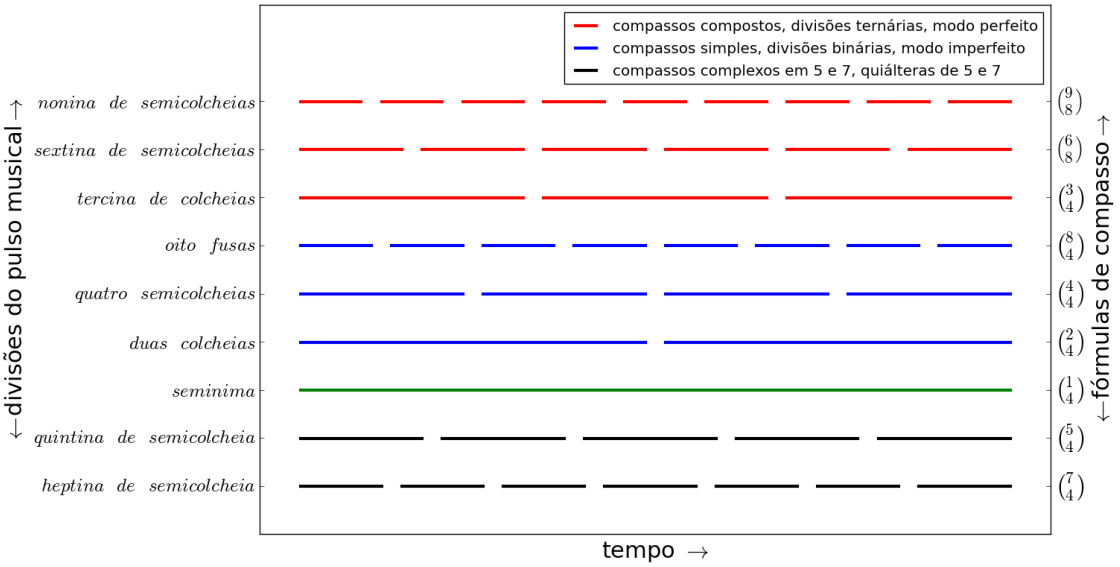

Figura 2.25 – *Divisões e aglomerações do pulso musical para estabelecimento de métrica. Ao lado esquerdo estão as divisões da semínima estabelecida como pulso. Ao lado direito, fórmulas de compasso que especificam as mesmas métricas, mas na escala das aglomerações do pulso musical.*

As relações duais (compassos simples e divisões binárias) costumam ocorrer em ritmos de dança e ocasiões festivas, e são chamadas imperfeitas. As relações triádicas incidem mais na música ritualística e relacionada ao sagrado e são ditas perfeitas.

As unidades mais fortes (acentuadas) são as que consistem nas 'cabeças das divisões'. A cabeça de uma unidade é a primeira parte da subdivisão. Nas divisões binárias (2, 4 e 8 dos casos considerados), as unidades consideradas fortes se revezam com as fracas (e.g. a divisão em 4 é forte, fraco, meio-forte, fraco). Nas divisões ternárias (3, 6 e 9) à unidade forte (primeira) se

[30] Em ordem crescente de ocorrência na música escrita e étnica, as divisões do pulso musical e seus agrupamentos sequenciais no tempo ocorrem com fatores 2, 4 e 8, depois 3, 6 (dois grupos de 3 ou 3 grupos de 2) e 9 e 12 (3 e 4 grupos de 3). Por último os primos 5 e 7, completando 1-9 e 12. Outras métricas são incidentes (divisões ou agrupamentos em 13, 17, etc), embora menos usuais e principalmente em contextos de música experimental e erudita do século XX e XXI. Por mais complexas que pareçam, as métricas costumam ser composições e decomposições de 1-9 partes iguais.(3, 41).

sucedem 2 unidades fracas (e.g. a divisão em 3 é forte, fraco fraco)[31].

A acentuação em tempo fraco é o contratempo, as notas iniciadas em tempo fraco e cuja duração se prolonga sobre tempos fortes são as síncopas.

As notas podem ocorrer dentro e fora destas divisões da *'métrica musical'*. Nos casos mais comportados, as notas ocorrem exatamente nestas divisões, com maior incidência em ataques nos tempos fortes. Em casos extremos, não se pode perceber a métrica.(3) Variações pequenas na grade ajudam a compor a interpretação musical e diferenças entre estilos.(10)

Pode-se convencionar o pulso como nível $j = 0$ de agrupamento, o nível $j = -1$ a primeira subdivisão do pulso, o nível $j = 1$ a primeira algomeração dos pulsos e assim por diante.

Desta forma, P_i^j é a i-ésima unidade de pulsos no nível j de agrupamento: P_{10}^0 é o décimo pulso, P_3^1 é a terceira unidade de agrupamento de pulsos (possivelmente terceiro compasso), P_2^{-1} é a segunda unidade da subdivisão do pulso.

Especial atenção para os limites de j: as divisões do pulso são durações apreciáveis como ritmo; as junções do pulso somam, no máximo do escopo, uma música ou um conjunto coeso de músicas. Dito de outra forma: a duração de $P_i^{min(j)}$, $\forall\ i$ é maior que $50ms$ e as durações de $P_i^{\text{máx}(j)}$ *somadas* em $\forall\ i$ devem resultar em uma duração menor do que alguns minutos ou, no máximo, poucas horas.

Cada nível j possui alguns índices i. Sempre que i soma três (ou múltiplo de três) índices há uma relação perfeita, quando soma dois, quatro ou oito índices há uma relação imperfeita, como na figura 2.25.

Qualquer unidade (nota), de uma dada sequência musical que tenha métrica pode ser univoca-

[31] A divisão em 6 é considerada composta mas pode ocorrer também como uma divisão binária. Uma divisão binária que sofre então uma divisão ternária resulta em duas unidades divididas em três unidades cada: forte (subdividido em forte fraco, fraco) e fraco (subdividido em forte, fraco, fraco). A outra forma de ocorrer a divisão em 6 é através de uma divisão ternária que sofre então uma divisão binária, resultando em: uma unidade forte (subdividido em forte e fraco) e duas unidades fracas (subdivididas em forte e fraco cada).

mente assim especificada:

$$P^{\{j_k\}}_{\{i_k\}} \tag{2.82}$$

em que j_k é o nível de aglomeração e i_k é a ordem da unidade em si.

Como um exemplo, $P^{-1,0,1}_{3,2,2}$ é a terceira subdivisão P^{-1}_3 do segundo pulso P^0_2 do segundo aglomerado de pulsos P^1_2. Cada unidade ou conjunto de unidades P^j_i pode ser associada a uma sequência de amostras temporais T_i que forma uma nota musical.

A montagem *Poli Hit Mia* utiliza as diferentes métricas e está no Apêndice B.3.6, disponível online como parte da MASSA.

2.3.5 Estruturas direcionais

Dadas as estruturas musicais básicas tanto frequenciais (acordes e escalas) quanto rítmicas (divisões e aglomerações simples, compostas e complexas), é natural pensar em como realizar estas estruturas de forma que tenha coesão e sentido.(53) Para tal, os arcos são a base: partindo de algum lugar e voltando, forma-se um arco. Uma música sem nenhum arco pode ser compreendida como uma música sem coesão. Na subseção seguinte exploramos arcos cíclicos e nesta os provenientes de estruturas direcionais.

Os arcos podem ser decompostos em duas sequências convergentes: uma que atinge o ápice (chamado literalmente de clímax pela teoria musical tradicional) e outra que, de forma paradigmática, volta do ápice à região de partida. Distingue-se entre arcos cujos clímax estão: no começo, no meio, no final, na primeira metade e na segunda metade como esquematicamente a dispostos na figura 2.26.(12)

Seja $U_i = \{u_i\}_0^{\Lambda_U - 1}$ uma sequência crescente. A sequência $R_i = \{r_i\}_0^{2\Lambda_U - 2} = \left\{u_{(\Lambda_U - 1 - |\Lambda_U - 1 - i|)}\right\}_0^{2\Lambda_U - 2}$ é uma sequência que apresenta simetria especular perfeita, i.e. a segunda metade é uma ver-

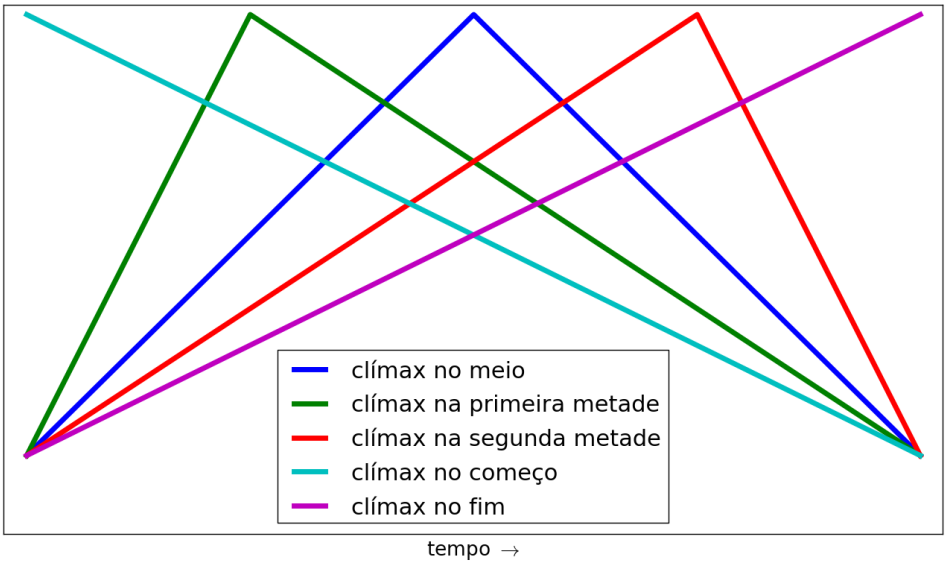

Figura 2.26 *– Distinções canônicas do clímax musical em uma melodia e outros domínios.*

são espelhada da primeira. Segundo conceitos musicais, o clímax está exatamente no meio da sequência. Pode-se modificar isso com a substituição por sequências de tamanhos diferentes. Toda a teoria matemática de sequências, já estabelecida, pode ser utilizada para geração destes arcos.(12, 54) Teoricamente, estas sequências aplicadas desta forma a qualquer característica dos eventos musicais produz arcos, pois resultam no afastamento e aproximação de uma parametrização inicial. Assim, é possível para uma mesma sequência de eventos possuir um número de arcos distintos, com tamanhos e clímax diferentes. Este é um recurso interessante e útil e a correlação dos arcos resulta na coerência da escuta.(47)

A montagem sonora *Dirracional* expõe estes arcos em estruturas direcionais. Seu código está no Apêndice B.3.7 e disponível online como parte da MASSA.(2)

2.3.6 Estruturas cíclicas

O entendimento filosófico de que o pensamento humano se baseia na percepção de semelhanças e diferenças dentre os estímulos e objetos coloca as simetrias no cerne do processo cognitivo.(55) Matematicamente, as simetrias são grupos algébricos e os grupos finitos são sempre isomorfos a um grupo de permutações. Pode-se dizer que as permutações representam quaisquer simetrias em um sistema finito. Na música, as permutações são ubíquas e estão presentes em técnicas, o que confirma seu papel central. A aplicação sucessiva das permutações gera arcos cíclicos.(11, 56, 57) A esta abordagem foram dedicados dois trabalhos acadêmicos para geração de estruturas musicais.(58, 59)

Qualquer conjunto de permutações pode ser utilizado como gerador de grupos algébricos.(57) As propriedades que definem um grupo G são:

$$\begin{aligned}
\forall\ p_1, p_2 \in G \Rightarrow \quad & p_1 \bullet p_2 = p_3 \in G && \text{(propriedade de fechamento)} \\
\forall\ p_1, p_2, p_3 \in G \Rightarrow \quad & (p_1 \bullet p_2) \bullet p_3 = p_1 \bullet (p_2 \bullet p_3) && \text{(propriedade da associatividade)} \\
\exists\ e \in G : \quad & p \bullet e = e \bullet p \quad \forall p \in G && \text{(existência do elemento neutro)} \\
\forall\ p \in G, \exists\ p^{-1} : \quad & p \bullet p^{-1} = p^{-1} \bullet p = e && \text{(existência do inverso)}
\end{aligned} \tag{2.83}$$

Da primeira propriedade conclui-se que toda permutação pode ser operada com outra permutação. De fato, pode-se aplicar uma permutação, depois outra e, se comparadas as ordenações inicial e final, há uma permutação resultante.

Todo elemento p operado consigo mesmo um número suficiente de vezes n atinge o elemento neutro $p^n = e$, pois caso contrário com a propriedade do fechamento se prova que o grupo é infinito (gerado por p). O menor $n\ :\ p^n = e$ é chamado de ordem do elemento. Assim, uma permutação finita p aplicada sucessivamente atinge a ordenação inicial dos elementos,

formando um ciclo. Este ciclo, se utilizado para parametros de notas musicais, resulta em um arco musical.

Estes arcos podem ser efetuados pelo uso conjunto de diferentes permutações. Como exemplo histórico, a tradição chamada *change ringing* concebe música através de sinos tocados um após o outro e então tocados novamente, mas em uma ordem diferente. Este processo se repete até que se atinja a ordenação inicial. O conjunto de ordenações diferentes percorridas é um *peal*. A tabela 2.27 representa um *peal* tradicional de 3 sinos (1, 2 e 3) que explora todas as suas ordenações. Cada linha apresenta uma ordenação dos sinos a ser tocada. As permutações estão entre as linhas. Neste caso, a estrutura musical consiste em permutações propriamente ditas e algumas permutações diferentes estão operando para resultar em um comportamento cíclico.

Tabela 2.27 – *Change Ringing:* Peal *(padrão) com 3 sinos*

1	2	3
2	1	3
2	3	1
3	2	1
3	1	2
1	3	2
1	2	3

A utilização de permutações na música pode ser resumida da seguinte forma: seja L_i uma sequência de eventos musicais (e.g. notas) e p uma permutação. $L'_i = p(L_i)$ consiste nos mesmos elementos de L_i com a ordenação modificada. As permutações são usualmente escritas em notação cíclica ou natural. Esta notação consiste na ordem dos índices resultante da permutação. Assim, convencionada a ordenação original dada pela sequência de seus índices [0 1 2 3 4 5 ...] a permutação é notada pela sequência que produz (ex. [1 3 7 0 ...]).

Não é necessário permutar os elementos de L_i, mas somente alguma ou algumas de suas características. Assim, seja p^f uma permutação nas frequências e L_i uma sequência de notas básicas como expostas ao final de 2.1.6. A nova sequência $L'_i = p^f(L_i)$ consiste nas mesmas notas

musicais, na mesma ordem e com as mesmas características, com as frequências fundamentais permutadas segundo o padrão que p^f apresenta.

Por último, duas sutilezas deste procedimento. Primeiro, a permutação p não precisa envolver todos os elementos de L_i, i.e. ela pode operar em subconjuntos de L_i. Segundo, nem todos os elementos l_i precisam ser executados a cada consulta de estado realizada. Para exemplificar, seja L_i uma sequência de notas musicais l_i. Se i vai de 0 a n, e $n > 4$, a cada compasso de 4 notas pode-se executar as primeiras 4 notas. As outras notas de L_i podem incidir nos compassos em que as permutações aloquem estas notas para as primeiras quatro notas de L_i.

Cada uma destas permutações p_i possui, ao menos, segundo a exposição acima: dimensões das notas em que opera (frequência, duração, *fades*, intensidade, etc) e período de incidência (a cada quantas consultas é aplicada a permutação). Na realização das notas de L_i, uma forma fácil e coerente é executar as primeiras n notas[32].

Esta abordagem resultou em alguns trabalhos e implementações computacionais.(58, 59) Abaixo está um uso coerente desta técnica abaixo e no Apêndice C apontada a implementação computacional disponibilizada em.(2)

2.3.7 Idioma musical?

Existem diversas empreitadas que se propõem a modelar e explorar entendimentos sobre a linguagem musical e da linguística aplicada à música ou ainda para discernimento entre o que seriam diferentes 'idiomas musicais'.(9, 30, 47, 52) De forma simplificada, um idioma musical é fruto da repetição de elementos e da repetição de relações entre elementos presentes no decorrer da música. Bastante relacionado é o uso de dicotomias como: repetição e variação, relaxamento e tensão, equilíbrio e desequilíbrio, consonância e dissonância, etc.

[32] A execução de notas disjuntas de L_i equivale a modificar a permutação e executar as primeiras notas.

2.3.8 Usos musicais

Primeiro, a nota básica foi definida e suas características postas em termos claros e quantitativos (seção 2.1.6). Em seguida, a composição interna da nota foi abordada, e compreendidas suas transições internas e tratamentos imediatos (seção 2.2). Por fim, esta presente seção dedica-se a organizar estas notas musicais em música. A gama de recursos com uma infinidade de possibilidades de resultados é situação típica e cara às artes.(8, 30)

Existem estudos para cada recurso apresentado. Por exemplo, pode-se obter as harmonias triádicas 'sujas' (com notas não pertencentes à tríade) através de sobreposições de quartas justas. Outro exemplo interessante é a presença simultânea de ritmos em diferentes métricas, constituindo o que chamamos de *polirritmia*. A montagem musical *Poli-hit mia* explora estas métricas simultâneas através de trem de impulsos convoluídos com as notas que compõem cada linha. Seu código está no Apêndice B.3.6 e disponível online como parte da MASSA.

As escalas microtonais são importantes na música do século XX (44) e possuem usos com resultados muito marcantes, como os quartos de tom na música indiana. A sequência musical *MicroTom* explora estes recursos, incluindo melodias microtonais e harmonias microtonais com várias notas em um âmbito de alturas bastante reduzido. Seu código está no Apêndice B.3.3 e disponível online como parte da MASSA.

Os vínculos entre parâmetros são formas poderosas de obter peças e montagens musicais. O número de notas permutadas pode variar no decorrer da música, revelando vínculo com a duração da peça. As harmonias podem consituir-se triádicas (eqs. 2.81) com notas replicadas em várias oitavas e mais numerosas quanto menor a profundidade e frequência de vibratos (eqs. 2.56, 2.57, 2.58, 2.59, 2.60), dentre outras incontáveis possibilidades.

As simetrias apresentadas nas divisões da oitava (eqs. 2.77) e das simetrias apresentadas através das permutações (tabela 2.27 e eqs. 2.83) podem ser usadas em conjunto. Nas peças *3 trios* esta

associação é feita de forma sistemática para possibilitar uma audição a ela dedicada. Esta é uma peça instrumental e não consta dentre os códigos dos Apêndices e da MASSA.

O *PPEPPS* (Pure Python EP: Projeto Solvente) é um EP sintetizado com os recursos apresentados neste trabalho. Com pouca parametrização, o programa gera músicas inteiras, permitindo a composição de músicas e conjuntos de músicas com facilidade. As possibilidades de compartilhamento permitem obter este album (ou outro feito de forma semelhante) através de poucas linhas de código e, pela execução, obter uma pasta com as músicas.

3 *Conclusões e trabalhos futuros*

No capítulo anterior está um sistema conciso que relaciona elementos musicais ao som digital. *Scripts* implementam estas relações, e em conjunto foram nomeados MASSA (Música e Áudio em Sequências e Séries Amostrais). A exposição didática destes desenvovimentos no capítulo anterior destina-se a facilitar a utilização do arcabouço.

As possibilidades abertas por estes resultados envolvem a criação de interfaces de geração de ruídos e outros sons em alta fidelidade (*hi-fi*), experimentos psicoacústicos e a utilização destes resultados para fins artísticos e didáticos. A incorporação de conhecimentos em programação é bastante facilitada através de recursos audiovisuais, o que já realizamos por práticas de *liveco-ding* e cursos focados em ferramentas especializadas, como o Puredata e o ChucK. Está prevista a utilização destes resultados com métodos generativos para geração de materiais artísticos.

A disposição online destes conteúdos na forma de hipertexto junto aos códigos e exemplos sonoros, todos em licenças livres, facilita colaborações e geração de subprodutos em co-autoria, e com isso a expansão da MASSA com novas implementações e desenvolvimentos das montagens musicais. Explorações sistemáticas de parametrizações (dos tremolos, da ADSR, etc) em alta fidelidade tem utilidade artística e é possibilidata por este trabalho com controle amostral.

Este trabalho também teve resultados não previstos, como a formação de grupos de interesse em torno da questão criativa aliada à computação. Neste contexto, destaca-se o grupo labMacambira.sf.net, que reúne colaboradores de todo o Brasil e alguns fora do país. Este grupo já

apresentou contribuições relevantes em diferentes áreas como Democracia Direta Digital, ferramentas de georeferenciamento e atividades artísticas e educacionais, como cursos, workshops e apresentações artísticas. Vários destes resultados estão apresentados no Apêndice E e no acervo online criado, que ultrapassa 700 vídeos, documentações escritas, diversos software originais e contribuições em software externos utilizados no mundo todo, como o Firefox, Scilab, LibreOffice, GEM/Puredata, para citar somente alguns exemplos.(60–62)

Há um aumento no número de pesquisas em música em andamento no campus de São Carlos da USP, o que sugere facilidade para estabelecer parcerias. As publicações acadêmicas efetivadas durante este mestrado também apontam para uma multidisciplinaridade, tratando diretamente de questões humanas como artes, filosofia, humor e linguagem falada e escrita, através de artifícios lineares e estatísticos.(66–69) Os desdobramentos estão alcançando redes sociais e teorias epistemológicas com base em pesquisas prévias, dos orientadores deste trabalho, com forte presença de redes complexas e processamento de linguagem natural.

REFERÊNCIAS

1 ZAMPRONHA, E. S. *Notação, representação e composição:* um novo paradigma da escritura musical. São Paulo: Annablume, 2000.

2 FABBRI, R.; COSTA, L. F.; OLIVEIRA JUNIOR, O. N. MASSA: música e áudio em sequências e séries amostrais. Disponível em: <http://labmacambira.git.sourceforge.net/git/gitweb.cgi?p=labmacambira/massa;a=blob;f=README.txt>. Acesso em: 18 nov. 2012.

3 ROEDERER, J. G. *The physics and psychophysics of music:* an introduction. 4th ed. New York: Springer Verlag, 2009. ISBN: 9780387094700.

4 MENEZES, F. *A acústica musical em palavras e sons*. Cotia: Ateliê Editorial, 2004. ISBN: 8574802166.

5 EVEREST, F. A. *Master handbook of acoustics*. New York: McGraw-Hill, 2001. ISBN: 0071360972.

6 ALTEN, S. R. *Audio in media*. 9th ed. Boston: Wadsworth Publishing, 2010. ISBN: 9780495572398.

7 WISNIK, J. M. *O som e o sentido*. São Paulo: Companhia das Letras, 1999. ISBN: 8571640424.

8 WEBERN, A. *O caminho para a música nova*. São Paulo: Novas Metas, 1984.

9 LERDAHL, F.; JACKENDOFF, R. *A generative theory of tonal music*. Cambridge: MIT Press, 1983. ISBN: 026262107X.

10 COOK, P. R. *Real sound synthesis for interactive applications*. Natick: A K Peters, 2002. ISBN: 1568811683.

11 ZAMACOIS, J. *Curso de formas musicales:* con numerosos ejemplos musicales. Barcelona: Idea Books, 2002. ISBN: 8482362356.

12 SCHOENBERG, A.; STRANG, G.; STEIN, L.; SEINCMAN, E. *Fundamentos da composição musical*. São Paulo: EDUSP, 1991. ISBN: 8531400457.

13 ROADS, C. *Microsound.* Cambridge: MIT Press, 2004. ISBN: 9780262681544.

14 CHACON, S.; HAMANO, J.; PEARCE, S. *Pro Git*. Berkeley: Apress, 2009. ISBN: 9781430218333.

15 OLIPHANT, T. E. A *guide to numPy*. Spanish Fork: Trelgol Publishing, 2006.

16 COURNAPEAU, D. *Audiolab, a python package to make noise with numpy arrays*. Disponível em: <http://www.ar.media.kyoto-u.ac.jp/members/david/softwares/audiolab>. Acesso em: 18 jan. 2013.

17 VAN ROSSUM, G.; DRAKE JR, F. L. *Python tutorial*. S. l.: Odense Universitet, Institut for Matematik og Datalogi, 1995.

18 VAN ROSSUM, G. V.; DRAKE JR, F. *Python reference manual*. S. l.: Iuniverse, 1995. ISBN: 9780595136681.

19 RAYMOND, E. S. *The art of Unix programming*. Boston: Addison-Wesley, 2004. ISBN: 0131429019.

20 LESSIG, L. *Free culture:* how big media uses technology and the law to lock down culture and coltrol creativity. New York: Penguin Press, 2004. ISBN: 9781594200069.

21 EMMERSON, S. Living electronic music. Aldershot: Ashgate, 2007. ISBN: 9780754655480.

22 LOVELOCK, W.; HOLLOWAY, E. A concise history of music. London: Bell, 1953. ISBN: 9780713506785.

23 LACERDA, O. Compêndio de teoria elementar da música. 3. ed. São Paulo: Ricordi Brasileira, 1966.

24 CHOWNING, J. M. Digital sound synthesis, acoustics and perception: a rich intersection. In: COST G-6 CONFERENCE ON DIGITAL AUDIO EFFECTS (DAFX-00), 2000, Verona. *Proceedings…* Verona: University of Verona, 2000. p. DAFX-1-DAFX-6.

25 OPPENHEIM, A. V.; SCHAFER, R. W. *Discrete-time signal processing*. Upper Saddle River: Prentice-Hall, 1989. ISBN: 9780131988422.

26 CHENG, C. I.; WAKEFIELD, G. H. Introduction to head-related transfer functions (HRTFs): representations of HRTFs in time, frequency, and space. *Journal of the Audio Engineering Society*, v. 49, n. 4, p. 231-249, 2001.

27 HEEGER, D. Perception lecture notes: auditory pathways and sound localization. Disponível em: <http://www.cns.nyu.edu/~david/courses/perception/lecturenotes/localization/localization.html>. Acesso em: 16 out. 2012.

28 ALGAZI, V.; DUDA, R. O.; THOMPSON, D. M.; AVENDANO, C. The cipic hrtf database. In: IEEE WORKSHOP ON APPLICATIONS OF SIGNAL PROCESSING TO AUDIO AND ACOUSTICS, 2001, New York. *Proceedings.*.. Piscataway: Institute of Electrical and Electronics Engineers, 2001. p. W2001-1- W2001-4.

29 CARTY, B.; LAZZARINI, V. Binaural HRTF based spatialisation: new approaches and implementation. In: INTERNATIONAL CONFERENCE ON DIGITAL AUDIO EFFECTS (DAFX-09), 12., 2009, Como. *Proceedings.*.. London: Department of Electronic Engineering - Queen Mary University of London, 2009. p. 49-54.

30 SCHOENBERG, A. *Harmonia*. São Paulo: Ed. UNESP, 1999.

31 GEIGER, G. Table lookup oscillators using generic integrated wavetables. In: INTERNATIONAL CONFERENCE ON DIGITAL AUDIO EFFECTS (DAFX-06), 9., 2006, Montreal. *Proceedings.*.. Montreal: McGill University, 2006. p. DAFX-169-DAFX-172.

32 BRISTOW-JOHNSON, R. Wavetable synthesis 101, a fundamental perspective. In: AES CONVENTION, 101., 1996, Los Angeles. *Proceedings.*... Disponível em: <http://www.musicdsp.org/files/Wavetable-101.pdf>. Acesso em: 22 nov. 2012.

33 DEHAENE, S. The neural basis of the Weber–Fechner law: a logarithmic mental number line. *Trends in Cognitive Sciences*, v. 7, n. 4, p. 145–147, 2003.

34 GUILLAUME, P. *Music and acoustics:* from instrument to computer. London: ISTE, 2006. ISBN: 9781905209262.

35 SMITH, S. W. *The scientist and engineer's guide to digital signal processing*. 2nd. ed. San Diego: California Technical Publishing, 1999.

36 SMITH III, J. O. *Mathematics of the Discrete Fourier Transform (DFT) with audio applications*. 2nd ed. 2007. ISBN: 9780974560748. Disponível em: <https://ccrma.stanford.edu/~jos/log>. Acesso em: 16 out. 2012.

37 MUSIC-DSP source code archive. Disponível em: <musicdsp.org>. Acesso em: 23 jan. 2013.

38 PORRES, A. T.; FURLANETE, F.; MANZOLLI, J. Análise de dissonância sensorial de espectros sonoros. In: CONGRESSO DA ANPPOM, 16., 2006, Brasília. *Anais*... Brasília: UnB, 2006.

39 PORRES, A.; PIRES, A. Um external de aspereza para puredata & MAX/MSP. In: SIMPÓSIO BRASILERIO DE COMPUTAÇÃO MUSICAL, 12., 2009, Recife. *Anais*... Recife: Sociedade Brasileira de Computação (SBC), 2009.

40 SCHOTTSTAEDT, B. *An introduction to FM*. Disponível em: <https://ccrma.stanford.edu/software/snd/snd/fm.html>. Acesso em: 12 ago. 2012.

41 GRAMANI, J. E. *Rítmica viva:* a consciência musical do ritmo. Campinas: UNICAMP, 1996.

42 ROADS, C. *The computer music tutorial*. Cambridge: MIT Press, 1996.

43 CLOUGH, J.; ENGEBRETSEN, N.; KOCHAVI, J. Scales, sets, and interval cycles: a taxonomy. *Music Theory Spectrum*, v.21, n. 1, p. 74–104, 1999.

44 WILKINSON, S. R. *Tuning in*: microtonality in electronic music: a basic guide to alternate scales, temperaments, and microtuning using synthesizers. Milwaukee: H. Leonard Books, 1988.

45 KOSTKA, S. et al. *Tonal harmony*: with an introduction to twentieth. New York: McGraw-Hill, 1995.

46 KOELLREUTTER, H. *Harmonia funcional*. São Paulo: Ricordi, 1986.

47 SALZER, F. *Structural hearing*: tonal coherence in music. New York: Dover Publications, 1962.

48 ALDWELL, E.; SCHACHTER, C.; CADWALLADER, A. *Harmony & voice leading.* Boston: Wadsworth Publishing Company, 2010.

49 FUX, J.; MANN, A. *The study of counterpoint*: from Johann Joseph Fux's gradus ad Parnassum. New York: WW Norton & Company, 1965. ISBN:-10: 0393002772.

50 TRAGTENBERG, L. *Contraponto:* uma arte de compor. 2. ed. São Paulo: Edusp, 2002.

51 SCHOENBERG, A.; STEIN, L. *Preliminary exercises in counterpoint.* London: Faber & Faber, 1963.

52 ASSIS, G. O. A. *Em busca do som*. São Paulo: Ed. UNESP. ISBN: 8539302079.

53 BOULEZ, P. A música hoje. 3. ed. São Paulo: Perspectiva, 2005. ISBN: 8527302896.

54 GUIDORIZZI, H. L. *Um curso de cálculo.* 5. ed. Rio de Janeiro: Livros Técnicos e Científicos Editora, 2001. v. 3.

55 DELEUZE, G.; GUATTARI, F. *What is philosophy?* New York: Columbia University Press, 1996. ISBN: 9780231079891.

56 DUCKWORTH, R.; STEDMAN, F. *Tintinnalogia, or, the art of ringing*. Teddington: Echo Library, 2007. ISBN: 9781406826203

57 BUDDEN, F. J. *The fascination of groups*. Cambridge: Cambridge University Press, 1972.

58 FABBRI, R.; MAIA JR, A. Applications of group theory on granular synthesis. In: SIMPÓSIO BRASILERIO DE COMPUTAÇÃO MUSICAL, 11., 2007, São Paulo. *Anais...* São Paulo: Sociedade Brasileira de Computação (SBC), 2007.

59 FABBRI, R.; MAIA JR, A. Applications of group theory on sequencing and spatialization of granular sounds. In: AES Brazil Conference, 6., 2008, São Paulo. *Proceedings...* São Paulo: Sociedade Brasileira de Computação (SBC), 2008. v. 1, p. 1-3.

60 LABORATÓRIO Macambira. #labmacambira @ Freenode. Disponível em: <http://labmacambira.sourceforge.net>. Acesso em: 18 jan. 2013.

61 WIKI do Lab Macambira. #labmacambira @ Freenode. Disponível em: <http://wiki.nosdigitais.teia.org.br/Lab_Macambira>. Acesso em: 18 jan. 2013.

62 CANAL Vimeo do Lab Macambira (mais de 700 videos). #labmacambira @ Freenode. Disponível em: <https://vimeo.com/channels/labmacambira>. Acesso em: 18 jan. 2013.

63 FABBRI, R. Repositório da Dissertação. Disponível em: <http://labmacambira.git.sourceforge.net/git/gitweb.cgi?p=labmacambira/dissertacao;a=tree;h=refs/heads/msc>. Acesso em: 19 jan. 2013.

64 FABBRI, R.; TAVARES, T. Audioexperiments. Disponível em: <https://www.assembla.com/spaces/audioexperiments/wiki>. Acesso em: 12 jan. 2013.

65 FABBRI, R.; MENDES, R. S. Compressão de Áudio via wavelets, aproximações polinomiais e permutações. In: ENCONTRO DOS ALUNOS E DOCENTES DO DEPARTAMENTO DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL (EADCA), 2., 2009, Campinas. *Anais...* Campinas: Faculdade de Engenharia Elétrica e de Computação - Universidade Estadual de Campinas, 2009. p. 155-158.

66 AMANCIO, D. R.; FABBRI, R.; OLIVEIRA JUNIOR, O. N.; NUNES, M. G. V.; COSTA, L. F. Distinguishing between positive and negative opinions with complex network features. In: ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS (ACL), 48.; WORKSHOP ON GRAPH-BASED METHODS FOR NATURAL LANGUAGE PROCESSING, 2010, Uppsala. *Proceeding.* Stroudsburg: Association for Computational Linguistics (ACL), 2010. p. 83–87.

67 FABBRI, R. et al. Speech polarity detection using complex networks measures: first explorations. In: INTERNATIONAL WORKSHOP ON COMPLEX NETWORKS (CompleNet), 2., 2010, Rio de Janeiro. *Final Program.…* Rio de Janeiro: Universidade Federal do Rio de Janeiro (UFRJ), 2010. p. 9.

68 AMANCIO, D. R.; FABBRI, R.; OLIVEIRA JUNIOR, O. N.; NUNES, M. G. V.; COSTA, L. F. Opinion discrimination using complex network features. *Communications in Computer and Information Science*, v. 116, p. 154–162, 2011.

69 VIEIRA, V.; FABBRI, R.; TRAVIESO, G.; OLIVEIRA JUNIOR, O. N.; COSTA, L. F A quantitative approach to evolution of music and philosophy. *Journal of Statistical Mechanics:* theory and experiment, v. 2012, n. 8, p. P08010.

APÊNDICE A – Código computacional dos procedimentos expostos no capítulo 2

A.1 Código Python das relações descritas na seção 2.1

Todas as equações da seção 2.1 estão implementadas abaixo em Python e disponibizadas como parte da toolbox MASSA.(2)

equações da seção 2.1 em Python

```
#-*- coding: utf8 -*-
import numpy as n
import scikits.audiolab as a

############ 2.1.1 Duração
# a equação relaciona o número de amostras à duração do som
f_a = 44100.  # frequência de amostragem
Delta = 3.7  # duração de Delta segundos  = >

Lambda = int(f_a*Delta)  # número de amostras
# 2.1
T_i = n.zeros(Lambda)  # silêncio com ~ Delta segundos

# escrita em disco como arquivo PCM (WAV no caso)
a.wavwrite(T_i, "silencio.wav", f_a)

############ 2.1.2 Volume
Lambda = 100.  # 100 amostras
T_i = n.random.random(Lambda)  # 100 amostras quaisquer

# 2.2  Potência
```

```
pot = (T_i**2.).sum()/Lambda

T2_i = n.random.normal(size=Lambda)
pot2 = (T2_i**2.).sum()/Lambda  # potência 2
# 2.3 Diferença de volume em decibels, dadas as potências
V_dB = 10.*n.log10(pot2/pot)

# 2.4 dobra amplitude  = > ganha 6 dB
T2_i = 2.*T_i
pot = (T_i**2.).sum()/Lambda  # potência
pot2 = (T2_i**2.).sum()/Lambda  # potência 2
V_dB = 10.*n.log10(pot2/pot)
Mm6 = abs(V_dB - 6) < .5  # Mm6 é True

# 2.5 dobra potência  = > ganha 3 dB
pot2 = 2.*pot
V_dB = 10.*n.log10(pot2/pot)
Mm3 = abs(V_dB - 3) < .5  # Mm3 é True

# 2.7 dobra volume  = > + 10 dB  = > aplitude * 3.16
V_dB = 10.
A = 10.**(V_dB/20.)
T2_i = A*T_i  # A ~ 3.1622776601

# 2.8 Conversão de decibels em amplificação
A = 10.**(V_dB/20.)


########## 2.1.3 Altura
f_0 = 441.
lambda_0 = f_a/f_0
periodo = n.arcsin(n.random.random(lambda_0))  # amostras quaisquer
# 2.9 Som com frequência fundamental f_0
Tf_i = n.array(list(periodo)*1000)  # 1000 períodos

# normalizando para convenção no intervalo [-1,1]
Tf_i = ((Tf_i-Tf_i.min())/(Tf_i.max()-Tf_i.min()))*2.-1.
a.wavwrite(Tf_i, "f_0.wav", f_a)  # escrita em disco


########## 2.1.4 Timbre
T = 100000.  # número de amostras das sequencias
ii = n.arange(T)
f = 220.5
lambda_f = f_a/f
# 2.10 Senoide
Sf_i = n.sin(2.*n.pi*f*ii/f_a)
# 2.11 Dente de serra
```

```
Df_i = (2./lambda_f)*(ii % lambda_f)-1
# 2.12 Triangular
Tf_i = 1.-n.abs(2.-(4./lambda_f)*(ii % lambda_f))
# 2.13 Onda quadrada
Qf_i = ((ii % lambda_f) < (lambda_f/2))*2-1

Rf_i = a.wavread("22686__acclivity__oboe-a-440_periodo.wav")[0]
# 2.14 Período amostrado
Tf_i = Rf_i[n.int64(ii) % len(Rf_i)]


############ 2.1.5 O espectro no som amostrado
Lambda = 50.
T_i = n.random.random(Lambda)*2.-1.
C_k = n.fft.fft(T_i)
A_k = n.real(C_k)
B_K = n.imag(C_k)
w_k = 2.*n.pi*n.arange(Lambda)/Lambda


# 2.15 Recomposição do espectro no tempo
def t(i):
    return (1./Lambda)*n.sum(C_k*n.e**(1j*w_k*i))


# 2.16 Recomposição real
def tR(i):
    return (1./Lambda)*n.sum(n.abs(C_k)*n.cos(w_k*i-n.angle(C_k)))

# 2.17 Número de coeficientes espectrais pareados
tau = (Lambda - Lambda % 2)/2 + Lambda % 2-1
# 2.18 Coeficientes equivalentes
kk = n.arange(tau)
F_k = C_k[1:tau+1]
F2_k = C_k[Lambda-tau:Lambda-1][::-1]

# 2.19 Coeficientes equivalentes: módulos
ab = n.abs(F_k)
ab2 = n.abs(F2_k)
MIN = n.abs(ab-ab2).sum()  # MIN ~ 0.0
# 2.20 Coeficientes equivalentes: fases
an = n.angle(F_k)
an2 = n.angle(F2_k)
MIN = n.abs(an+an2).sum()  # MIN ~ 0.0

# 2.21 Combinação das componentes em cada amostra
w_k = 2*n.pi*n.arange(Lambda)/Lambda

```

```
118
119 def t_(i):
120     return (1./Lambda)*(A_k[0]+2.*n.sum(n.abs(C_k[1:tau+1]) *
121                          n.cos(w_k*i-n.angle(C_k)) + A_k[Lambda/2] *
122                          (1-Lambda % 2)))
123
124
125 ############ 2.1.6 A nota básica
126 f = 220.5  # Herz
127 Delta = 2.5  # segundos
128 Lambda = int(2.5*f_a)
129 ii = n.arange(Lambda)
130 Lf_i = Df_i  # Já fizemos Df_i acima
131 # 2.24 Nota Básica
132 TfD_i = Lf_i[ii % len(Lf_i)]
133
134
135 ############ 2.1.7 Localização espacial
136 zeta = 0.215  # metros
137 # tomemos localizacao (x,y) qualquer
138 x = 1.5  # metros
139 y = 1.  # metros
140 # 2.25 distâncias de cada ouvido
141 d = n.sqrt((x-zeta/2)**2+y**2)
142 d2 = n.sqrt((x+zeta/2)**2+y**2)
143 # 2.26 Distância de Tempo Interaural
144 DTI = (d2-d)/343.2  # segundos
145 # 2.27 Distância de Intensidade Interaural
146 DII = 20*n.log10(d/d2)  # dBs
147
148 # 2.28 aplicação de DTI e DII em T_i
149 Lambda_DTI = int(DTI*f_a)
150 DII_a = d/d2
151 T_i = 1-n.abs(2-(4./lambda_f)*(ii % lambda_f))  # triangular
152 T2_i = n.hstack((n.zeros(Lambda_DTI), DII_a*T_i))
153 T_i = n.hstack((T_i, n.zeros(Lambda_DTI)))
154
155 som = n.vstack((T2_i, T_i)).T
156 a.wavwrite(som, "estereo.wav", f_a)
157 # espelhando
158 som = n.vstack((T_i, T2_i)).T
159 a.wavwrite(som, "estereo2wav", f_a)
160
161 # 2.29 ângulo do objeto
162 theta = n.arctan(y/x)
163
164
165 ############ 2.1.8 Usos musicais
```

```
166 Delta = 3.  # 3 segundos
167 Lambda = int(Delta*f_a)
168 f1 = 200.  # Hz
169 foo = n.linspace(0., Delta*f1*2.*n.pi, Lambda, endpoint=False)
170 T1_i = n.sin(foo)  # senoide de Delta segundos e freq  =  f1
171
172 f2 = 245.  # Hz
173 lambda_f2 = int(f_a/f2)
174 T2_i = (n.arange(Lambda) % lambda_f < (lambda_f2/2))*2-1  # quadrada
175
176 f3 = 252.  # Hz
177 lambda_f3 = f_a/f3
178 T3_i = n.arange(Lambda) % lambda_f3  # Dente de serra
179 T3_i = (T3_i/T3_i.max())*2-1
180
181 # 2.30 mixagem
182 T_i = T1_i+T2_i+T3_i
183 # normalização
184 T_i = ((T_i-T_i.min())/(T_i.max()-T_i.min()))*2-1
185 # escrita em disco
186 a.wavwrite(T_i, "mixados.wav", f_a)
187
188 # 2.31 concatenação
189 T_i = n.hstack((T1_i, T2_i, T3_i))
190 # escrita em disco
191 a.wavwrite(T_i, "concatenados.wav", f_a)
```

A.2 Código Python das relações descritas na seção 2.2

Todas as equações da seção 2.2 estão implementadas abaixo em Python e disponibizadas como parte da toolbox MASSA.(2)

equações da seção 2.2 em Python

```
#-*- coding: utf8 -*-
import numpy as n
import scikits.audiolab as a

f_a = 44100  # Hz, frequência de amostragem

############ 2.2.1 Tabela de busca (LUT)
# tamanho da tabela: use par para não conflitar abaixo
# e ao menos 1024
Lambda_tilde = Lt = 1024

# Senoide
foo = n.linspace(0, 2*n.pi, Lt, endpoint=False)
S_i = n.sin(foo)  # um período da senoide com T amostras

# Quadrada:
Q_i = n.hstack((n.ones(Lt/2)*-1, n.ones(Lt/2)))

# Triangular:
foo = n.linspace(-1, 1, Lt/2, endpoint=False)
Tr_i = n.hstack((foo, foo*-1))

# Dente de Serra:
D_i = n.linspace(-1, 1, Lt)

# som real, importar período e
# usar T correto: o número de amostras do período
Rf_i = a.wavread("22686__acclivity__oboe-a-440_periodo.wav")[0]

f = 110.  # Hz
Delta = 3.4  # segundos
Lambda = int(Delta*f_a)

# Amostras:
ii = n.arange(Lambda)

# 2.32 LUT
Gamma_i = n.array(ii*f*Lt/f_a, dtype=n.int)
# Pode-se usar S_i, Q_i, D_i ou qualquer período de som real
```

```
# suficientemente grande
L_i = Tr_i
TfD_i = L_i[Gamma_i % Lt]


############ 2.2.2 Variações incrementais de frequência e amplitude
# VARIAÇÕES DE FREQUÊNCIA
f_0 = 100.  # freq inicial em Hz
f_f = 300.  # freq final em Hz
Delta = 2.4  # duração

Lambda = int(f_a*Delta)
ii = n.arange(Lambda)
# 2.33 - variação linear
f_i = f_0+(f_f-f_0)*ii/(float(Lambda)-1)
# 2.34 coeficientes para a LUT
D_gamma_i = f_i*Lt/f_a
Gamma_i = n.cumsum(D_gamma_i)
Gamma_i = n.array(Gamma_i, dtype=n.int)
# 2.35 som resultante
Tf0ff_i = L_i[Gamma_i % Lt]

## 2.36 - variação exponencial
f_i = f_0*(f_f/f_0)**(ii/(float(Lambda)-1))
# 2.37 coeficientes para a LUT
D_gamma_i = f_i*Lt/f_a
Gamma_i = n.cumsum(D_gamma_i)
Gamma_i = n.array(Gamma_i, dtype=n.int)
# 2.38 som resultante
Tf0ff_i = L_i[Gamma_i % Lt]


# VARIAÇÕES DE AMPLITUDE
# sintetizando um som qualquer para
# a variação de amplitude
f = 220.  # Hz
Delta = 3.9  # segundos
Lambda = int(Delta*f_a)

# Amostras:
ii = n.arange(Lambda)

# (como em 2.30)
Gamma_i = n.array(ii*f*Lt/f_a, dtype=n.int)
L_i = Tr_i  # pode-se usar igualmente S_i, Q_i, D_i ou
# qualquer período de som real suficientemente grande
T_i = TfD_i = L_i[Gamma_i % Lt]

```

```
a_0 = 1.  # razão da amplitude em que é iniciada a sequência
a_f = 12.  # razão da amplitude em que é finalizada
alpha = 1.  # índice de suavidade da transição
# 2.39 envoltória exponencial para transição de amplitude
A_i = a_0*(a_f/a_0)**((ii/float(Lambda))**alpha)
# 2.40 aplicação da envoltória no som T_i
T2_i = A_i*T_i

# 2.41 envoltória linear de amplitude
A_i = a_0+(a_f-a_0)*(ii/float(Lambda))

# 2.42 transição exponencial de V_dB
V_dB = 31.
T2_i = T_i*((10*(V_dB/20.))**((ii/float(Lambda))**alpha))


############ 2.2.3 Aplicação de filtros digitais
# VEJA iir.py para a geraçào da figura 2.17
# T_i herdado
# resposta ao impulso sintética (reverb)
H_i = (n.random.random(10)*2-1)*n.e**(-n.arange(10))

# 2.43 Convolução
T2_i = n.convolve(T_i, H_i)

# 2.44 veja linhas seguintes para aplicação da
# equação a diferenças :-)

fc = .1
# 2.45 passa baixas de polo simples
x = n.e**(-2*n.pi*fc)  # fc  = > freq de corte em 3dB
# coeficientes
a0 = 1-x
b1 = x
# aplicação do filtro
T2_i = [T_i[0]]
for t_i in T_i[1:]:
    T2_i.append(t_i*a_0+T2_i[-1]*b1)

# 2.46 passa altas de polo simples
x = n.e**(-2*n.pi*fc)  # fc = > freq de corte em 3dB
a0 = (1+x)/2
a1 = -(1+x)/2
b1 = x

# aplicação do filtro
T2_i = [a0*T_i[0]]
last = T_i[0]
```

```
for t_i in T_i[1:]:
    T2_i += [a0*t_i + a1*last + b1*T2_i[-1]]
    last = n.copy(t_i)


fc = .1
bw = .05
# 2.47 Variáveis auxiliares para os filtros nó
r = 1-3*bw
k = (1-2*r*n.cos(2*n.pi*fc)+r**2)/(2-2*n.cos(2*n.pi*fc))

# 2.48 passa banda
# coefs passa banda
a0 = 1-k
a1 = -2*(k-r)*n.cos(2*n.pi*fc)
a2 = r**2 - k
b1 = 2*r*n.cos(2*n.pi*fc)
b2 = -r**2

# aplicacao do filtro em T_i resultando em T2_i
T2_i = [a0*T_i[0]]
T2_i += [a0*T_i[1]+a1*T_i[0]+b1*T2_i[-1]]
last1 = T_i[1]
last2 = T_i[0]
for t_i in T_i[2:]:
    T2_i += [a0*t_i+a1*last1+a2*last2+b1*T2_i[-1]+b2*T2_i[-2]]
    last2 = n.copy(last1)
    last1 = n.copy(t_i)

# 2.49 rejeita banda
# coeficientes
a0 = k
a1 = -2*k*n.cos(2*n.pi*fc)
a2 = k
b1 = 2*r*n.cos(2*n.pi*fc)
b2 = -r**2

# aplicacao do filtro em T_i resultando em T2_i
T2_i = [a0*T_i[0]]
T2_i += [a0*T_i[1]+a1*T_i[0]+b1*T2_i[-1]]
last1 = T_i[1]
last2 = T_i[0]
for t_i in T_i[2:]:
    T2_i += [a0*t_i+a1*last1+a2*last2+b1*T2_i[-1]+b2*T2_i[-2]]
    last2 = n.copy(last1)
    last1 = n.copy(t_i)


############ 2.2.4 Ruídos
```

```
185 # VEJA ruidos.py para o script que gerou a figura 2.18
186 Lambda = 100000  # Lambda sempre par
187 # diferença das frequências entre coeficiêntes vizinhos:
188 df = f_a/float(Lambda)
189
190 # 2.50 Ruido branco
191 # geração de espectro com módulo 1 uniforme
192 # e fase aleatória
193 coefs = n.exp(1j*n.random.uniform(0, 2*n.pi, Lambda))
194 # real par, imaginaria impar
195 coefs[Lambda/2+1:] = n.real(coefs[1:Lambda/2])[::-1] - 1j * \
196     n.imag(coefs[1:Lambda/2])[::-1]
197 coefs[0] = 0.  # sem bias
198 coefs[Lambda/2] = 1.  # freq max eh real simplesmente
199
200 # as frequências relativas a cada coeficiente
201 # acima de Lambda/2 nao vale
202 fi = n.arange(coefs.shape[0])*df
203 f0 = 15.  # iniciamos o ruido em 15 Hz
204 i0 = n.floor(f0/df)  # primeiro coef a valer
205 coefs[:i0] = n.zeros(i0)
206 f0 = fi[i0]
207
208 # obtenção do ruído em suas amostras temporais
209 ruido = n.fft.ifft(coefs)
210 r = n.real(ruido)
211 r = ((r-r.min())/(r.max()-r.min()))*2-1
212 a.wavwrite(r, 'branco.wav', f_a)
213
214
215 # 2.51 Ruído rosa
216 # a cada oitava, perdemos 3dB
217 fator = 10.**(-3/20.)
218 alphai = fator**(n.log2(fi[i0:]/f0))
219
220 c = n.copy(coefs)
221 c[i0:] = coefs[i0:]*alphai
222 # real par, imaginaria impar
223 c[Lambda/2+1:] = n.real(c[1:Lambda/2])[::-1] - 1j * \
224     n.imag(c[1:Lambda/2])[::-1]
225
226 ruido = n.fft.ifft(c)
227 r = n.real(ruido)
228 r = ((r-r.min())/(r.max()-r.min()))*2-1
229 a.wavwrite(r, 'rosa.wav', f_a)
230
231
232 # 2.52 Ruído marrom
233 # para cada oitava, perdemos 3dB
```

```
234 fator = 10.**(-6/20.)
235 alphai = fator**(n.log2(fi[i0:]/f0))
236 c = n.copy(coefs)
237 c[i0:] = c[i0:]*alphai
238
239 # real par, imaginaria impar
240 c[Lambda/2+1:] = n.real(c[1:Lambda/2])[::-1] - 1j * \
241     n.imag(c[1:Lambda/2])[::-1]
242
243 # realizando amostras temporais do ruído marrom
244 ruido = n.fft.ifft(c)
245 r = n.real(ruido)
246 r = ((r-r.min())/(r.max()-r.min()))*2-1
247 a.wavwrite(r, 'marrom.wav', f_a)
248
249
250 # 2.53 Ruído azul
251 # para cada oitava, ganhamos 3dB
252 fator = 10.**(3/20.)
253 alphai = fator**(n.log2(fi[i0:]/f0))
254 c = n.copy(coefs)
255 c[i0:] = c[i0:]*alphai
256
257 # real par, imaginaria impar
258 c[Lambda/2+1:] = n.real(c[1:Lambda/2])[::-1] - 1j * \
259     n.imag(c[1:Lambda/2])[::-1]
260
261 # realizando amostras temporais do ruído azul
262 ruido = n.fft.ifft(c)
263 r = n.real(ruido)
264 r = ((r-r.min())/(r.max()-r.min()))*2-1
265 a.wavwrite(r, 'azul.wav', f_a)
266
267
268 # 2.54 Ruido violeta
269 # a cada oitava, ganhamos 6dB
270 fator = 10.**(6/20.)
271 alphai = fator**(n.log2(fi[i0:]/f0))
272 c = n.copy(coefs)
273 c[i0:] = c[i0:]*alphai
274
275 # real par, imaginaria impar
276 c[Lambda/2+1:] = n.real(c[1:Lambda/2])[::-1] - 1j * \
277     n.imag(c[1:Lambda/2])[::-1]
278
279 ruido = n.fft.ifft(c)
280 r = n.real(ruido)
281 r = ((r-r.min())/(r.max()-r.min()))*2-1
282 a.wavwrite(r, 'violeta.wav', f_a)
```

```
283
284 # 2.55 Ruído preto
285 # a cada oitava, perdemos mais que 6dB
286 fator = 10.**(-12/20.)
287 alphai = fator**(n.log2(fi[i0:]/f0))
288 c = n.copy(coefs)
289 c[i0:] = c[i0:]*alphai
290
291 # real par, imaginaria impar
292 c[Lambda/2+1:] = n.real(c[1:Lambda/2])[::-1] - 1j * \
293     n.imag(c[1:Lambda/2])[::-1]
294
295 ruido = n.fft.ifft(c)
296 r = n.real(ruido)
297 r = ((r-r.min())/(r.max()-r.min()))*2-1
298 a.wavwrite(r, 'preto.wav', f_a)
299
300
301 ########## 2.2.5 Tremolo e vibrato, AM e FM
302 # VEJA: vibrato.py e tremolo.py para as figuras 2.19 e 2.20
303 f = 220.
304 Lv = 2048  # tamanho da tabela do vibrato
305 fv = 1.5  # frequência do vibrato
306 nu = 1.6  # desvio maximo em semitons do vibrato (profundidade)
307 Delta = 5.2  # duração do som
308 Lambda = int(Delta*f_a)
309
310 # tabela do vibrato
311 x = n.linspace(0, 2*n.pi, Lv, endpoint=False)
312 tabv = n.sin(x)  # o vibrato será senoidal
313
314 ii = n.arange(Lambda)  # índices
315 # 2.56 índices da LUT para o vibrato
316 Gammav_i = n.array(ii*fv*float(Lv)/f_a, n.int)  # índices para a LUT
317 # 2.57 padrão de oscilação do vibrato para cada amostra
318 Tv_i = tabv[Gammav_i % Lv]
319 # 2.58 frequência em cada amostra
320 F_i = f*(2.**(Tv_i*nu/12.))
321 # 2.59 índices para LUT do som
322 D_gamma_i = F_i*(Lt/float(f_a))  # movimentação na tabela por amostra
323 Gamma_i = n.cumsum(D_gamma_i)  # a movimentação na tabela total
324 Gamma_i = n.array(Gamma_i, dtype=n.int)  # já os índices
325 # 2.60 som em si
326 T_i = Tr_i[Gamma_i % Lt]  # busca dos índices na tabela
327
328 a.wavwrite(T_i, "vibrato.wav", f_a)  # escrita do som
329
330
```

```
Tt_i = n.copy(Tv_i)
# 2.61 Envoltória do tremolo
V_dB = 12.  # decibels envolvidos na variação
A_i = 10**((V_dB/20)*Tt_i)
# 2.62 Aplicação na sequência T_i
Gamma_i = n.array(ii*f*Lt/f_a, dtype=n.int)
T_i = Tr_i[Gamma_i % Lt]
T_i = T_i*A_i
a.wavwrite(T_i, "tremolo.wav", f_a)  # escrita do som


# 2.63 – Espectro da FM, implementada em 2.66–70
# 2.64 – Função de Bessel, foge ao escopo
# 2.65 – Espectro da AM, implementada em 2.70,71 abaixo

fv = 60.  # > 20Hz
# 2.66 índices para a LUT da moduladora da FM
Gammav_i = n.array(ii*fv*float(Lv)/f_a, n.int)
# 2.67 padrão de oscilação da moduladora
Tfm_i = tabv[Gammav_i % Lv]
f = 330.
mu = 40.
# 2.68 Frequência em cada amostra na FM
f_i = f+Tfm_i*mu
# 2.69 índices da LUT para síntese do som
D_gamma_i = f_i*(Lt/float(f_a))  # movimentação na tabela por amostra
Gamma_i = n.cumsum(D_gamma_i)  # a movimentação na tabela total
Gamma_i = n.array(Gamma_i, dtype=n.int)  # já os índices
# 2.70 FM
T_i = S_i[Gamma_i % Lt]  # busca dos índices na tabela

a.wavwrite(T_i, "fm.wav", f_a)  # escrita do som


Tam_i = n.copy(Tfm_i)
V_dB = 12.
alpha = 10**(V_dB/20.)  # profundidade da AM
# 2.71 Envoltória para AM
A_i = 1+alpha*Tam_i
Gamma_i = n.array(ii*f*Lt/f_a, dtype=n.int)
# 2.70 AM
T_i = Tr_i[Gamma_i % Lt]*(A_i)
a.wavwrite(T_i, "am.wav", f_a)  # escrita do som


############ 2.2.5 Usos musicais
# 2.73 Veja peça Tremolos, Vibratos e a Frequência
```

```
# 2.74 ADSR – variação linear
Delta = 5.  # duração total em segundos
Delta_A = 0.1  # Ataque
Delta_D = .3  # Decay
Delta_R = .2  # Release
a_S = .1  # nível de sustentação

Lambda = int(f_a*Delta)
Lambda_A = int(f_a*Delta_A)
Lambda_D = int(f_a*Delta_D)
Lambda_R = int(f_a*Delta_R)

# Realização da envoltória ADSR: A_i
ii = n.arange(Lambda_A, dtype=n.float)
A = ii/(Lambda_A-1)
A_i = A
ii = n.arange(Lambda_A, Lambda_D+Lambda_A, dtype=n.float)
D = 1-(1-a_S)*((ii-Lambda_A)/(Lambda_D-1))
A_i = n.hstack((A_i, D))
S = a_S*n.ones(Lambda-Lambda_R-(Lambda_A+Lambda_D), dtype=n.float)
A_i = n.hstack((A_i, S))
ii = n.arange(Lambda-Lambda_R, Lambda, dtype=n.float)
R = a_S-a_S*((ii-(Lambda-Lambda_R))/(Lambda_R-1))
A_i = n.hstack((A_i, R))

# 2.75 Realização do som com a envoltória
ii = n.arange(Lambda, dtype=n.float)
Gamma_i = n.array(ii*f*Lt/f_a, dtype=n.int)
T_i = Tr_i[Gamma_i % Lt]*(A_i)

a.wavwrite(T_i, "adsr.wav", f_a)  # escrita do som em disco


# 2.74 ADSR – variação Exponencial
xi = 1e-2  # -180dB para iniciar o fade in e finalizar o fade out
De = 2*100.  # duracao total (\Delta)
DA = 2*20.  # duracao do ataque \Delta_A
DD = 2*20.  # duracao do decay \Delta_D
DR = 2*20.  # duracao do release \Delta_R
SS = .4  # fração da amplitude em que ocorre o sustain

A = xi*(1./xi)**(n.arange(DA)/(DA-1))  # amostras do ataque
A_i = n.copy(A)
D = a_S**((n.arange(DA, DA+DD)-DA)/(DD-1))  # amostras do decay
A_i = n.hstack((A_i, D))
S = a_S*n.ones(De-DR-(DA+DD))  # amostras do sustain
A_i = n.hstack((A_i, S))
R = (SS)*(xi/SS)**((n.arange(De-DR, De)+DR-De)/(DR-1))  # release
A_i = n.hstack((A_i,  R))
```

```
427
428 # 2.75 Realização do som com a envoltória
429 ii = n.arange(Lambda, dtype=n.float)
430 Gamma_i = n.array(ii*f*Lt/f_a, dtype=n.int)
431 T_i = Tr_i[Gamma_i % Lt]*(A_i)
432
433 a.wavwrite(T_i, "adsr_exp.wav", f_a)  # escrita do som em disco
```

A.3 Código Python das relações descritas na seção 2.3

Todas as equações da seção 2.3 estão implementadas abaixo em Python e disponibizadas como parte da toolbox MASSA.(2)

equações da seção 2.3 em Python

```
#_8_ coding: utf8 -*-
import numpy as n

f_a = 44100.  # Hz, frequência de amostragem
Lambda_tilde = Lt = 1024.
foo = n.linspace(0, 2*n.pi, Lt, endpoint=False)
S_i = n.sin(foo)  # um período da senóide com T amostras


def v(f=200, d=2., tab=S_i, fv=2., nu=2., tabv=S_i):
    Lambda = n.floor(f_a*d)
    ii = n.arange(Lambda)
    Lv = float(len(S_i))

    Gammav_i = n.floor(ii*fv*Lv/f_a)  # índices para a LUT
    Gammav_i = n.array(Gammav_i, n.int)
    # padrão de variação do vibrato para cada amostra
    Tv_i = tabv[Gammav_i % int(Lv)]

    # frequência em Hz em cada amostra
    F_i = f*(2.**(Tv_i*nu/12.))
    # a movimentação na tabela por amostra
    D_gamma_i = F_i*(Lt/float(f_a))
    Gamma_i = n.cumsum(D_gamma_i)  # a movimentação na tabela total
    Gamma_i = n.floor(Gamma_i)  # já os índices
    Gamma_i = n.array(Gamma_i, dtype=n.int)  # já os índices
    return tab[Gamma_i % int(Lt)]  # busca dos índices na tabela


########## 2.3.1 Afinação, intervalos, escalas e acordes

# 2.76 Intervalos
I1j = 0.
I2m = 1.
I2M = 2.
I3m = 3.
I3M = 4.
I4J = 5.
ITR = 6.
```

```
I5J = 7.
I6m = 8.
I6M = 9.
I7m = 10.
I7M = 11.
I8J = 12.
I_i = n.arange(13.)


# o intervalo soma nove nomenclatuda da inversão
# mas soma sempre 12 na inversão de semitons
def inv(I):
    """retorna intervalo inverso de I: 0< =  I < = 12"""
    return 12-I


# intervalo harmônico
def intervaloHarmonico(f, I):
    return (v(f)+v(f*2.**(I/12.)))*0.5


# intervalo melódico
def intervaloMelodico(f, I):
    return n.hstack((v(f), v(f*2.**(I/12.))))

# 2.77 Escalas simétricas
Ec_i = [0., 1., 2., 3., 4., 5., 6., 7., 8., 9., 10., 11.]
Et_i = [0., 2., 4., 6., 8., 10.]
Etm_i = [0., 3., 6., 9.]
EtM_i = [0., 4., 8.]
Ett_i = [0., 6.]

# 2.78 Escalas diatônicas
Em_i = [0., 2., 3., 5., 7., 8., 10.]
Emlo_i = [1., 3., 5., 6., 8., 10.]
EM_i = [0., 2., 4., 5., 7., 9., 11.]
Emd_i = [0., 2., 3., 5., 7., 9., 10.]
Emf_i = [0., 1., 3., 5., 7., 8., 10.]
Eml_i = [0., 2., 4., 6., 7., 9., 11.]
Emmi_i = [0., 2., 4., 5., 7., 8., 10.]

# 2.79 Padrão diatônico
E_i_ = n.roll(n.array([2.,2.,1.,2.,2.,2.,1.]), n.random.randint(7.))
E_i = n.cumsum(E_i_)-E_i_[0.]


# 2.80 Escalas menores harmônica e melódica
Em_i = [0., 2., 3., 5., 7., 8., 10.]
Emh_i = [0., 2., 3., 5., 7., 8., 11.]
```

```
Emm_i = [0.,2.,3.,5.,7.,9.,11.,12.,10.,8.,7.,5.,3.,2.,0.]


# 2.81 Tríades
AM_i = [0., 4., 7.]
Am_i = [0., 3., 7.]
Ad_i = [0., 3., 6.]
Aa_i = [0., 4., 8.]

def comSetimam(A): return A+[10.]
def comSetimaM(A): return A+[11.]

# Microtonalidade de quartos de tom
# e sétimos de oitava
# com
epslon = 2**(1/12.)
s1 = [0., 0.25, 1.75, 2., 2.25, 4., 5., 5.25]
#ou
epslon = 2**(1/7.)
s2 = [0., 1., 2., 3., 4., 5., 6.]


########## 2.3.2 Rudimentos de harmonia
# Figura 2.22
def relativa(TT):
    """TT é tríade maior ou menor em posicão fechada e fundamental"""
    T = n.copy(TT)
    if T[1]-T[0] == 4.:  # ac maior
        T[2] = 9.  # retorna acorde menor abaixo
    elif T[1]-T[0] == 3.:  # ac menor
        T[0] = 10.  # retorna acorde maior acima
    else:
        print("send me only minor or major perfect triads")
    return T


def antiRelativa(TT):
    T = n.copy(TT)
    if T[1]-T[0] == 4.:  # maior
        T[0] = 11.  # retorna menor acima
    if T[1]-T[0] == 3.:  # menor
        T[2] = 8.  # retorna maior abaixo
    return T


class Mediana:
    def sup(self, TT):
        T = n.copy(TT)
        if T[1]-T[0] == 4.:  # maior
```

```
138             T[0] = 11.
139             T[2] = 8.  # retorna maior
140         if T[1]-T[0] == 3.:  # menor
141             T[0] = 10.
142             T[2] -= 1.  # retorna menor
143         return T
144 
145     def inf(self, TT):
146         T = n.copy(TT)
147         if T[1]-T[0] == 4.:  # maior
148             T[2] = 9
149             T[0] = 1.  # retorna maior
150         if T[1]-T[0] == 3.:  # menor
151             T[2] = 8.
152             T[0] = 11.  # retorna menor
153         return T
154 
155     def supD(self, TT):
156         """Preserva a quinta da primeira tríade na terça """
157         T = n.copy(TT)
158         if T[1]-T[0] == 4.:  # maior
159             T[0] = 10.
160             T[1] = 3.  # retorna maior
161         if T[1]-T[0] == 3.:  # menor
162             T[0] = 11.
163             T[1] = 4.  # retorna menor
164         return T
165 
166     def infD(self, TT):
167         T = n.copy(TT)
168         if T[1]-T[0] == 4.:  # maior
169             T[1] = 3.
170             T[2] = 8.  # retorna maior
171         if T[1]-T[0] == 3.:  # menor
172             T[1] = 4.
173             T[2] = 9.  # retorna menor
174         return T
175 
176 # Tônicas e funções principais
177 tonicaM = [0., 4., 7.]
178 tonicam = [0., 3., 7.]
179 subM = [0., 5., 9.]
180 subm = [0., 5., 8.]
181 dom = [2., 7., 11.]
182 Vm = [2., 7., 10.]  # quinto grau menor nao eh dominante
183 
184 
185 ########## 2.3.3 Contraponto
186 def contraNotaNotaSup(alturas=[0,2,4,5,5,0,2,0,2,2,2,0,7,\
187                                5,4,4,4,0,2,4,5,5,5]):
```

```
188     """Realiza rotina de independência das vozes
189
190     Limitado em 1 oitava acima da nota"""
191     primeiraNota=alturas[0]+(7,12)[n.random.randint(2)]
192     contra=[primeiraNota]
193
194     i=0
195     cont=0 # contador de paralelas
196     reg=0 # registrador de intervalo em que se fez a paralela
197     for al in alturas[:-1]:
198         mov_cf=alturas[i:i+2]
199         atual_cf,seguinte_cf=mov_cf
200         if seguinte_cf-atual_cf>0:
201             mov="asc"
202         elif seguinte_cf-atual_cf<0:
203             mov="asc"
204         else:
205             mov="obl"
206
207         # possibilidades por consonancia
208         possiveis=[seguinte_cf+interval for interval in\
209                                     [0,3,4,5,7,8,9,12]]
210         movs=[]
211         for pos in possiveis:
212             if pos -contra[i] < 0:
213                 movs.append("desc")
214             if pos - contra[i] > 0:
215                 movs.append("asc")
216             else:
217                 movs.append("obl")
218
219         movt=[]
220         for m in movs:
221             if 'obl' in (m,mov):
222                 movt.append("obl")
223             elif m==mov:
224                 movt.append("direto")
225             else:
226                 movt.append("contrario")
227         blacklist=[]
228         for nota,mt in zip(possiveis,movt):
229
230             if mt == "direto": # mov direto
231                 # n aceita intervalo perfeito
232                 if nota-seguinte_cf in (0,7,8,12):
233                     possiveis.remove(nota)
234         ok=0
235         while not ok:
236             nnota=possiveis[n.random.randint(len(possiveis))]
237             if nnota-seguinte_cf==contra[i]-atual_cf: # paralelo
```

```
238                 intervalo=contra[i]-atual_cf
239                 novo_intervalo=nnota-seguinte_cf
240                 if abs(intervalo-novo_intervalo)==1: # do mesmo tipo 3 ou 6
241                     if cont==2: # se já teve 2 paralelas
242                         pass # outro intrevalo
243                     else:
244                         cont+=1
245                         ok=1
246             else: # mov obl ou contrario
247                 cont=0 # zera paralelos
248                 ok=1
249         contra.append(nnota)
250         i+=1
251     return contra
252
253
254 ############ 2.3.4 Ritmo
255 # Veja peça Poli Hit Mia no Apêndice B
256
257
258 ############ 2.3.5 Estruturas direcionais
259 # Veja peça Dirracional no Apêndice B
260
261
262 ############ 2.3.6 Estruturas cíclicas
263 # Veja as peças 3 Trios no Apêndice B
264 # e o PPEPPS no Apêndice C
```

APÊNDICE B – Código Computacional das Peças Musicais

Todas as peças a seguir foram feitas para exemplificar as relações apresentadas no capítulo 2 e são disponibilizadas online junto ao *toolbox* MASSA.(2)

B.1 Peças referentes à seção 2.1

B.1.1 Quadros sonoros

A montagem musical ‘quadros sonoros’ é dedicada para demonstração da mixagem pela soma direta de sequências amostrais (subseção 2.1.8). Esta rotina sintetiza 5 pequenas peças de sonoridades estáticas. Peça demonstrativa dos conceitos apresentados na seção 2.1. Houve relatos em listas de *emails* de que estas sonoridades causaram alteração do estado de consciência. Em especial, o quadro 5 causou em diferentes pessoas o mesmo efeito: a sensação de amplificação e distorção dos sons da mandimbula.

Quadros sonoros 1-5

```
#-*- coding: utf8 -*-
import numpy as n
import scikits.audiolab as a

# Peça musical baseada em sons estáticos
# são mixagens de sons básicos apresentados na seção 2.1

f_a = 44100  # 44.1kHz, frequência de amostragem de CDs
Delta = 180.  # cada quadro terá Delta segundos
Lambda = int(Delta*f_a)  # número de amostras

ii = n.linspace(0, Delta*2*n.pi, Lambda, endpoint=False)

# frequências que dividem f_a
fs = []
for i in xrange(1, f_a/2+1):
    if f_a/float(i) == int(f_a/i):
        fs.append(i)

# Quadro 1: senoides no grave em batimento e
# no agudo uma dente de serra bem suave
f1 = 100
f2 = 100.5

som = n.sin(ii*f1)+n.sin(ii*f2)
dente_aguda = (n.arange(Lambda) % 4-2)
som += dente_aguda/80

# normalição no intervalo [-1,1]
```

```
som = ((som - som.min())/(som.max()-som.min()))*2-1

a.wavwrite(som, "quadro1.wav", f_a)
print(u"quadro 1 feito em quadro1.wav (mono), são %i amostras \
em uma frequência de amostragem de %iHz" % (len(som), f_a))


# Quadro 2: 3 conjuntos separados de triangulares
fs2 = fs[20:21]+fs[65:70]+fs[77:]
som = n.zeros(Lambda)
ii = n.arange(Lambda)
for f in fs2:
    lambda_f = f_a/f
    som += (1-n.abs(2-(4./lambda_f)*(ii % lambda_f)))*(1./f**1.2)

# normalizando no intervalo [-1,1]
som = ((som - som.min())/(som.max()-som.min()))*2-1

a.wavwrite(som, "quadro2.wav", f_a)
print("quadro 2 feito em quadro2.wav (mono), são %i amostras \
em uma frequência de amostragem de %iHz" % (len(som), f_a))


# Quadro 3: estereofonia alternada no espectro harmônico
f = 50.
fs3 = [f*i for i in xrange(1, 7)]  # 6 harmônicos
som_d = n.zeros(Lambda)
som_e = n.zeros(Lambda)
ii = n.linspace(0, Delta*2*n.pi, Lambda, endpoint=False)
i = 0
for f in fs3:
    if i % 2 == 0:
        som_d += n.sin(f*ii)*(1./f)
    else:
        som_e += n.sin(f*ii)*(1./f)
    i += 1
som = n.vstack((som_d, som_e)).T
som = ((som - som.min())/(som.max()-som.min()))*2-1

a.wavwrite(som, "quadro3.wav", f_a)
print("quadro 3 feito em quadro3.wav (estéreo), são %i amostras \
em uma frequência de amostragem de %iHz" % (len(som), f_a))


# Quadro 4: batimentos intercalados por ouvido e com defasagens
fs4 = n.array([50, 51.01, 52.01, 53])
som_d = n.zeros(Lambda)
som_e = n.zeros(Lambda)
ii = n.linspace(0, Delta*2*n.pi, Lambda, endpoint=False)
```

```
i = 0
for f in fs4:
    if i % 2 == 0:
        som_d += n.sin(f*ii)*(1./f)
    else:
        som_e += n.sin(f*ii)*(1./f)
    i += 1
som = n.vstack((som_d, som_e)).T

fs4 = n.array([500, 501.01, 502.01, 503])
som_d = n.zeros(Lambda)
som_e = n.zeros(Lambda)
ii = n.linspace(0, Delta*2*n.pi, Lambda, endpoint=False)
i = 0
for f in fs4:
    if i % 2 == 0:
        som_d += n.sin(f*ii)*(1./f)
    else:
        som_e += n.sin(f*ii)*(1./f)
    i += 1
som += n.vstack((som_d, som_e)).T/60
som = ((som - som.min())/(som.max()-som.min()))*2-1

a.wavwrite(som, "quadro4.wav", f_a)
print("quadro 4 feito em quadro4.wav (estéreo), são %i amostras \
em uma frequência de amostragem de %iHz" % (len(som), f_a))


# Quadro 5: Dente de serra grave bate com harmônico em cada lado
f = 42.  # Hz, freq da dente
lambda_f = 44100/f
dente = ((n.arange(float(Lambda)) % lambda_f)/lambda_f)*2-1

ii = n.linspace(0, Delta*2*n.pi, Lambda, endpoint=False)
som_d = dente+n.sin(ii*43)+n.sin(ii*84)
som_e = dente+n.sin(ii*43) + n.sin(ii*126.3)

som = n.vstack((som_d, som_e)).T
som = ((som - som.min())/(som.max()-som.min()))*2-1

a.wavwrite(som, "quadro5.wav", f_a)
print("quadro 5 feito em quadro5.wav (estéreo), são %i amostras \
em uma frequência de amostragem de %iHz" % (len(som), f_a))
```

B.1.2 Reduced-fi

Pequena peça musical para a concatenação de sequências amostrais como notas musicais (subseção 2.1.8). Sintetiza uma pequena peça de 25 segundos em Python puro, i.e. sem utilizar bibliotecas externas como *Numpy* ou *Scikits/Audiolab*. Peça demonstrativa dos conceitos apresentados na seção 2.1 e disponibilizada online junto ao *toolbox* MASSA.(2)

reduced-fi

```
#! /usr/bin/python

# Rode com:
# ./reduced-fi.py
# ou
# python reduced-fi.py

import math as m
import random as r

def p(freq):
    return 2*m.pi*freq/44100

def d(dur):
    return int(dur*44100)

def s(freq,dur):
    return [m.sin(p(freq)*i) for i in xrange(d(dur))]

ss=(s(100,.1)+s(200,.1)+s(400,.1)+s(800,.1))*4
ss+=(s(100,.1)+s(200,.1)+s(400,.1)+s(800,.1))*4
ss+=(s(200,.05)+s(100,.05)+s(3200,.05)+s(800,.05))*8
ss+=(s(200,.05)+s(100,.05)+s(3200,.05)+s(800,.05))*8

ss+=(s(100,.01)+s(200,.01)+s(400,.01)+s(800,.01))*4
ss+=(s(100,.01)+s(200,.01)+s(400,.01)+s(800,.01))*4
ss+=(s(200,.005)+s(100,.005)+s(3200,.005)+s(800,.005))*8
ss+=(s(200,.005)+s(100,.005)+s(3200,.005)+s(800,.005))*8
ss+=s(300,2)

ss+=(s(100,.01)+s(200,.01)+s(400,.01)+s(800,.01))*4
ss+=(s(100,.01)+s(200,.01)+s(4000,.01)+s(800,.01))*4
ss+=(s(200,.005)+s(100,.005)+s(6400,.005)+s(800,.005))*8
ss+=(s(200,.005)+s(100,.005)+s(3200,.005)+s(800,.005))*8
ss+=s(300,2)

```

```
foo=ss[:int(44100*1)]; r.shuffle(foo); ss+=[i*.1 for i in foo]
ss+=s(300,2)


foo=ss[:int(44100*.5)]; r.shuffle(foo); ss+=[i*.1 for i in foo]
ss+=(s(100,.01)+s(200,.01)+s(400,.01)+s(800,.01))*4
ss+=(s(100,.01)+s(200,.01)+s(400,.01)+s(800,.01))*4
ss+=(s(200,.005)+s(100,.005)+s(3200,.005)+s(800,.005))*8
ss+=(s(200,.005)+s(100,.005)+s(3200,.005)+s(800,.005))*8

ss+=(s(100,.01)+s(200,.01)+s(400,.01)+s(800,.01))*4
ss+=(s(100,.01)+s(200,.01)+s(4000,.01)+s(800,.01))*4
ss+=(s(200,.005)+s(100,.005)+s(6400,.005)+s(800,.005))*8
ss+=(s(2000,.005)+s(100,.005)+s(10000,.005)+s(800,.005))*8
ss+=s(300,2)

ss+=(s(50,.1)+s(200,.1)+s(400,.1)+s(800,1))*1
ss+=(s(50,1)+s(200,.1)+s(400,.1)+s(800,.1))*1
ss+=(s(100,.5)+s(100,.05)+s(3200,.05)+s(800,.05))*1
ss+=(s(200,.05)+s(100,.05)+s(3200,.05)+s(400,.5))*1
foo=ss[:int(44100*.25)]; r.shuffle(foo); ss+=[i*.1 for i in foo]

ss+=s(400,.2)
ss+=s(200,.8)
ss+=s(150,.5)
ss+=s(100,1.5)


##########################
seq=ss
## Gravando em disco
import wave, struct
smin=min(seq); smax=max(seq)
seq= [(-.5+(i-smin)/(smax-smin)) for i in seq] # normalizando
sound = wave.open('musica.wav','w')
sound.setframerate(44100)
sound.setsampwidth(2) # sempre 16bit/sample (2 bytes)
sound.setnchannels(1) # mono
sonic_vector=[i*(2**15-1) for i in seq] # 16 bit com sinal
sound.writeframes(struct.pack('h'*len(sonic_vector),\
                      *[int(i) for i in sonic_vector]))
sound.close()
print "arquivo musica.wav gravado"
```

B.2 Peças referentes à seção 2.2

B.2.1 Transita para metro

Sintetiza em 1 pequena peça de 49 segundos com varreduras de frequência, chamados *chirps*. Também é proposta para explorações de transições de intensidade. Peça demonstrativa dos conceitos apresentados na subseção 2.2.2 e disponibilizada junto à MASSA.(2)

Transita para metro

```
#-*- coding: utf8 -*-
import numpy as n, scikits.audiolab as aa

# peça dedicada e expor as diferentes transições
# de intensidade e altura

# partes
# 1) Transições de altura log e lin e com alpha
# 2) Transições de intensidade log e in e com alpha
# 3) Usos combinados de ambos

f_a = 44100. # Hz, frequência de amostragem
Lambda_tilde=Lt=1024 # tamanho da tabela para LUT

# Senoide
foo=n.linspace(0,2*n.pi,Lt,endpoint=False)
S_i=n.sin(foo) # um período da senóide com T amostras

# Quadrada:
Q_i=n.hstack(  ( n.ones(Lt/2)*-1 , n.ones(Lt/2) )  )

# Siangular:
foo=n.linspace(-1,1,Lt/2,endpoint=False)
Tr_i=n.hstack(  ( foo , foo*-1 )   )

# Dente de Serra:
D_i=n.linspace(-1,1,Lt)

def T(f1,f2,dur,ttype="exp",tab=S_i,alpha=1.):
    Lambda=n.floor(dur*f_a)
    ii=n.arange(Lambda)

    if ttype=="exp":
        f_i=f1*(f2/f1)**(  (ii/(float(Lambda)-1))**alpha  ) # exponencial
    else:
```

```
36          f_i=f1+(f2-f1)*ii/(Lambda-1) # linear
37
38      Lt=len(tab)
39      D_gamma_i=f_i*Lt/f_a
40      Gamma_i=n.cumsum(D_gamma_i)
41      Gamma_i=n.array(Gamma_i,dtype=n.int)
42      return tab[Gamma_i%Lt]
43
44  ##########################################
45  # PARTE 1)
46
47  # 1a: comparando variação linear e logarítmica
48  #class
49  """parametros pré-estabelecidos para maior coerência da exploração"""
50
51  f=[50.,100.,150.,200.,250.,300.,350.,400.,450.,500.,550.,  600.]
52  #  01  11    25  31    43  55    67   71   82   83   94#   105
53  d=[0.1,0.2,0.3,0.5,1.0,1.5,2.0,3.0,5.0,7.0,10.0]
54  #  0    1   2   3   4   5   6   7   8  9    10
55  a=[0.01,0.1,1.0,10.,100.]
56  #   0    1   2   3    4
57
58
59  #intro (8 segundos)
60  intr=n.hstack((T(f[9],f[0],d[7]),T(f[0],f[10],d[8],'lin',Tr_i)))
61
62  #entrada (8 segundos)
63  entr=  n.hstack(( T(f[10],f[10],d[8],'lin',Tr_i),n.zeros(f_a*d[7])))+ \
64             n.hstack(( T(f[3],f[0],d[3]),
65       T(f[1],f[0],d[3]),T(f[1],f[0],d[3]),T(f[1],f[0],d[3]),
66       T(f[2],f[0],d[3]),
67       T(f[1],f[0],d[3]),T(f[1],f[0],d[3]),T(f[1],f[0],d[3]),
68       T(f[3],f[0],d[3]),
69       T(f[1],f[0],d[3]),T(f[1],f[0],d[3]),T(f[1],f[0],d[3]),
70       T(f[5],f[0],d[3]),
71       T(f[3],f[0],d[3]),T(f[2],f[0],d[3]),T(f[1],f[0],d[3],'lin') ))
72
73  #desenvolvimento da entrada (8 segundos)
74  devEntr=n.hstack(( T(f[0],f[0],d[8]),T(f[0],f[0],d[4]),
75       T(f[0],f[0],d[3],'lin',Tr_i),
76       T(f[0],f[0],d[3],'lin',Tr_i),T(f[0],f[0],d[3],'lin',Q_i),\
77       T(f[0],f[0],d[3],'lin',Tr_i) )) + n.hstack(( T(f[5],f[2],d[3]),
78       T(f[5],f[8],d[3]),T(f[5],f[2],d[3]),T(f[8],f[5],d[3]),
79       T(f[5],f[2],d[3]),
80       T(f[5],f[1],d[3]),T(f[5],f[0],d[3]),T(f[1],f[2],d[3]),
81       T(f[5],f[2],d[3]),
82       T(f[10],f[2],d[3]),T(f[10],f[0],d[3]),T(f[10],f[9],d[3]),
83       T(f[5],f[2],d[3]),
84       T(f[7],f[5],d[3]),T(f[7],f[2],d[3]),T(f[0],f[2],d[3]) ))
85
```

```
#sublimada 1 / passagem para outros funcionamentos e estereofonia 8s
sub_e=n.hstack(( T(f[1],f[0],d[3],'log',Q_i),
     T(f[1],f[0],d[3]),T(f[1],f[0],d[3],'lin'),T(f[1],f[0],d[3]),
     T(f[1],f[0],d[3]),
     T(f[1],f[0],d[3]),T(f[1],f[0],d[3],'lin'),T(f[1],f[0],d[3]),
      T(f[1],f[0],d[3],'log',Q_i),
     T(f[1],f[0],d[3]),T(f[1],f[0],d[3],'lin'),T(f[1],f[0],d[3]),
      T(f[1],f[0],d[3]),
     T(f[1],f[0],d[3]),T(f[1],f[0],d[3],'lin'),T(f[1],f[0],d[3]) )) + \
                 n.hstack(( T(f[0],f[10],d[9])+\
                 T(f[0],f[10],d[9],'lin',Tr_i),n.zeros(f_a*d[4]) ))

sub_d=n.hstack(( T(f[1],f[0],d[3],'log',Q_i),
     T(f[1],f[0],d[3]),T(f[1],f[0],d[3],'lin'),T(f[1],f[0],d[3]),
     T(f[1],f[0],d[3]),
     T(f[1],f[0],d[3]),T(f[1],f[0],d[3],'lin'),T(f[1],f[0],d[3]),
      T(f[1],f[0],d[3],'log',Q_i),
     T(f[1],f[0],d[3]),T(f[1],f[0],d[3],'lin'),T(f[1],f[0],d[3]),
      T(f[1],f[0],d[3]),
     T(f[1],f[0],d[3]),T(f[1],f[0],d[3],'lin'),T(f[1],f[0],d[3]) )) + \
      n.hstack(( T(f[0],f[10],d[9],'log',Tr_i)+\
      T(f[0],f[10],d[9],'lin'),n.zeros(f_a*d[4]) ))

# dev da sublimada +8s
sub2_e=n.hstack(( T(f[10],f[9],d[3]),
     T(f[10],f[7],d[3]),T(f[10],f[9],d[3]),T(f[10],f[6],d[3]),
     T(f[10],f[9],d[3]),
     T(f[10],f[7],d[3]),T(f[10],f[9],d[3]),T(f[10],f[6],d[3]), )) + \
n.hstack(( T(f[10],f[8],d[3],tab=Tr_i),
     T(f[10],f[5],d[3],tab=Tr_i),T(f[10],f[4],d[3],tab=Tr_i),
         T(f[10],f[6],d[3],tab=Tr_i),
     T(f[10],f[10],d[3],tab=Tr_i),
     T(f[10],f[5],d[3],tab=Tr_i),T(f[10],f[0],d[3],tab=Tr_i),
         T(f[10],f[1],d[3],tab=Tr_i)  ))


sub2_d=n.hstack(( T(f[10],f[9],d[3],tab=Tr_i),
     T(f[10],f[7],d[3],tab=Tr_i),T(f[10],f[9],d[3],tab=Tr_i),
        T(f[10],f[6],d[3],tab=Tr_i),
     T(f[10],f[9],d[3],tab=Tr_i),
     T(f[10],f[7],d[3],tab=Tr_i),T(f[10],f[9],d[3],tab=Tr_i),
     T(f[10],f[6],d[3],tab=Tr_i) )) + n.hstack(( T(f[10],f[8],d[3]),
     T(f[10],f[5],d[3]),T(f[10],f[4],d[3]),T(f[10],f[6],d[3]),
     T(f[10],f[10],d[3]),
     T(f[10],f[5],d[3]),T(f[10],f[0],d[3]),T(f[10],f[1],d[3]) ))

# rarefação e repetições preparando para término
rar_e=n.hstack(( T(f[7],f[9],d[3]),
     n.zeros(d[3]), T(f[10],f[9],d[3]),T(f[0],f[9],d[3]),
     n.zeros(d[3]),T(f[0],f[9],d[5]),
```

```
136      n.zeros(d[6]),
137      T(f[1],f[9],d[6],'lin')    ))
138
139 rar_d=n.hstack(( T(f[7],f[9],d[3],'lin'),
140      n.zeros(d[3]), n.zeros(d[3]),T(f[1],f[9],d[3]),
141      T(f[0],f[10],d[4],'lin',Tr_i),
142      T(f[9],f[9],d[4]),
143      T(f[1],f[9],d[6],'log')    ))
144
145
146 # finalização com elementos dos outros momentos
147 intr=n.hstack((T(f[9],f[0],d[7]),T(f[0],f[10],d[8],'lin',Tr_i)))
148
149 fin_e=n.copy(intr)
150 fin_d=n.copy(intr)
151 fin_e[d[4]*f_a:d[5]*f_a]=n.zeros(d[3]*f_a)
152 fin_d[d[5]*f_a:d[6]*f_a]=n.zeros(d[3]*f_a)
153
154 fin_e[7*f_a:7.5*f_a]=n.zeros(d[3]*f_a)
155 fin_d[7.25*f_a:7.75*f_a]=n.zeros(d[3]*f_a)
156
157 _e=n.hstack((   intr,entr,devEntr,sub_e,sub2_e,rar_e,fin_e   ))
158 _d=n.hstack((   intr,entr,devEntr,sub_d,sub2_d,rar_d,fin_d,n.zeros(2)   ))
159 s=n.vstack((_e,_d)).T
160
161 s=((s-s.min())/(s.max()-s.min()))*2-1
162
163 aa.wavwrite(s,'trans1.wav',f_a)
164
165
166 # Exploracao sistemática dos tremolos:
167 #first=n.array(())
168 #for d in d[:5]:
169 #  prinT(d)
170 #  for f in f:
171 #    for f2 in f:
172 #      first=n.hstack((first,
173 #        trans(f,f2,d),
174 #        trans(f,f2,d,'lin'),
175 #        trans(f,f2,d,tab=Tr_i),
176 #        trans(f,f2,d,'lin',tab=Tr_i),
177 #        trans(f,f2,d,tab=D_i),
178 #        trans(f,f2,d,'lin',tab=D_i),
179 #        (trans(f,f2,d)+trans(f,f2,d,'lin'))*.5,
180 #        (trans(f,f2,d,tab=Tr_i)+trans(f,f2,d,'lin',tab=Tr_i))*.5,
181 #        (trans(f,f2,d,tab=Q_i)+trans(f,f2,d,'lin',tab=Q_i))*0.5  ))
```

B.2.2 Vibra e treme

Para explorações de tremolos e vibratos como expostos na subseção 2.2.5, sintetiza 17 pequenas montagens de 8 a 24 segundos cada. Script disponibilizado online junto ao toolbox MASSA.(2)

B.2.3 Tremolos, vibratos e a frequência

Pequena montagem musical para demonstração de vínculo entre os parâmetros dos tremolos e vibratos com a frequência fundamental central da nota. Resulta em 19 pequenas montagens de 4-32 segundos. Peça demonstrativa dos conceitos apresentados na subseção 2.2.6 e disponibilizada online junto ao toolbox MASSA.(2)

B.2.4 Trenzinho de caipiras impulsivos

Pequena peça musical para demonstração do deslocamento causado pela convolução com o impulso. Sintetiza 11 pequenas montages cada uma com duração de 4-240 segundos. Peça demonstrativa dos conceitos apresentados na subseção 2.2.3 e disponibilizada online junto ao toolbox MASSA.

B.2.5 Ruidosa faixa

Pequena peça musical de 240 segundo spara demonstração de filtragens diversas em ruídos e para reverberação, conceitos apresentados na subseção 2.2.3 e disponibilizada online junto ao toolbox MASSA.(2)

ruidosa faixa

```
#-*- coding: utf8 -*-
import numpy as n, scikits.audiolab as a
H=n.hstack
V=n.vstack

```

```
f_a = 44100. # Hz, frequência de amostragem

########## 2.2.1 Tabela de busca (LUT)
Lambda_tilde=Lt=1024.

# Senoide
foo=n.linspace(0,2*n.pi,Lt,endpoint=False)
S_i=n.sin(foo) # um período da senóide com T amostras

# Quadrada:
Q_i=n.hstack(  ( n.ones(Lt/2)*-1 , n.ones(Lt/2) )  )

# Triangular:
foo=n.linspace(-1,1,Lt/2,endpoint=False)
Tr_i=n.hstack(  ( foo , foo*-1 )   )

# Dente de Serra:
D_i=n.linspace(-1,1,Lt)

def v(f=200,d=2.,tab=S_i,fv=2.,nu=2.,tabv=S_i):
    Lambda=n.floor(f_a*d)
    ii=n.arange(Lambda)
    Lv=float(len(S_i))

    Gammav_i=n.floor(ii*fv*Lv/f_a) # índices para a LUT
    Gammav_i=n.array(Gammav_i,n.int)
    # padrão de variação do vibrato para cada amostra
    Tv_i=tabv[Gammav_i%int(Lv)]

    # frequência em Hz em cada amostra
    F_i=f*(   2.**(  Tv_i*nu/12.  )   )
    # a movimentação na tabela por amostra
    D_gamma_i=F_i*(Lt/float(f_a))
    Gamma_i=n.cumsum(D_gamma_i) # a movimentação na tabela total
    Gamma_i=n.floor( Gamma_i) # já os índices
    Gamma_i=n.array( Gamma_i, dtype=n.int) # já os índices
    return tab[Gamma_i%int(Lt)] # busca dos índices na tabela

def A(fa=2.,V_dB=10.,d=2.,taba=S_i):
    Lambda=n.floor(f_a*d)
    ii=n.arange(Lambda)
    Lt=float(len(taba))
    Gammaa_i=n.floor(ii*fa*Lt/f_a) # índices para a LUT
    Gammaa_i=n.array(Gammaa_i,n.int)
    # variação da amplitude em cada amostra
    A_i=taba[Gammaa_i%int(Lt)]
    A_i=A_i*10.**(V_dB/20.)
    return A_i
```

```
56  def adsr(som,A=10.,D=20.,S=-20.,R=100.,xi=1e-2):
57      a_S=10**(S/20.)
58      Lambda=len(som)
59      Lambda_A=int(A*f_a*0.001)
60      Lambda_D=int(D*f_a*0.001)
61      Lambda_R=int(R*f_a*0.001)
62
63      ii=n.arange(Lambda_A,dtype=n.float)
64      A=ii/(Lambda_A-1)
65      A_i=A
66      ii=n.arange(Lambda_A,Lambda_D+Lambda_A,dtype=n.float)
67      D=1-(1-a_S)*(    ( ii-Lambda_A )/( Lambda_D-1) )
68      A_i=n.hstack(  (A_i, D  )   )
69      S=n.ones(Lambda-Lambda_R-(Lambda_A+Lambda_D),dtype=n.float)*a_S
70      A_i=n.hstack( ( A_i, S )  )
71      ii=n.arange(Lambda-Lambda_R,Lambda,dtype=n.float)
72      R=a_S-a_S*((ii-(Lambda-Lambda_R))/(Lambda_R-1))
73      A_i=n.hstack(  (A_i,R)  )
74
75      return som*A_i
76
77
78  BPM=60.  # 80 batidas por minuto
79  DELTA=BPM/60  # duração da batida
80  LAMBDA=DELTA*f_a  # número de samples da batida
81  LAMBDA_=int(LAMBDA)  # inteiro para operação com índices
82
83  #cabeca=[1]+[0]*(LAMBDA-1)
84  #contra=[0]*Lambda/2+[1]+[0]*(Lambda/2-1)
85  tempo=n.zeros(LAMBDA)
86  cabeca=n.copy(tempo); cabeca[0]=1.
87  contra=n.copy(tempo); contra[LAMBDA_/2]=1.
88
89
90  # tempo de musica
91  Delta=4*DELTA  # segundos
92  Lambda=Delta*f_a
93  Lambda_=int(Lambda)
94  ii=n.arange(Lambda_)
95  linha_cabeca=cabeca[ii%LAMBDA_]
96  linha_contra=contra[ii%LAMBDA_]
97
98  som1=adsr(v(tabv=Tr_i ,d=.3,fv=3.,nu=7.0,f=300.),10,10,-10.)
99  som2=adsr(v(tabv=Tr_i ,d=.2,fv=2.,nu=1.),10,10,-10.)
100 som3=adsr(v(tabv=Tr_i ,d=.2,fv=10.,nu=7.),10,10,-10.)
101 som4=adsr(v(tabv=Tr_i ,d=.2,fv=3.,nu=7.,f=1800.),1.,100.,-60.,80.)
102 som5=adsr(v(tabv=Tr_i ,d=.2,fv=3.,nu=7.,f=1800.)*A(d=.2,fa=100.),
103                                                   1.,100.,-60.,80.)
104 som6=adsr(v(tabv=Tr_i ,d=.2,fv=30.,nu=7.,f=1800.)*A(d=.2),
105                                             1.,100.,-60.,80.)
```

```
106
107 em3=n.copy(tempo);em3[[0,LAMBDA_/3,2*LAMBDA_/3]]=1.
108
109 linha_em3=em3[ii%LAMBDA_]
110
111 ############
112 #RUIDOS
113
114 Lambda = 100000  # Lambda sempre par
115 # diferença das frequências entre coeficiêntes vizinhos:
116 df=f_a/float(Lambda)
117
118 # e fase aleatoria
119 coefs=n.exp(1j*n.random.uniform(0, 2*n.pi, Lambda))
120 # real par, imaginaria impar
121 coefs[Lambda/2+1:]=n.real(coefs[1:Lambda/2])[::-1] \
122                    - 1j*n.imag(coefs[1:Lambda/2])[::-1]
123 coefs[0]=0.  # sem bias
124 coefs[Lambda/2]=1.  # freq max eh real simplesmente
125
126 # as frequências relativas a cada coeficiente
127 # acima de Lambda/2 nao vale
128 fi=n.arange(coefs.shape[0])*df
129 f0=15.  # iniciamos o ruido em 15 Hz
130 i0=n.floor(f0/df)  # primeiro coeff a valer
131 coefs[:i0]=n.zeros(i0)
132 f0=fi[i0]
133
134 # realizando o ruído em suas amostras temporais
135 ruido=n.fft.ifft(coefs)
136 r=n.real(ruido)
137 rb=((r-r.min())/(r.max()-r.min()))*2-1  # ruido branco
138
139 # fazendo ruido preto
140 fator=10.**(-7/20.)
141 alphai=fator**(n.log2(fi[i0:]/f0))
142 c=n.copy(coefs)
143 c[i0:]=c[i0:]*alphai
144
145 # real par, imaginaria impar
146 c[Lambda/2+1:]=n.real(c[1:Lambda/2])[::-1] -\
147                1j*n.imag(c[1:Lambda/2])[::-1]
148
149 ruido=n.fft.ifft(c)
150 r=n.real(ruido)
151 rp=((r-r.min())/(r.max()-r.min()))*2-1
152
153 LR=rb[n.arange(int(len(linha_em3)*2.5))%len(rb)]*\
154    A(d=int(len(linha_em3)*2.5)/f_a,fa=.2,V_dB=50.)*10**(-60/20.)
155
```

```
156 LR2=rb[n.arange(int(len(linha_em3)*4.5))%len(rb)]*\
157     A(d=int(len(linha_em3)*4.5)/f_a)*.05
158
159 LR3=6.*rp[n.arange(int(len(linha_em3)*6.5))%len(rp)]*\
160     A(d=int(len(linha_em3)*6.5)/f_a)*.05
161
162
163 obj1=rb[:int(.4*f_a)]*A(d=.4,fa=15.)
164 obj2=rp[:int(.4*f_a)]*A(d=.4,fa=10.)
165
166 obj3=adsr(rb[:int(.4*f_a)]*A(d=.4,fa=15.))
167 obj4=adsr(rp[:int(.4*f_a)]*A(d=.4,fa=10.),S=-5)
168
169 obj5=adsr(rp[:int(1.4*f_a)]*A(d=1.4,fa=10.),5.,500.,-20,200)
170
171 ##########
172
173 l1=n.convolve(obj1,linha_em3)[:len(linha_em3)]
174 l2=n.convolve(obj2,linha_em3)[:len(linha_em3)]
175 l3=n.convolve(obj3,linha_em3)[:len(linha_em3)]
176 l4=n.convolve(obj4,linha_em3)[:len(linha_em3)]
177 l6=n.convolve(obj5,linha_em3)[:len(linha_em3)]
178
179 l1_=n.convolve(obj1,linha_cabeca)[:len(linha_em3)]
180 l2_=n.convolve(obj2,linha_contra)[:len(linha_em3)]
181 l3_=n.convolve(obj3,linha_cabeca)[:len(linha_em3)]
182 l4_=n.convolve(obj4,linha_contra)[:len(linha_em3)]
183 l6_=n.convolve(obj5,linha_cabeca)[:len(linha_em3)]
184
185
186 linha1=n.convolve(som2,linha_cabeca)[:len(linha_cabeca)]
187 linha2=n.convolve(som4,linha_em3)[:len(linha_em3)]
188 linha4=n.convolve(som5,linha_em3)[:len(linha_em3)]
189 linha6=n.convolve(som6,linha_em3)[:len(linha_em3)]
190 linha3=n.convolve(som2,linha_contra)[:len(linha_contra)]
191
192
193 H_i=(n.random.random(int(f_a*1.2))*2-1)*n.e**(-n.arange(int(f_a*1.2)))
194 def r(l):
195     return n.convolve(H_i,l)[:len(linha_em3)]
196
197 som_e=n.hstack((r(linha2)+l1,linha3+r(l2),linha1+l3,
198                         r(linha1)+linha2+linha3,r(l6)))
199 som_e=n.hstack((som_e,r(linha4)+l1_,r(l2_),l3_+linha3+linha1,
200                             r(l4_)+linha1+linha3,r(l6_)+linha2))
201
202 som_d=n.hstack((linha1+r(linha2),linha2+linha3,r(linha3)+linha1,
203                                         linha2+linha3+l4,linha2))
204 som_d=n.hstack((som_d,r(linha4)+l1_,r(l2_),l3_+linha4+linha1,
205                           r(l4_)+linha2+linha3,r(linha6)+linha2))
```

```
206
207 som=n.vstack((som_e,som_d))
208
209 som[:,:len(LR)]+=LR
210
211 som[:,len(l1)*3:len(l1)*3+len(LR2)]+=LR2
212 som[:,int(len(l1)*3.5):int(len(l1)*3.5)+len(LR3)]+=LR3
213 som[:,len(l1)*7:len(l1)*7+len(LR)]+=LR
214
215 T_i=som
216
217 T_i=(T_i-T_i.min())/(T_i.max()-T_i.min())
218
219 a.wavwrite(n.hstack((T_i,T_i,T_i,T_i,T_i,T_i)).T,
220                           "ruidosaFaixa4.wav",f_a)
```

B.2.6 Bela Rugosi

Pequena peça musical de 96 segundos para demonstração da rugosidade causada por oscilações com frequencias $\approx 13-30Hz$, conceitos apresentados na subseção 2.2.5. A peça é também disponibilizada online junto à toolbox MASSA.(2)

bela rugosi

```
#-*- coding: utf8 -*-
import numpy as n, scikits.audiolab as a
H=n.hstack
V=n.vstack

f_a = 44100. # Hz, frequência de amostragem

# Tamanho da LUT > 2**10 para usar também em oscilacoes
# de comprimento de onda grandes (LFO)
Lambda_tilde=Lt=(2.**5)*1024.
# Senoide
foo=n.linspace(0,2*n.pi,Lt,endpoint=False)
S_i=n.sin(foo) # um período da senóide com T amostras

# Quadrada:
Q_i=n.hstack(  ( n.ones(Lt/2)*-1 , n.ones(Lt/2) )  )

# Triangular:
foo=n.linspace(-1,1,Lt/2,endpoint=False)
Tr_i=n.hstack(  ( foo , foo*-1 )   )

# Dente de Serra:
D_i=n.linspace(-1,1,Lt)


def v(f=200,d=2.,tab=S_i,fv=2.,nu=2.,tabv=S_i):
    Lambda=n.floor(f_a*d)
    ii=n.arange(Lambda)
    Lv=float(len(tabv))

    Gammav_i=n.floor(ii*fv*Lv/f_a) # índices para a LUT
    Gammav_i=n.array(Gammav_i,n.int)
    # padrão de variação do vibrato para cada amostra
    Tv_i=tabv[Gammav_i%int(Lv)]

    # frequência em Hz em cada amostra
    F_i=f*(   2.**(  Tv_i*nu/12.  )   )
    # a movimentação na tabela por amostra
```

```
    D_gamma_i=F_i*(Lt/float(f_a))
    Gamma_i=n.cumsum(D_gamma_i) # a movimentação na tabela total
    Gamma_i=n.floor( Gamma_i) # já os índices
    Gamma_i=n.array( Gamma_i, dtype=n.int) # já os índices
    return tab[Gamma_i%int(Lt)] # busca dos índices na tabela


dd=v(tabv=Tr_i ,d=2,fv=35.,nu=7.0)
dd2=v(tabv=D_i ,d=2,fv=35.,nu=7.0)

dd3=v(tabv=Tr_i,d=2,fv=35.,nu=7.0)
dd4=v(tabv=S_i ,d=2,fv=35.,nu=7.0)

dd5=v(tabv=Q_i ,d=2,fv=35.,nu=7.0)
dd6=v(tabv=S_i ,d=2,fv=35.,nu=7.0)

dd7=v(tabv=Q_i ,d=2,fv=35.,nu=7.0)
dd8=v(tabv=D_i ,d=2,fv=35.,nu=7.0)

zz=V((H((dd,dd3,dd5,dd7)), H((dd2,dd4,dd6,dd8))))
aa1=n.array(list(v(tabv=Q_i,fv=.5,f=200,nu=9))*4)
aa2=n.array(v(tabv=Tr_i,fv=.25/2.,f=200,nu=9,d=8.))
aa3=n.array(v(fv=.25/2.,f=200,nu=9,d=8.))

aa=n.vstack(( (zz+aa1).T*.5, (zz+aa2).T*.5,(zz+aa3).T*.5))

dd= v(tab=Tr_i,tabv=Tr_i ,d=2,fv=35.,nu=7.0)
dd2=v(tab=Tr_i,tabv=D_i ,d=2,fv=35.,nu=7.0)

dd3=v(tab=Tr_i,tabv=Tr_i,d=2,fv=35.,nu=7.0)
dd4=v(tab=Tr_i,tabv=S_i ,d=2,fv=35.,nu=7.0)

dd5=v(tab=Tr_i,tabv=Q_i ,d=2,fv=35.,nu=7.0)
dd6=v(tab=Tr_i,tabv=S_i ,d=2,fv=35.,nu=7.0)

dd7=v(tab=Tr_i,tabv=Q_i ,d=2,fv=35.,nu=7.0)
dd8=v(tab=Tr_i,tabv=D_i ,d=2,fv=35.,nu=7.0)

zz=V((H((dd,dd3,dd5,dd7)), H((dd2,dd4,dd6,dd8))))
aa1=n.array(list(v(tabv=Q_i,fv=.5,f=200,nu=9))*4)
aa2=n.array(v(tabv=Tr_i,fv=.25/2.,f=200,nu=9,d=8.))
aa3=n.array(v(fv=.25/2.,f=200,nu=9,d=8.))

aa=n.vstack(( aa, (zz+aa1).T*.5, (zz+aa2).T*.5,(zz+aa3).T*.5))

dd= v(tab=Q_i,tabv=Tr_i ,d=2,fv=35.,nu=7.0)
dd2=v(tab=Q_i,tabv=D_i ,d=2,fv=35.,nu=7.0)

dd3=v(tab=Q_i,tabv=Tr_i,d=2,fv=35.,nu=7.0)
dd4=v(tab=Q_i,tabv=S_i ,d=2,fv=35.,nu=7.0)
```

```
89
90  dd5=v(tab=Q_i,tabv=Q_i ,d=2,fv=35.,nu=7.0)
91  dd6=v(tab=Q_i,tabv=S_i ,d=2,fv=35.,nu=7.0)
92
93  dd7=v(tab=Q_i,tabv=Q_i ,d=2,fv=35.,nu=7.0)
94  dd8=v(tab=Q_i,tabv=D_i ,d=2,fv=35.,nu=7.0)
95
96  zz=V((H((dd,dd3,dd5,dd7)), H((dd2,dd4,dd6,dd8))))
97  aa1=n.array(list(v(tabv=Q_i,fv=.5,f=200,nu=9))*4)
98  aa2=n.array(v(tabv=Tr_i,fv=.25/2.,f=200,nu=9,d=8.))
99  aa3=n.array(v(fv=.25/2.,f=200,nu=9,d=8.))
100
101 aa=n.vstack(( aa, (zz+aa1).T*.5, (zz+aa2).T*.5,(zz+aa3).T*.5))
102
103 dd= v(tab=D_i,tabv=Tr_i ,d=2,fv=35.,nu=7.0)
104 dd2=v(tab=D_i,tabv=D_i ,d=2,fv=35.,nu=7.0)
105
106 dd3=v(tab=D_i,tabv=Tr_i,d=2,fv=35.,nu=7.0)
107 dd4=v(tab=D_i,tabv=S_i ,d=2,fv=35.,nu=7.0)
108
109 dd5=v(tab=D_i,tabv=Q_i ,d=2,fv=35.,nu=7.0)
110 dd6=v(tab=D_i,tabv=S_i ,d=2,fv=35.,nu=7.0)
111
112 dd7=v(tab=D_i,tabv=Q_i ,d=2,fv=35.,nu=7.0)
113 dd8=v(tab=D_i,tabv=D_i ,d=2,fv=35.,nu=7.0)
114
115 zz=V((H((dd,dd3,dd5,dd7)), H((dd2,dd4,dd6,dd8))))
116 aa1=n.array(list(v(tabv=Q_i,fv=.5,f=200,nu=9))*4)
117 aa2=n.array(v(tabv=Tr_i,fv=.25/2.,f=200,nu=9,d=8.))
118 aa3=n.array(v(fv=.25/2.,f=200,nu=9,d=8.))
119
120 aa=n.vstack(( aa, (zz+aa1).T*.5, (zz+aa2).T*.5,(zz+aa3).T*.5))
121
122 print("BellaRugosiSdadE.wav escrito")
123 a.wavwrite(aa,"BellaRugosiSdadE.wav",f_a)
```

B.2.7 Chorus infantil

Script para demonstração do efeito *chorus* (inspirado em coro de cantores) . Sintetiza 4 pequenas montagens de 8-32 segundos com os conceitos apresentados na subseção 2.2.6 e disponibilizada online junto ao toolbox MASSA.(2)

B.2.8 ADa e SaRa

Pequenas peças musicais para demonstração da envoltória ADSR. Sintetiza 7 peças de 5-17 segundos, cetradas nos conceitos apresentados na subseção 2.2.6 e disponibilizada online junto à MASSA.(2)

B.3 Peças referentes à seção 2.3

B.3.1 Intervalos entre alturas

Pequena peça musical de 45 segundos para exploração do sistema de intervalos, demonstrativa dos conceitos apresentados na subseção 2.3.1, e disponibilizada online junto ao toolbox MASSA.

B.3.2 Cristais

Pequena peça musical de 64 segundos para demonstração das escalas simétricas, conceitos apresentados na subseção 2.3.1, e disponibilizada online junto à MASSA.(2)

cristais

```
#-*- coding: utf8 -*-
import numpy as n, scikits.audiolab as a
H=n.hstack
V=n.vstack

f_a = 44100. # Hz, frequência de amostragem

Lambda_tilde=Lt=1024.*16

# Senoide
foo=n.linspace(0,2*n.pi,Lt,endpoint=False)
S_i=n.sin(foo) # um período da senóide com T amostras

# Quadrada:
Q_i=n.hstack(  ( n.ones(Lt/2)*-1 , n.ones(Lt/2) )  )

# Triangular:
foo=n.linspace(-1,1,Lt/2,endpoint=False)
Tr_i=n.hstack(  ( foo , foo*-1 )   )

# Dente de Serra:
D_i=n.linspace(-1,1,Lt)

def v(f=200,d=2.,tab=S_i,fv=2.,nu=2.,tabv=S_i):
    Lambda=n.floor(f_a*d)
    ii=n.arange(Lambda)
    Lv=float(len(tabv))

    Gammav_i=n.floor(ii*fv*Lv/f_a) # índices para a LUT
```

```
    Gammav_i=n.array(Gammav_i,n.int)
    # padrão de variação do vibrato para cada amostra
    Tv_i=tabv[Gammav_i%int(Lv)]

    # frequência em Hz em cada amostra
    F_i=f*(   2.**(  Tv_i*nu/12.  )   )
    # a movimentação na tabela por amostra
    D_gamma_i=F_i*(Lt/float(f_a))
    Gamma_i=n.cumsum(D_gamma_i) # a movimentação na tabela total
    Gamma_i=n.floor( Gamma_i) # já os índices
    Gamma_i=n.array( Gamma_i, dtype=n.int) # já os índices
    return tab[Gamma_i%int(Lt)] # busca dos índices na tabela

def A(fa=2.,V_dB=10.,d=2.,taba=S_i):
    Lambda=n.floor(f_a*d)
    ii=n.arange(Lambda)
    Lt=float(len(taba))
    Gammaa_i=n.floor(ii*fa*Lt/f_a) # índices para a LUT
    Gammaa_i=n.array(Gammaa_i,n.int)
    # variação da amplitude em cada amostra
    A_i=taba[Gammaa_i%int(Lt)]
    A_i=A_i*10.**(V_dB/20.)
    return A_i

def adsr(som,A=10.,D=20.,S=-20.,R=100.,xi=1e-2):
    a_S=10**(S/20.)
    Lambda=len(som)
    Lambda_A=int(A*f_a*0.001)
    Lambda_D=int(D*f_a*0.001)
    Lambda_R=int(R*f_a*0.001)

    ii=n.arange(Lambda_A,dtype=n.float)
    A=ii/(Lambda_A-1)
    A_i=A
    ii=n.arange(Lambda_A,Lambda_D+Lambda_A,dtype=n.float)
    D=1-(1-a_S)*(   ( ii-Lambda_A )/( Lambda_D-1) )
    A_i=n.hstack(  (A_i, D  )   )
    S=n.ones(Lambda-Lambda_R-(Lambda_A+Lambda_D),dtype=n.float)*a_S
    A_i=n.hstack( ( A_i, S )  )
    ii=n.arange(Lambda-Lambda_R,Lambda,dtype=n.float)
    R=a_S-a_S*((ii-(Lambda-Lambda_R))/(Lambda_R-1))
    A_i=n.hstack(  (A_i,R)  )

    return som*A_i


# 2..7.5. Escalas simétricas
Ec_i=[0.,1.,2.,3.,4.,5.,6.,7.,8.,9.,10.,11.]
Et_i=[0.,2.,4.,6.,8.,10.]
Etm_i=[0.,3.,6.,9.]
```

```
80  EtM_i=[0.,4.,8.]
81  Ett_i=[0.,6.]
82
83  notast_A=[adsr( v(200*2**(i/12.),d=.2 ) ) for i in Et_i]
84
85  notast_D=[adsr( v(400*2**(-i/12.),d=.2 ) ) for i in Et_i]
86  s=H( notast_D +notast_A+notast_D)
87  s1=n.copy(s)
88
89  notastm_A=[adsr( v(200*2**(i/12.),d=.2 ) ) for i in Etm_i]
90
91  notastm_D=[adsr( v(400*2**(-i/12.),d=.2 ) ) for i in Etm_i]
92  s=H(( notastm_D +notastm_A+notastm_D))
93
94  notastM_A=[adsr( v(200*2**(i/12.),d=.2 ) ) for i in  EtM_i]
95
96  notastM_D=[adsr( v(400*2**(-i/12.),d=.2 ) ) for i in EtM_i]
97  s=H(( [list(s)]+ notastM_D +notastM_A+notastM_D))
98
99  notastt_A=[adsr( v(200*2**(i/12.),d=.2 ) ) for i in  Ett_i]
100
101 notastt_D=[adsr( v(400*2**(-i/12.),d=.2 ) ) for i in Ett_i]
102 s=H(( [list(s)]+ notastt_D +notastt_A+notastt_D))
103
104 s=H((s,s[::-1]))
105 s=H((s,s[::2],s[::-2]))
106 s=H((  s,s+H((s[::2],s[::-2])),
107        s+H((s[::4],s[::-4],s[::4],s[::-8],s[::-8])) ))
108
109 s=((s-s.min())/(s.max()-s.min()))*2.-1.
110 a.wavwrite(s,"cristais.wav",f_a) # escrita do som
```

B.3.3 Micro tom

Pequena peça musical para demonstração da microtonalidade. Usados de quartos de tom e de sétimos de oitava, conceitos apresentados na subseção 2.3.1. Disponibilizada *online* junto à MASSA.(2)

micro tom

```
#-*- coding: utf8 -*-
import numpy as n, scikits.audiolab as a
import random as r
H=n.hstack
V=n.vstack

f_a = 44100. # Hz, frequência de amostragem

########## 2.2.1 Tabela de busca (LUT)
Lambda_tilde=Lt=1024.*16

# Senoide
foo=n.linspace(0,2*n.pi,Lt,endpoint=False)
S_i=n.sin(foo) # um período da senóide com T amostras

# Quadrada:
Q_i=n.hstack(  ( n.ones(Lt/2)*-1 , n.ones(Lt/2) )  )

# Triangular:
foo=n.linspace(-1,1,Lt/2,endpoint=False)
Tr_i=n.hstack(  ( foo , foo*-1 )   )

# Dente de Serra:
D_i=n.linspace(-1,1,Lt)


def v(f=200,d=2.,tab=S_i,fv=2.,nu=2.,tabv=S_i):
    Lambda=n.floor(f_a*d)
    ii=n.arange(Lambda)
    Lv=float(len(tabv))

    Gammav_i=n.floor(ii*fv*Lv/f_a) # índices para a LUT
    Gammav_i=n.array(Gammav_i,n.int)
    # padrão de variação do vibrato para cada amostra
    Tv_i=tabv[Gammav_i%int(Lv)]

    # frequência em Hz em cada amostra
    F_i=f*(   2.**(  Tv_i*nu/12.  )   )
```

```
39      # a movimentação na tabela por amostra
40      D_gamma_i=F_i*(Lt/float(f_a))
41      Gamma_i=n.cumsum(D_gamma_i) # a movimentação na tabela total
42      Gamma_i=n.floor( Gamma_i) # já os índices
43      Gamma_i=n.array( Gamma_i, dtype=n.int) # já os índices
44      return tab[Gamma_i%int(Lt)] # busca dos índices na tabela
45
46
47  def A(fa=2.,V_dB=10.,d=2.,taba=S_i):
48      Lambda=n.floor(f_a*d)
49      ii=n.arange(Lambda)
50      Lt=float(len(taba))
51      Gammaa_i=n.floor(ii*fa*Lt/f_a) # índices para a LUT
52      Gammaa_i=n.array(Gammaa_i,n.int)
53      A_i=taba[Gammaa_i%int(Lt)]
54      A_i=A_i*10.**(V_dB/20.)
55      return A_i
56
57
58  def adsr(som,A=10.,D=20.,S=-20.,R=100.,xi=1e-2):
59      a_S=10**(S/20.)
60      Lambda=len(som)
61      Lambda_A=int(A*f_a*0.001)
62      Lambda_D=int(D*f_a*0.001)
63      Lambda_R=int(R*f_a*0.001)
64
65      ii=n.arange(Lambda_A,dtype=n.float)
66      A=ii/(Lambda_A-1)
67      A_i=A
68      ii=n.arange(Lambda_A,Lambda_D+Lambda_A,dtype=n.float)
69      D=1-(1-a_S)*(    ( ii-Lambda_A )/( Lambda_D-1) )
70      A_i=n.hstack(  (A_i, D  )   )
71      S=n.ones(Lambda-Lambda_R-(Lambda_A+Lambda_D),dtype=n.float)*a_S
72      A_i=n.hstack( ( A_i, S )  )
73      ii=n.arange(Lambda-Lambda_R,Lambda,dtype=n.float)
74      R=a_S-a_S*((ii-(Lambda-Lambda_R))/(Lambda_R-1))
75      A_i=n.hstack(  (A_i,R)  )
76
77      return som*A_i
78
79
80  def ac(f=200.,notas=[0.,4.,7.,12.],tab=S_i):
81      acorde=adsr(v(tab=tab,f=f*2.**(notas[-1]/12.),nu=0))
82      for na in notas[:-1]:
83          acorde+=adsr(v(tab=tab,f=f*2**(na/12.),nu=0))
84
85      return acorde
86
87
88  # Microtonalidade
```

```
epslon=2**(1/12.)  # divisão da oitava em 12 semitons
# quartos de tom sao metade do semitom
ff=[0.,0.5,3.5,4.,4.5,5. ,  7.,7.5]
dd=[.5,.25,  .25,.5/2,.25, .25,.5/4,.5/4]
tt=[adsr(v(tab=Tr_i,f=200*2**(f/12),d=d,nu=0),R=30.) for
        f,d in zip(ff,dd)]
s=H((tt+tt[::-1]))

ff=[3.5,4.,4.5,  6.5, 7.,7.5, 7. ,  6.5]
dd=[.5,.25, .25,  .5/2, .25, .25,.5/4,.5/4]
tt=[adsr(v(tab=Tr_i,f=200*2**(f/12),d=d,nu=0),R=30.) for
        f,d in zip(ff,dd)]
so=H((tt+tt[::-1]))
s=H(([s]+tt+tt[::-1]))

# ou em 7 grados na oitava com um estilo tradicional tailandês
epslon=2**(1/7.)
ff=[0.,1.,2.,3.,4.,5.,6.,7.]
dd=[.8,0.4,0.2,.1,0.4]

ff=[200.*2.**(r.choice(ff)/7.) for i in xrange(15*3)] # 15 notas
dd=[r.choice(dd) for i in              xrange(15*3)] # 15 notas

tt=[adsr(v(tab=Tr_i,f=f,d=d,nu=0),R=50.) for f,d in zip(ff,dd)]
s=H(([s]+tt+tt[::-1]))

sa=H((s[::2],s[::-2],s[::2],s[::-2]))

ff=[0.,1.,2.,3.,4.,5.,6.,7.]
ff=[50.*2.**(r.choice(ff)/7.) for i in xrange(15*3)] # 15 notas
dd=[.8,0.4,0.2,.1,0.4]
dd=[r.choice(dd) for i in              xrange(15*3)] # 15 notas

tt=[adsr(v(tab=Tr_i,f=f,d=d,nu=7.,fv=12),R=50.) for
        f,d in zip(ff,dd)]
sg=H((tt+tt[::-1]))*2.

m=min(len(so),len(sa),len(sg))
foo=H((so[:m]+sa[:m]+sg[:m]))

s=H((s,foo,foo,foo,foo,foo,foo,foo))

s=((s-s.min())/(s.max()-s.min()))*2.-1.
a.wavwrite(s,"microTom.wav",f_a)
```

B.3.4 Acorde cedo

Pequena peça musical de 40 segundos para demonstração dos encadeamentos tonais básicos, conceitos apresentados na subseção 2.3.2, e disponibilizada online como parte da MASSA.(2)

B.3.5 Conta ponto

Pequena montagem musical de 17 segundos para demonstração de vozes conduzidas por regras simples para manterem independência, conceitos apresentados na subseção 2.3.3, e disponibilizada online junto à MASSA.(2)

B.3.6 Poli Hit Mia

Pequena montagem musical de 99 segundos para demonstração da métrica musical e das diferentes contagens, conceitos apresentados na seção 2.3.4, e disponibilizada online junto à MASSA.(2)

B.3.7 Dirracional

Pequena peça musical de 22 segundos para demonstração das estruturas direcionais e posicionamentos dos clímax, conceitos apresentados na seção 2.3, e disponibilizada online junto à MASSA.(2)

dirracional

```
#-*- coding: utf8 -*-
import numpy as n, scikits.audiolab as a
import random as r
H=n.hstack
V=n.vstack

f_a = 44100. # Hz, frequência de amostragem

########## 2.2.1 Tabela de busca (LUT)
```

```
Lambda_tilde=Lt=1024.*16

# Senoide
foo=n.linspace(0,2*n.pi,Lt,endpoint=False)
S_i=n.sin(foo) # um período da senóide com T amostras

# Quadrada:
Q_i=n.hstack(  ( n.ones(Lt/2)*-1 , n.ones(Lt/2) )  )

# Triangular:
foo=n.linspace(-1,1,Lt/2,endpoint=False)
Tr_i=n.hstack(  ( foo , foo*-1 )   )

# Dente de Serra:
D_i=n.linspace(-1,1,Lt)


def v(f=200,d=2.,tab=S_i,fv=2.,nu=2.,tabv=S_i):
    Lambda=n.floor(f_a*d)
    ii=n.arange(Lambda)
    Lv=float(len(tabv))

    Gammav_i=n.floor(ii*fv*Lv/f_a) # índices para a LUT
    Gammav_i=n.array(Gammav_i,n.int)
    # padrão de variação do vibrato para cada amostra
    Tv_i=tabv[Gammav_i%int(Lv)]

    # frequência em Hz em cada amostra
    F_i=f*(   2.**(  Tv_i*nu/12.  )   )
    # a movimentação na tabela por amostra
    D_gamma_i=F_i*(Lt/float(f_a))
    Gamma_i=n.cumsum(D_gamma_i) # a movimentação na tabela total
    Gamma_i=n.floor( Gamma_i) # já os índices
    Gamma_i=n.array( Gamma_i, dtype=n.int) # já os índices
    return tab[Gamma_i%int(Lt)] # busca dos índices na tabela

def A(fa=2.,V_dB=10.,d=2.,taba=S_i):
    Lambda=n.floor(f_a*d)
    ii=n.arange(Lambda)
    Lt=float(len(taba))
    Gammaa_i=n.floor(ii*fa*Lt/f_a) # índices para a LUT
    Gammaa_i=n.array(Gammaa_i,n.int)
    # variação da amplitude em cada amostra
    A_i=taba[Gammaa_i%int(Lt)]
    A_i=A_i*10.**(V_dB/20.)
    return A_i

def adsr(som,A=10.,D=20.,S=-20.,R=100.,xi=1e-2):
    a_S=10**(S/20.)
    Lambda=len(som)
```

```
60      Lambda_A=int(A*f_a*0.001)
61      Lambda_D=int(D*f_a*0.001)
62      Lambda_R=int(R*f_a*0.001)
63
64      ii=n.arange(Lambda_A,dtype=n.float)
65      A=ii/(Lambda_A-1)
66      A_i=A
67      ii=n.arange(Lambda_A,Lambda_D+Lambda_A,dtype=n.float)
68      D=1-(1-a_S)*(   ( ii-Lambda_A )/( Lambda_D-1) )
69      A_i=n.hstack(  (A_i, D  )   )
70      S=n.ones(Lambda-Lambda_R-(Lambda_A+Lambda_D),dtype=n.float)*a_S
71      A_i=n.hstack( ( A_i, S )  )
72      ii=n.arange(Lambda-Lambda_R,Lambda,dtype=n.float)
73      R=a_S-a_S*((ii-(Lambda-Lambda_R))/(Lambda_R-1))
74      A_i=n.hstack(  (A_i,R)  )
75
76      return som*A_i
77
78  seq=[(i-1.)/i for i in range(60)[1::2]]
79  # particionamento d oitava em 17 grados, 18 com a oitava
80  seq=[i for i in n.linspace(0,1,14)]
81  s1=seq[::2]+seq[::-2]  # simetrica
82  s2=seq[::4]+seq[::-2]  # na primeira metade
83  s3=seq[::2]+seq[::-4]  # na segunda metade
84  s4=seq  # no comec
85  s5=seq[::-1]  # no fim
86  pausa=p=[0.]*8
87  s_=s1+p+s2+p+s3+p+s4+p+s5+p+p
88
89  s=H(([adsr(v(tab=Tr_i,f=800.*2.**(ss),nu=0.5,d=.2,fv=20),
90                              70.,100.,R=20.) for ss in s_]))
91  s=((s-s.min())/(s.max()-s.min()))*2-1
92
93  a.wavwrite(s,"dirracional.wav",f_a)
```

APÊNDICE C – Finite Groups in Granular and Unit Synthesis *e a síntese de um EP*

C.1 FIGGUS

Escrito como um módulo python, o FIGGUS sintetiza estruturas musicais através de permutações, como apresentadas na subseção 2.3.6. Utiliza os princípios da MASSA e os códigos foram reescritos para python nativo, i.e. sem a utilização de bibliotecas externas como o Numpy e o Audiolab, para facilitar o uso de terceiros. O FIGGUS é parte da toolbox MASSA como exemplo de implementação dos princípios da *toolbox* com as biblitecas padrão da linguagem Python.(2)

C.1.1 FIGGUS.py

Arquivo principal, possui todas as rotinas.

C.1.2 tables.py

Arquivo auxiliar para tratar as tabelas separadamente.

C.1.3 __init__.py

Inicialização do módulo.

C.2 PPEPPS: músicas de um EP solvente

O PPEPPS (Pure Python EP: Projeto Solvente) usa o FIGGUS para sintetizar um EP inteiro. As músicas estão abaixo, junto com o arquivo que executa cada uma.

C.2.1 RUNME make EP MUSIC.py

Arquivo que executa os outros um por um para sintetizar as músicas do EP.

RUNME_make_now_an_EP_MUSIC.py

```
import os

os.mkdir('ep-Projeto_Solvente')

execfile("examples/triangulo3_B.py")
os.system('mv sound.wav ep-Projeto_Solvente/01-Eter.wav')
print("gravado o arquivo 'ep-Projeto_Solvente/01-Eter.wav'")

execfile("examples/try4_B.py")
os.system('mv sound.wav ep-Projeto_Solvente/04-Butano.wav')
print("gravado o arquivo 'ep-Projeto_Solvente/04-Butano.wav'")

execfile("examples/try5_B.py")
os.system('mv sound.wav ep-Projeto_Solvente/05-Thinner.wav')
print("gravado o arquivo 'ep-Projeto_Solvente/05-Thinner.wav'")

execfile("examples/try6_B.py")
os.system('mv sound.wav ep-Projeto_Solvente/06-Tolueno.wav')
print("gravado o arquivo 'ep-Projeto_Solvente/06-Tolueno.wav'")

execfile("examples/try7.py")
os.system('mv both.wav ep-Projeto_Solvente/07-Benzina.wav')
print("gravado o arquivo 'ep-Projeto_Solvente/07-Benzina.wav'")

execfile("examples/try3_B.py")
os.system('mv sound.wav ep-Projeto_Solvente/08-LSA.wav')
print("gravado o arquivo 'ep-Projeto_Solvente/08-LSA.wav'")

execfile("examples/try2.py")
os.system('mv sound.wav ep-Projeto_Solvente/09-Cloroformio.wav')
print("gravado o arquivo 'ep-Projeto_Solvente/09-Cloroformio.wav'")

execfile("examples/try5.py")
```

```
os.system('mv sound.wav ep-Projeto_Solvente/10-Agua.wav')
print("gravado o arquivo 'ep-Projeto_Solvente/10-Agua.wav'")
```

C.2.2 Éter

```
import figgus as f

n=3
grains=[f.UnitGrain(),f.UnitGrain(),f.UnitGrain()]
s=f.Sequence(grains)

def doCompass(durs,freqs,perms,ncomps):
    for i in xrange(n):
        s.ordered_units[i].freq=eval(freqs)
        s.ordered_units[i].duration=eval(durs)

    p=f.Pattern(s,f.PermutationPattern(perms),ncomps)
    p.synthesizeSonicVectors()
    return p.sonic_vector

semitom = "2**(1/12.)" # 12 semintons sums to an octave
dur=".5"
sound=[]

####


sounda=doCompass(dur,"200*("+semitom+")**(3*i)",(3,1,2),3*4)
soundb=doCompass(dur,"200*2*("+semitom+")**(3*i)",(2,3,1),3*4)
sound = sounda+[(i+j)*.5 for i,j in zip(sounda,soundb)]

dur=".3"
sounda1=doCompass(dur,"100*("+semitom+")**(3*i)",(3,1,2),3*4)
soundb1=doCompass(dur,"200*(2/3.)*("+semitom+")**(3*i)",(2,3,1),3*4)
stemp = [(i+j)*.5 for i,j in zip(sounda1,soundb1)]
sound += stemp

sound += sounda1+soundb1

sounda=doCompass(dur,"200*("+semitom+")**(3*i)",(3,1,2),3*4)
soundb=doCompass(dur,"200*("+semitom+")**(3*i)",(2,3,1),3*4)

stemp2 = [(i+j)*.5 for i,j in zip(sounda,soundb)]
stemp3 = [(i+j)*.5 for i,j in zip(stemp,stemp2)]
sound += 2*stemp3

io=f.IOUtils()
io.recordFile(sound) #records a mono 16 bit

```

```
45 print("\n== triangulo gira em crescente ==\n")
```

C.2.3 Benzina

```
import figgus as f

n=4
grains=[f.UnitGrain(),f.UnitGrain(),f.UnitGrain(),f.UnitGrain()]
s=f.Sequence(grains)

def doCompass(durs,freqs,perms,ncomps):
    for i in xrange(n):
    s.ordered_units[i].freq=eval(freqs)
    s.ordered_units[i].duration=eval(durs)
    #print 1*(i+1)

    p=f.Pattern(s,f.PermutationPattern(perms),ncomps)
    p.synthesizeSonicVectors()
    return p.sonic_vector

semitom = "2**(1/12.)"
dur=".2"
sound=[]

####
sound1=doCompass(dur,"200*("+semitom+")**(3*i)",(2,3,1),3)
sound1B=doCompass(dur,"4*200*("+semitom+")**(3*i)",(2,3,1),3)
sound1C=doCompass(dur,".5*200*("+semitom+")**(3*i)",(2,3,1),3)
# Inverting quirality
sound2=doCompass(dur,"200*("+semitom+")**(3*i)",(3,1,2),3)
sound2B=doCompass(dur,"4*200*("+semitom+")**(3*i)",(3,1,2),3)
sound2C=doCompass(dur,".5*200*("+semitom+")**(3*i)",(3,1,2),3)
silencio=doCompass(dur,"0",(2,3,1),1)



left=[(i+j+k)*.32 for i,j,k in zip(sound1,sound2B,sound1C,)]
left1=left[:]
right=[(i+j+k)*.32 for i,j,k in zip(sound2,sound1B,sound2C)]
right1=right[:]
left += right1
right += left1
left*=2
right*=2

left2=[(i+j+k)*.32 for i,j,k in zip(sound1,sound1B,sound2C,)]
left+=left2
right2=[(i+j+k)*.32 for i,j,k in zip(sound2,sound2B,sound1C)]
right+=right2
left+=right2
right+=left2
```

```
48
49 left30=[(i+j+k)*.32 for i,j,k in zip(sound1,sound2B,sound2C)]
50 left+=left30
51 right30=[(i+j+k)*.32 for i,j,k in zip(sound2,sound2B,sound1C)]
52 right+=right30
53 left+=right30
54 right+=left30
55
56
57 left3=[(i+j+k)*.32 for i,j,k in zip(sound2,sound2B,sound2C)]
58 left+=left3
59 right3=[(i+j+k)*.32 for i,j,k in zip(sound2,sound2B,sound1C)]
60 right+=right3
61 left+=right3
62 right+=left3
63
64 ###===
65
66 L=len(sound2)
67 left33= sound2[:L/3]+sound2B[L/3:2*L/3]+sound2C[2*L/3:] # Sobe
68 right33=sound2C[:L/3]+sound2[L/3:2*L/3]+sound2B[2*L/3:] # Desce
69 left+=left33+right33
70 right+=right33+left33
71
72 left4= sound1[:L/3]+sound1C[L/3:2*L/3]+sound1B[2*L/3:] # Sobe
73 right4=sound1B[:L/3]+sound1[L/3:2*L/3]+sound1C[2*L/3:] # Sobe
74 left+=(left4+right4)            *                  2
75 right+=(right4+left4)           *                  2
76
77 left5= sound1[:L/3]+sound1B[L/3:2*L/3]+sound1C[2*L/3:] # Desce
78 right5=sound1B[:L/3]+sound1C[L/3:2*L/3]+sound1[2*L/3:] # Desce
79 left+=(left5+right5)            *                  2
80 right+=(right5+left5)           *                  2
81
82 left6= sound2[:L/3]+sound2B[L/3:2*L/3]+sound2C[2*L/3:] # Desce
83 right6=sound2C[:L/3]+sound2B[L/3:2*L/3]+sound2[2*L/3:] # Sobe
84 left+=left6+right6
85 right+=right6+left6
86
87 ###===
88
89
90 left+=left3
91 right+=right3
92 left+=right3
93 right+=left3
94
95 ####
96
```

```
97
98  left*=2
99  right*=2
100
101 io=f.IOUtils()
102 #io.recordFile(left,[],"left.wav") #records a mono 16 bit
103 #io.recordFile(right,[],"right.wav") #records a mono 16 bit
104 io.recordFile(left,right,"both.wav") #records a mono 16 bit
```

C.2.4 Clorofórmio

```
import figgus as f

n=4
grains=[f.UnitGrain(),f.UnitGrain(),f.UnitGrain(),f.UnitGrain()]
s=f.Sequence(grains)

#########
# Construction loop: in each compass,
# I choose some parameters for our sequence,
# More sppeciffically, i choose parameters for
# each of the 4 units/grains and a permutation

# The permutation gives periodicity, inteligibility,
# and builds third order archs in the musical discourse
# 1st order: grain
# 2nd order: sequence/compass
# 3rd order: permutation cycle
# 4th order: permutation cycleS periods toguether and
#                                   other aesthetic resources
# 5th order: the piece as it is
# 6th order: sets of pieces
# 7th order: the whole social/human context of the piece,
#                                        life of the author, etc.
#
sound=[]

# COMPASS 1-4 *.*.*.*.*.*.*.*.*.*.*.*.*.*.*.* 1
for i in xrange(n):
    s.ordered_units[i].freq=10000 - 1000*(i+1)
    s.ordered_units[i].duration=.2 + 0.05*i

s.ordered_units[1].freq=100
s.ordered_units[3].freq=500

p=f.Pattern(s,f.PermutationPattern(),8)
p.synthesizeSonicVectors()
sound+=p.sonic_vector

# COMPASS 5-8 *.*.*.*.*.*.*.*.*.*.*.*.*.*.*.* 2
for i in xrange(n):
    s.ordered_units[i].freq=10000 - 1000*(i+1)
    s.ordered_units[i].duration=.2# + 0.05*i

s.ordered_units[1].freq=100
s.ordered_units[3].freq=500

#pp=PermutationPattern()
```

```
p=f.Pattern(s,f.PermutationPattern(),8)
p.synthesizeSonicVectors()
sound+=p.sonic_vector

# COMPASS 9-12 *.*.*.*.*.*.*.*.*.*.*.*.*.* 3
for i in xrange(n):
    s.ordered_units[i].freq=10000 - 1000*(i+1)
    s.ordered_units[i].duration=.05*i#.2# + 0.05*i

s.ordered_units[1].freq=100
s.ordered_units[3].freq=500

#pp=PermutationPattern()
p=f.Pattern(s,f.PermutationPattern(),8)
p.synthesizeSonicVectors()
sound+=p.sonic_vector

##################

# COMPASS 13-16 *.*.*.*.*.*.*.*.*.*.*.*.*.* 4
for i in xrange(n):
    s.ordered_units[i].freq=10000 - 1000*(i+1)
    s.ordered_units[i].duration=.1#.2# + 0.05*i

s.ordered_units[1].freq=100
s.ordered_units[3].freq=500

#pp=PermutationPattern()
p=f.Pattern(s,f.PermutationPattern(),8)
p.synthesizeSonicVectors()
sound+=p.sonic_vector

######## 4 compassos de 4 1
# COMPASS 1-4 *.*.*.*.*.*.*.*.*.*.*.*.*.* 1
for i in xrange(n):
    s.ordered_units[i].freq=10000 - 1000*(i+1)
    s.ordered_units[i].duration=.2 + 0.05*i

s.ordered_units[1].freq=100
s.ordered_units[3].freq=500*3./2 # Uma quinta a cima o pedal/bola

#pp=PermutationPattern()s
p=f.Pattern(s,f.PermutationPattern(),8)
p.synthesizeSonicVectors()
sound+=p.sonic_vector


# COMPASS 5-8 *.*.*.*.*.*.*.*.*.*.*.*.*.* 2
for i in xrange(n):
```

```
97      s.ordered_units[i].freq=10000 - 1000*(i+1)
98      s.ordered_units[i].duration=.2# + 0.05*i
99
100 s.ordered_units[1].freq=100
101 s.ordered_units[3].freq=500*3./2 # Uma quinta a cima o pedal/bola
102
103 p=f.Pattern(s,f.PermutationPattern(),8)
104 p.synthesizeSonicVectors()
105 sound+=p.sonic_vector
106
107 # COMPASS 9-12 *.*.*.*.*.*.*.*.*.*.*.*.*.* 3
108 for i in xrange(n):
109     s.ordered_units[i].freq=10000 - 1000*(i+1)
110     s.ordered_units[i].duration=.01*i**2#.2# + 0.05*i
111
112 s.ordered_units[1].freq=100
113 s.ordered_units[3].freq=500*3./2 # Uma quinta a cima o pedal/bola
114
115 p=f.Pattern(s,f.PermutationPattern(),8)
116 p.synthesizeSonicVectors()
117 sound+=p.sonic_vector
118
119 ################
120 # COMPASS 13-16 *.*.*.*.*.*.*.*.*.*.*.*.*.* 4
121 for i in xrange(n):
122     s.ordered_units[i].freq=10000 - 1000*(i+1)
123     s.ordered_units[i].duration=.1#.2# + 0.05*i
124
125 s.ordered_units[1].freq=100
126 s.ordered_units[3].freq=500*3./2 # Uma quinta a cima o pedal/bola
127
128 p=f.Pattern(s,f.PermutationPattern(),4)
129 p.synthesizeSonicVectors()
130 sound+=p.sonic_vector
131
132 ####### 4 compass de 4 ::: 2
133 # COMPASS 1-4 *.*.*.*.*.*.*.*.*.*.*.*.*.* 1
134 for i in xrange(n):
135     s.ordered_units[i].freq=10000 - 1000*(i+1)
136     s.ordered_units[i].duration=.2 + 0.05*i
137     #print 100*(i+1)
138
139 s.ordered_units[1].freq=100
140 s.ordered_units[3].freq=500/2
141
142 #pp=PermutationPattern()
143 p=f.Pattern(s,f.PermutationPattern(),8)
144 p.synthesizeSonicVectors()
145 sound+=p.sonic_vector
```

```
146
147
148 # COMPASS 5-8 *.*.*.*.*.*.*.*.*.*.*.*.*.*.* 2
149 for i in xrange(n):
150     s.ordered_units[i].freq=10000 - 1000*(i+1)
151     s.ordered_units[i].duration=.2# + 0.05*i
152     #print 100*(i+1)
153
154 s.ordered_units[1].freq=100
155 s.ordered_units[3].freq=500/2
156
157 p=f.Pattern(s,f.PermutationPattern(),8)
158 p.synthesizeSonicVectors()
159 sound+=p.sonic_vector
160
161 # COMPASS 9-12 *.*.*.*.*.*.*.*.*.*.*.*.*.*.* 3
162 for i in xrange(n):
163     s.ordered_units[i].freq=10000 - 1000*(i+1)
164     s.ordered_units[i].duration=.2*i#.2# + 0.05*i
165
166 s.ordered_units[1].freq=100
167 s.ordered_units[3].freq=500/2
168
169 p=f.Pattern(s,f.PermutationPattern(),8)
170 p.synthesizeSonicVectors()
171 sound+=p.sonic_vector
172
173 ##################
174 # COMPASS 13-16 *.*.*.*.*.*.*.*.*.*.*.*.*.*.* 4
175 for i in xrange(n):
176     s.ordered_units[i].freq=10000 - 1000*(i+1)
177     s.ordered_units[i].duration=.1*i#.2# + 0.05*i
178
179 s.ordered_units[1].freq=100
180 s.ordered_units[3].freq=500/2
181
182 #pp=PermutationPattern()
183 p=f.Pattern(s,f.PermutationPattern(),8)
184 p.synthesizeSonicVectors()
185 sound+=p.sonic_vector
186 ###### Acabou 3 x 4 compass de 4 == tonica dominante tonica
187
188 io=f.IOUtils()
189 io.recordFile(sound)
190 print("== Ouca que 'o triangulo gira \
191         e existe uma bola do lado' (nome da peca) ==")
```

APÊNDICE D – Síntese FM e AM em escala logarítmica

$$\{t'_i\}_0^{\Lambda-1} = \left\{\cos\left[f_i.2\pi\frac{i}{f_a-1}\right]\right\}_0^{\Lambda-1} = \left\{\cos\left[f\left(\frac{f+\mu}{f}\right)^{sen\left(f'.2\pi\frac{i}{f_a-1}\right)}.2\pi\frac{i}{f_a-1}\right]\right\}_0^{\Lambda-1} =$$

$$= \left\{\cos\left[f\left(2^{\frac{\nu}{12}}\right)^{sen\left(f'.2\pi\frac{i}{f_a-1}\right)}.2\pi\frac{i}{f_a-1}\right]\right\}_0^{\Lambda-1} = \left\{\cos\left[f\left(2^{\frac{\nu}{12}\sin\left(f'.2\pi\frac{i}{f_a-1}\right)}\right).2\pi\frac{i}{f_a-1}\right]\right\}_0^{\Lambda-1} \Rightarrow$$

$$\Rightarrow \text{usando: } \left(2^x = e^{x\ln 2},\ \ e^x = \sum_0^\infty \frac{x^n}{n!},\ \ \sin^n x = y_1(n)\sum_k y_2(n,k).\sin[y_3(n,k,x)]\right) \Rightarrow$$

$$\Rightarrow \{t'_i\}_0^{\Lambda-1} = \left\{\sum_{k=-\infty}^{+\infty} J_k(\mu,f)\cos\left[y(f.f',f_a)\right]\right\}_0^{\Lambda-1} \tag{D.1}$$

Por brevidade, as identidades úteis para a obtenção da equação final, que explicita o conteúdo em frequências puras. Portanto, se utilizada a progressão exponencial de frequência, o espectro da FM perde a simplicidade. A função de Bessel dependerá não somente da profundidade de oscilação, mas também da frequência original do som. Além disso, as frequências dependem do produto das frequências da portadora e da moduladora ($f.f'$), o que também introduz complexidade.

No caso da modulação de amplitude na escala logarítmica:

$$\{t'_i\}_0^{\Lambda-1} = \{(a_{\text{máx}})^{ti'}.t_i\}_0^{\Lambda-1} = \left\{a_{\text{máx}}^{\sin\left(f'.2\pi\frac{i}{f_a-1}\right)}.P.\sin\left(f.2\pi\frac{i}{f_a-1}\right)\right\}_0^{\Lambda-1} \tag{D.2}$$

através das mesmas três identidades, observa-se um espectro com complexidade similar.

APÊNDICE E – Música digital em domínios não digitais

As tecnologias expostas neste trabalho tiveram origem em usos reais, advindas de práticas culturais e voltadas para repercussões sociais. Este material humano e artístico é o assunto deste capítulo. Pode-se dividir estes resultados em: experimentos abertos em áudio, música em tempo real, música em tempo diferido, música na matéria e repercussões no tecido social. Estes tópicos são tratados separadamente abaixo em termos do que é mantido *online*. Estas atividades extrapolam os usos musicais, especialmente devido às finalidades pedagódicas e sociais das práticas audiovisuais. Assim, por completude, alguns materiais didáticos e ferramentas *web* são descritas. Todos os desenvolvimentos desta dissertação estão em repositórios abertos.(63)

E.1 Experimentos abertos em áudio: LADSPAs, Wavelets e Redes Complexas

O código computacional feito para a arte sonora, incluindo a musical, manifesta-se como cultura pois é fruto de práticas espontâneas, diárias e coletivas. [1] A seguir estão alguns exemplos de resultados destas incidências culturais, especificamente em áudio e música.

[1] As chamadas culturas biopunk, ciberpunk, cipherpunk, hacker, digital e outras mais, possibilitadas pelos desenvolvimentos em telecomunicação, dizem respeito em menor ou maior grau à produção de código computacional como uma manifestação cultural. As elaborações de conceitos e ferramentas voltados para o compartilhamento nestas culturas estão na gênese do que se entende como 'Cultura Livre'.

Plugins LADSPA e lv2

LADSPA (Linux Audio Developers Simple Plugin API) é a API livre de *plugins* de áudio até a presente data. A última versão é a 1.1, em uso corrente até hoje e é de 2002 segundo os arquivos de cabeçalho dos códigos relacionados. O LADSPA *version* 2, abreviado lv2, é uma segunda versão do protocolo também estabelecida e estável, mas extensível, motivo pelo qual está sempre em desenvolvimento.

Na síntese, para maior qualidade, pode-se fazer a recomposição a partir do espectro desejado, como na seção 2.2.4. Menos purista porém mais eficiente é usar uma amostra curta do ruído e reproduzi-la indefinidamente através de *cross-fades*. Esta eficiência é desejada em um *plugin*, pois seus usos comuns incluem aplicações em tempo real.

Através deste trabalho, foi disponibilizado um pacote de plugins lv2 que sintetizam os ruídos branco, azul, violeta, rosa e marrom. Os códigos em C e a implementação como plugin lv2 estão no repositório git como pode ser visto em *online*[2].

Já na remoção de ruídos as abordagens são as mais variadas e extremamente dependentes da aplicação. A complexidade dos algorítmos atinge níveis de especialidade em processamento de sinais que já mereceram trabalhos dedicados. Adiante está a implementação de um removedor de ruído *'Hum'*. Este ruído é geralemte causado pela corrente alternada que alimenta os equipamentos utilizados.

A remoção de ruído *Hum* é baseada em uma sequência de filtros nó rejeita banda (veja subseção 2.2.3) idealmente dispostos nos harmônicos da frequência de oscilação da corrente elétrica. Cada um destes filtros deve permitir ajustes finos no fator de qualidade e também na frequência central pois trata-se de um sistema real passível dos mais diversos efeitos de distorção do modelo ideal. O código C/C++ e a implementação como *plugin* está no mesmo repositório

[2] Este link é o *README* do repositódio *git* com os códigos dos plugins: `http://labmacambira.git.sourceforge.net/git/gitweb.cgi?p=labmacambira/LV2;a=summary`.

disponibilizado para a síntese de ruídos.

Wavelets

Alguns dos experimentos em *wavelets* praticados estão em Audioexperiments.(64) Uma destas investidas superou em muito um experimento e constitui um protocolo de compactação de áudio desenvolvido com o Prof. Rafael Santos Mendes na FEEC/UNICAMP como consequência do mestrado realizado por André Luvizzoto sob orientação do mesmo professor. O método está bem descrito no artigo desenvolvido. Em resumo, o método consiste em particionar o áudio com sobreposições para contemplar aplicações reais sem as distorções de borda da decomposição e recomposição. Em seguida, o áudio é decomposto em árvore *Wavelet Packet*. Cada folha é então ordenada, os coeficientes com baixa energia eliminados e um polinômio encontrado pelo método dos mínimos quadrados é usado para representar os coeficientes restantes. Detalhes de escrita e leitura dos polinômios e das permutações completam o protocolo. Aplicações para *stream* de voz em tempo real foram vislumbradas e testes adicionais propostos.(65)

Redes Complexas

Com o fim de detectar nuanças emocionais no uso da linguagem natural, foram feitos alguns trabalhos em análise de fala e de texto, ambos focados no uso de redes complexas. O primeiro trabalho de processamento de texto foi apresentado na ACL de 2010, em Upsala, na Suécia, pelo Prof. Thiago Pardo, atual presidente da ACL.(66–68)

Os trabalhos focaram em distinguir polaridade em excertos de textos e de fala. Nos textos, a rede é formada a partir da sequência de palavras, em que cada palavra é um vértice e a sucessão imediata cria uma aresta. O pré-processamento dos textos consiste na lematização e na retiradas das *stop-words* e da pontuação. Nas falas, as bandas de frequência utilizadas são vértices e a transição entre uma banda e outra é uma aresta. Ambas as redes possuem arestas com peso e

podem ser utilizadas como direcionais ou não. Os trabalhos consistiam em formar estas redes com os bancos de dados correlatos, extrair medidas e aplicar reconhecimento de padrão. Ambos os estudos não fazem uso do conteúdo semântico, mas sim topológico dos sinais recebidos, e chegam a mais 80% de cobertura ou precisão em alguns casos, podendo contribuir com métodos tradicionais que fazem uso de redes semânticas, por exemplo.

E.2 Áudio e música

Este capítulo está dividido em 4 partes: música em tempo diferido (ou seja, que não é realizada em tempo real), música em tempo real, música na matéria (suporte físico em *hardware*) e música no tecido social (mobilizações humanas). O texto se baseia em exemplos reais da prática musical através do código, com aplicações e em utilização pelo autor, membros do LabMacambira.sf.net, parceiros, colaboradores e por usuários eventuais das naturezas mais diversas.

E.2.1 Música em tempo diferido: *minimum-fi* e FIGGUS

> 'I also find that intelligent people always respect the intelligence needed to construct a simple structure in a clear way that really works.'
>
> *Tom Johnson*

A realização musical em tempo diferido é o paradigma inicial da música computacional, iniciada com o *Music V*, depois com o *CSound*. Pode-se dizer que até hoje é a forma como compositores usualmente pensam a música: concebendo e escrevendo as estruturas que depois são executadas por instrumentistas ou aparelhos eletrônicos. Assim como a composição instrumental permite alguns refinamentos estruturais não presentes na improvisação instrumental, a realização musical em tempo diferido permite um detalhamento maior dos procedimentos do que a realização em tempo real.

O som musical pode ser caracterizado fisicamente, tornando possível a sua síntese digital. Estes sons e a organização deles são objeto do Capítulo 2 da presente dissertação. A utilização destes sons de forma direta, utilizando somente os recursos básicos de justaposição e sobreposição, está na pequena peça *minimum-fi*. Já o FIGGUS (FInite Groups in Granular and Unit Synthesis), utiliza os princípios do *minimum-fi* e utiliza simetrias através de permutações e de Teoria de Grupos, para a composição de músicas. O FIGGUS gera um EP[3] com um único comando. Este é o *PPEPPS*[4], como veremos a seguir.

Desta forma, esta seção exemplifica e explicita - através de dois exemplos reais - o uso de código para a síntese *musical*, desde as amostras relativas a uma nota com dada frequencia, amplitude e timbre, até a confecção de uma ferramenta derivada, que já incorpora propostas musicais e estruturas mais elaboradas.

Minimum-fi

Existe uma perene matiz estética e também tecnológica dedicada a realizar uma dada tarefa com *o mínimo* necessário. Na música, esta matiz é igualmente presente e decorre em grande parte do princípio de unidade e coerência[5]. Em código computacional, a empreitada para manifestar este princípio ele próprio de forma mínima resultou no *minimum-fi.py*, código Python em um único arquivo curto que sintetiza a música segundo as estruturas especificadas em linha. Na versão atual, de 2012 mas adiantada em 2011, os algorítmos em Python propriamente ditos somam 53 linhas e incluem apenas 5 funções. Com estas funções, estruturas musicais são criadas padrão a padrão, nota a nota, amostra por amostra. Na prática, e em linguagem cotidiana, as notas formam blocos e estruturas hierarquicamente superiores.

---

[3] *Extended Play*, um álbum musical maior que um single mas menor que um LP (*Long Play*) inteiro.

[4] Pure Python EP: Projeto Solvente.

[5] Este princípio tanto é fundamental que as escolas musicais possuem técnicas específicas, músicas possuem suas próprias convenções mantidas por toda a sua duração, os arcos mantém características, enfim, podemos até mesmo concluir que as simetrias definem o escopo musical.

Os princípios, bastante simples, são[6]:

- Deve-se ter um mecanismo de síntese sonora que possibilite a geração de unidades sonoras com diferentes timbres, controle sobre a frequência fundamental, duração e volume, como especificado na seção 2.1.

- Deve-se ser capaz de construir séries de unidades, sejam sobrepostas (e.g. acordes) ou justapostas (e.g. melodias).

Para o primeiro item, presta-se o procedimento de busca em tabelas/vetores com formas de ondas em alta resolução, chamado *lookup table*. O procedimento é barato computacionalmente, com resultados diversificados e tidos como de alta qualidade pois não acrescenta ruídos relevantes ao sinal. Uma descrição do procedimento está na subseção 2.2.1.

Através da utilização do *lookup* sucessivo (procura-se um valor na primeira tabela e este valor indica o valor na segunda tabela a ser utilizado), executa-se um *waveshaping*. Este procedimento é bastante apreciado pela simplicidade e eficácia na síntese de tímbres diversos e ricos em harmônicos e evolução temporal. Embora uma explicação exaustiva do waveshaping fuja ao escopo deste trabalho, este método se caracteriza pela aplicação de uma função não linear ao sinal de entrada. Neste caso o sinal de entrada é gerado pela primeira busca, a função não linear aplicada é a segunda busca, na outra tabela. Existem muitas formas de se executar um *waveshaping*, a seguir segue o que usamos no *minimum-fi*, que se sustenta principalmente por ser leve e simples:

Waveshaping com consultas sucessivas a tabelas

```
# T, incremento, n_amostras e dur definidos como no script anterior
ap=0.0
amostras=[0] * n_amostras
for i in xrange(n_amostras): # n_de_amostras = dur * 44100
```

[6] Osvaldo Lacerda, em seu livro *Compêndio de Teoria Elementar da Música* fala das propriedades do som musical e sua organização de forma condizente.

```
5       foo=func[int(ap)] # foo entre [-1,1]
6       bar=( bar + (T/2) )*(T/2) # centrando no meio da tabela e ampliando o âmbito
7       amostras[i] = func2[int(bar)]
8       ap = (SI + ap)%T
```

O segundo item - dos dois princípios expostos sobre o *minimum-fi* - presta-se à discretização do espaço musical. Unidades como batidas e notas tornam mais eficiente a comunicação pois a quantidade de estruturas sugeridas é maior e as estruturas são mais explícitas no discreto do que no contínuo. Roederer aponta que as próprias notas dos instrumentos musicais são um reflexo de que é mais eficiente o uso do discreto do que do contínuo para a geração de estruturas musicais.(3)

De fato, unidades bem definidas se mostram úteis na prática musical para fazer sequências de unidades. Quando as unidades são notas com frequência definida, as sequências de unidades justapostas no tempo tendem a ser compreendidas como melodias ou linhas melódicas. As sequências sobrepostas no tempo são comumente compreendidas como acordes, mas podem ser tidas simplesmente como sobreposições circunstanciais de duas ou mais linhas melódicas[7].(23)

As duas construções básicas explicitadas - baseadas na dicotomia melodia/harmonia, horizontalidade/verticalidade, justaposição/sobreposição - são realizadas através das funções *fazSequencia* e *fazAcorde* no *minimum-fi*. Vale notar elas são absolutamente equivalentes em uma análise puramente conceitual, i.e. uma delas pode ser omitida sem perda das possibilidades musicais. Isso fica particularmente óbvio quando se nota que os procedimentos de mixagem e concatenação são plenamente capazes de realizar o que estas funções realizam. Aliás, as funções nada mais são do que usos típicos e quase caricatos destes procedimentos: no *fazAcorde* a mixagem sobrepõe no tempo todas as unidades, no *fazSequencia* as unidades são todas juntapostas no tempo.

Como pode-se notar a seguir, as sequências de notas e os acordes, em última instância, são usos

[7] A música do século XX apresentou diversos modelos teóricos que quebram com este entendimento simplificado sobre a música, suas unidades básicas e estruturas relacionadas

específicos das 2 funções de síntese sonora implementadas: *lookup* e *lookupcruz*.

Realização de Sequências de Notas

```
# Note as variáveis que são fixadas de antemão
def fazSequencia(notas, f0, fator, dur_nota, dur_silencio,\
                                        funcao1, funcao2=0):
    seq=[] # a sequência é iniciada vazia
    for nota in notas: # para cada nota
        # a frequência é a fundamental vezes 2
        # elevado ao número de semitons:
        freq=f0*fator**nota
        if funcao2: # caso a funcao2 seja especificada
            # fazer o waveshaping:
            amostras=lookupcruz(funcao1,funcao2,dur_nota,freq)
        else:
            # ou faz com a forma de onda simples:
            amostras=lookup(funcao1,dur_nota,freq)
        # soma o silêncio
        amostras += [0]*int(taxa_amostragem*dur_silencio)
        seq += amostras # concatena na sequência principal
    # normalização em 2 passos:
    smin=min(seq); smax=max(seq)
    seq= [(-.5+(i-smin)/(smax-smin)) for i in seq]
    return seq # retorna a sequência pronta
```

Realização de Acordes de Notas

```
def fazAcorde(notas, f0, fator, dur_nota, dur_silencio,\
                                    funcao1, funcao2=0):
    seq=[]
    for nota in notas:
        freq=f0*fator**nota
        if funcao2:
            amostras=lookupcruz(funcao1,funcao2,dur_nota,freq)
        else:
            amostras=lookup(funcao1,dur_nota,freq)
        amostras=amostras+[0]*int(dur_silencio*taxa_amostragem)
        # para sobrepor unidades de tamanhos diferentes
        # é conveniente uma função auxiliar
        seq=somador(seq,amostras)
    # normalizaçào em duas linhas:
    smin=min(seq); smax=max(seq)
    seq=[(-.5+(i-smin)/(smax-smin)) for i in seq]
    return seq
```

A última das cinco funções utilizadas é uma soma amostra a amostra de dois sons. Para isso, é necessário completar com zeros a sequência com o menor número de amostras para somar rapidamente:

Somador (função auxiliar)

```
def somador(seq1,seq2):
    # Basicamente adiciona zeros na sequencia menor para somar rapidamente
    diff=len(seq1)-len(seq2)
    if diff > 0:
        seq2+=[0]*diff
    elif diff<0:
        seq1+=[0]*-diff

    seq=[(i+j) for i,j in zip(seq1,seq2)]
    return seq
```

Com isso o básico está coberto e o foco vai para a criação de estruturas. Por exemplo: pode-se criar as escalas completamente simétricas na oitava cromática, escalas diatônicas ou até microtonais, como as descritas na subseção 2.3.1.

O estabelecimento de pequenas sequências é bastante útil para reutilizações, variações e geração de materiais derivados, por exemplo:

Sequências diversas

```
# padrão rítmico e variações
b1=[0]*4
b2=[0]*7+[7]
b3=[0,7,0,1]
b3_2=[0,7,8,7]
b4=[0,7,7,7]
b5=[0,7]
b6=[0,19,7,12]
b7=[0,-1,0,-1]
b8=[0,-1,7,-1]
b9=[0,1,0,-1]
b10=[0,7,6,7]

# Série utilizada por Anton Webern
# em suas Variações Op.30
c=[0,1,4,3,2,5,6,9,8,7,10,11]

# Sequência em mixolídio comum, por exemplo,
# Em algumas manifestações do folclore nordestino:
d=[0,4,7,10,9,7,10]
```

O passo seguinte é sintetizar, mixar e concatenar para a obtenção de sequências musicais. A síntese de sequências e acordes pode ser feita desta forma:

Sintetizando sequências e acordes

```
# inicializamos as variáveis:
notas=[1,4,5,6,7]
fundamental=200
fator=2**(1/12.)
dur_nota=0.5
dur_silencio=0.1
onda1=senoide
onda2=dente_de_serra
# realização da sequência com senóides:
sequencia = fazSequencia( notas,
                          fundamental,
                          fator,
                          dur_nota,
                          dur_silencio,
                          onda1 )
# realização do acorde com waveshaping
acorde = fazAcorde( notas
                    fundamental,
                    fator,
                    dur_nota,
                    dur_silencio,
                    onda1,
                    onda2 )
```

Já a síntese de estruturas compostas (e.g. sequências de acordes e sobreposição de linhas melódicas), é feita com os recursos usuais da liguagem: listas em Python e vetores Numpy (mais eficientes em tempo de execução e também na simplicidade do código). Estes procedimentos são praticamente os mesmos se implementados em Scilab, C/C++, Javascript, PHP, etc.

A seguir, para fins didáticos, está a construção de acordes periódicos em python puro[8]:

Acordes periódicos

```
# aproveitando a variavel
# acorde do script anterior
dur = dur_nota + dur_silencio
silencio_1_tempo = [0]*int((dur)*taxa_amostragem)

# adicionando o acorde a cada 2 tempos 8 vezes
# ou seja, por 16 tempos, durante 4 compassos de 4
# a música terá o acorde sedo tocado a cada 2 tempos:
ac += (acorde+silencio_1_tempo)*8
```

[8] Uma implementação não didática destas três linhas de código pode ser feita em uma só linha.

Em posse destas sequências, acordes e recursos da linguagem, são formadas estruturas hierarquicamente superiores através da concatenação de estruturas, da mixagem de estruturas, e da amplificação (ou atenuação) seletiva das mesmas:

Amplificação e mixagem

```
#-# De posse das notas, fundamental, fator, durações e ondas #-#
# sequência do restrogrado das notas com waveshapping:
sequencia1 = fazSequencia(nt[::-1], f, fator, d, s, senoide, dente)
# sequência original das notas uma quinta justa acima,
# com dente de serra:
sequencia2 = fazSequencia(nt, f*(3/2.), fator, d, s, dente)

#-# Procedimentos básicos #-#
# concatenação
seq = sequencia1+sequencia2
# amplificação e concatenação
seq += [1.5*i for i in sequencia1]
# mixagem e concatenação
seq += [(i+j) for i,j in zip(sequencia1,sequencia2)]
# mixando igual acima, mas quando os comprimentos
# dos vetores não são os mesmos
seq += somador(seq,sequencia2)
```

Neste ponto, basta criar músicas e sequências de interesse estético ou para pesquisa. Os encadeamentos dependem de intensões estéticas, entendimentos musicais e estruturas abstratas que mantém a coerência e o interesse em uma peça musical. Para o leitor mais interessado, recomendamos uma visita ao capítulo 2 e à *toolbox* MASSA, onde pode-se encontrar o próprio script *minimum-fi.py* e outros experimentos. A seguir utilizamos esta base apresentada para sintetizar estruturas musicais e então um EP.

FIGGUS: FInite Groups in Granular and Unit Synthesis

Nesta subseção há um foco nas estruturas musicais. São elas, inseridas em um momento histórico e executadas por instrumentos específicos com as técnicas de época, que constituem em grande parte uma linguagem musical e músicas propriamente ditas. O FIGGUS constitui uma técnica composicional manifesta em software como ferramenta de síntese de estruturas musi-

cais. Esta técnica consiste na utilização de estruturas matemáticas que representam simetrias para a organização de materiais musicais.

O FIGGUS foi iniciado em 2006 com o físico-matemático Prof. Adolfo Maia Junior (do IMECC e NICS, ambos da UNICAMP) - bem anterior ao nascimento do *minimum-fi* - para tratar de simetrias na música com vistas à composição musical através de métodos matemáticos[9]. Mais especificamente, a proposta gerou um programa de síntese granular e de estruturas musicais através de Grupos Algébricos. O nome dado foi FIGGUS, sigla de FInite Groups in Granular and Unit Synthesis[10].

Na atual reescrita, embora ainda bastante atrás do FIGGUS original quanto à interface gráfica, a ferramenta opera diretamente em Python puro, com as biblitecas imbutidas por padrão. Isso permite com que o FIGGUS sintetize todo um EP usando somente os comandos:

Utilizando o FIGGUS para Sintetizar um EP

```
$ git clone git://labmacambira.git.sourceforge.net/gitroot\
                                          /labmacambira/FIGGUS
$ cd FIGGUS
$ python RUNME_make_now_an_EP_MUSIC.py
```

Desta forma, a ferramenta é simples de ser usada para experimentações e implementações adicionais. Outro uso planejado para o FIGGUS é a síntese de timbres e amálgamas sonoros através da Síntese Granular. Embora o foco atual seja outro, cabe algumas breves palavras sobre o assunto.

A Síntese Granular é uma área bem estabelecida tanto na acústica quanto na Computação Musical e se caracteriza pela geração de sons bastante curtos e em quantidade massiva. Tipicamente,

[9] Duas iniciações científicas trataram do assunto.

[10] Também foi usado o nome FIGGS (FInite Group in Granular Synthesis) dado que o termo *unit synthesis* não é usual na literatura. Posteriormente o primeiro autor deste trabalho recorreu novamente ao uso do nome **FIGGUS**. Isso foi motivado pela utilidade da técnica para síntese de estruturas musicais, que, nos usos que o **FIGGUS** teve, foi maior do que a utilidade para síntese de amálgamas sonoros tipicamente resultantes da síntese granular

os sons possuem entre 5 e 40 milissegundos e a quantidade destes *microsons*[11] pode chegar a milhares por segundo. O tratamento específico da síntese granular foge ao escopo deste trabalho. O leitor interessado pode consultar os artigos produzidos sobre Síntese Granular e Teoria de Grupos.(58, 59) Desta forma, o texto a seguir concentra-se em Grupos Finitos para a síntese de estruturas musicais.

Nas artes é de comum conhecimento o papel absolutamente central que as simetrias possuem. Na música, para citar somente alguns exemplos simples, há os numerosos estudos de simetrias na música de J. S. Bach, os jogos de dados de Mozart e os usos recorrentes da proporção áurea na música de Béla Bártok. Matematicamente, as *simetrias* são descritas por Grupos, e estes são definidos como um conjunto (seja G) munido de uma operação (seja $\bullet$), formando um grupo $(G, \bullet)$ satisfazendo as propriedades descritas na subseção 2.3.6. No FIGGUS, o grão ou unidade sonora é uma classe que possui apenas os seguintes atributos: duração (segundos), frequência (Hz), timbre (identificador para usar mediante implementações convenientes), intensidade (pico $\in [0, 1]$), e duração dos fades (in e out em segundos). O FIGGUS funciona com base em uma sequência de grãos especificada inicialmente e na qual operam as permutações. Aos grãos em sequência são aplicadas permutações. Para isso é bastante conveniente representar as permutações em classes próprias. A classe do padrão de permutação possui também um período de aplicação da permutação, ou seja, de quantas em quantas leituras da sequência apermutação é aplicada.

Em posse dos grãos, da sequência, das permutações e do padrão de permutações, pode-se realizar a estrutura musical em si. Basta adicionalmente especificar o número desejado de iterações da sequência. Com isso, a sequência de grãos é lida um número de vezes, aplicando as permutações na sequência de grãos segundo o padrão especificado, de forma a resultar em uma sequência musical. Note que se a permutação usar menos elementos que a sequência possui,

[11] Grãos sonoros e microsons são jargões equivalentes típicos da síntese granular. São usados para indicar sons com durações bastante curtas, como assinalado no texto.

alguns destes elementos ficarão estáticos nas iterações da sequência no padrão sonoro.

A partir das representações abstratas do padrão musical, são feitos os vetores sonoros da representação digital da música a ser realizada e pode-se escrever um arquivo de áudio propriamente dito. O mais conveniente, neste caso, é escrever um arquivo PCM (Pulse Code Modulation) em algum padrão amplamente utilizado e reconhecido. Ambos WAV e AIFF satisfazem estes requisitos. Mais especificamente, o padrão de CDs é WAV com 44100 amostras por segundo e 16 bits por amostra. As amostras dos vetores sonoros são calculados, por conveniência e convenção, no âmbito $[-1,1]$ e precisam ser normalizados para o âmbito $[-32767,32768]$ e truncados em números inteiros. Depois disso devem ser escritos em um arquivo com os *bits* como na convenção da linguagem C/C++. A bibioteca *struct* cuida dessa escrita do inteiro no formato correto, e a biblioteca *wave* escreve o cabeçalho no formato WAV adequado. Assim, a clase de escrita do vetor sonoro em arquivo comum fica assim:

Escrita do Vetor Sonoro em Arquivo WAV

```
import wave, struct
class IOUtils:
    def recordPattern(self,filename,pattern):
        self.recordFile(pattern.sonic_vector,[],"sound.wav",pattern.SR)

    def recordFile(self, sonic_vector=[], sonic_vector2=[],
                           filename="sound.wav", samplerate=44100):
        sound = wave.open(filename,'w')
        sound.setframerate(samplerate)

        sound.setsampwidth(2) # Always 16bit/sample (2 bytes)
        if not sonic_vector2:
            sound.setnchannels(1) # Mono
            sonic_vector=self.boundVector(sonic_vector)

            sonic_vector=[i*(2**15-1) for i in sonic_vector]
            sound.writeframes(struct.pack('h'*len(sonic_vector),
                                      *[int(i) for i in sonic_vector]))
        else:
            sound.setnchannels(2) # stereo
            sonic_vector=self.boundVector(sonic_vector)
            sonic_vector2=self.boundVector(sonic_vector2)

            sonic_vector=[i*(2**15-1) for i in sonic_vector]
            sonic_vector2=[i*(2**15-1) for i in sonic_vector2]
```

```
26              SV=[]
27              for i,j in zip(sonic_vector,sonic_vector2):
28                  SV.extend((i,j))
29              sound.writeframes(struct.pack('h'*len(SV),*SV))
30          sound.close()
31
32      def boundVector(self,vector):
33          """Bound vector in the [-1,1] interval"""
34          svmin=min(vector)
35          svmax=max(vector)
36          ambit=svmax-svmin
37          if svmax>1 or svmin<-1:
38              if svmax-svmin > 2:
39                  i=0
40                  for sample in vector:
41                      new_sample=(sample-svmin)/ambit # results in [0,1]
42                      new_sample=new_sample*2-1 # results in [-1,1]
43                      vector[i]=new_sample
44                      i+=1
45              elif svmax > 1:
46                  offset=svmax-ambit/2
47                  i=0
48                  for sample in vector:
49                      new_sample=sample - offset
50                      vector[i]=new_sample
51                      i+=1
52              elif svmin < -1:
53                  offset=-svmin -ambit/2
54                  i=0
55                  for sample in vector:
56                      new_sample=sample + offset
57                      vector[i] = new_sample
58                      i+=1
59          return vector
```

E.2.2 Música em tempo real: *Livecoding* e ABeatTracker (ABT)

Tornou-se usual a síntese sonora em tempo real mesmo em *laptops* populares. Assim, surgiram linguagens de domínio específico, para o áudio e a música, em sua maioria dedicadas - ou ao menos capacitadas - para interação em tempo de execução como *Puredata*, *SuperCollider*, *ChucK* e *ixilang*. Todos estes são exemplos de linguagens largamente utilizadas para a composição musical e síntese sonora em tempo real. Em outras palavras, estas linguagens possibilitam

que o usuário ouça o resultante sonoro do código utilizado e altere o código com resultados imediatos no processamento e sonoridades produzidas.

A exploração estética destas ferramentas em performances musicais é a proposta do *Livecoding*. Esta prática de *performance* se dedica especialmente à escrita de código em tempo real com vistas à projeção visual dos códigos enquanto são escritos, junto à projeção sonora que produzem. O ABeatTracker ainda pode ser considerado livecoding, mas já é um intermediário, com traços de um aplicativo ou ferramenta, embora os comandos sejam acessados pela escrita. É uma linguagem com macros para execução de rítmos através de *samples* em conjunto com instrumentos tradicionais e outras fontes sonoras/musicais externas.

Livecoding

Recentemente, grupos de ponta em música experimental estão desenvolvendo apresentações musicais públicas baseadas na escrita de código ao vivo. Este é um fenômeno cultural e estético, do qual vale ressaltar os grupos pioneiros *Slub* e *Benoît and the Mandelbrots* assim como as 'orquestras de *laptops*' *PLOrk*, *SLOrk* e *DubLOrk*[12]. Usualmente, o código é projetado para que a audiência possa ver o que está sendo escrito, no rítmo em que se escreve, e se projeta também o resultante sonoro por autofalantes.

Estas são motivações presentes em diferentes grupos[13], embora não necessariamente ou da mesmícima forma:

- A apresentação musical, com o uso do computador como instrumento, carece de recursos performáticos visuais dos instrumentos tradicionais. Os gestos são por demais discretos e a concentração do *performer*/instrumentista é bastante focada na tela do computador. Recursos performáticos são, portanto, preciosíssimos.

[12] Para maiores detalhes, veja: `http://toplap.org`.
[13] Vide 'Manifesto Live coding' em `http://toplap.org/wiki/ManifestoDraft`.

- O *feedback* auditivo do código projetado permite que o espectador infira significados dos códigos. Este recurso do *livecoding* é usado para desmistificar a programação de computadores e sua aplicação em computação musical, comumente considerada intangível.

- O código em si é um recurso poderosíssimo que permite ao usuário controlar os sons produzidos amostra por amostra ou em escalas maiores de tempo, como notas, compassos, fraseados inteiros ou mesmo em escalas maiores de tempo, como minutos, horas, dias e semanas.

- O compartilhamento do código é usual, leve e eficiente como entrega de tecnologia valiosa para a proposta estética. Isso motiva não só o programador a aplicar seus conhecimentos na música, mas também o músico a adentrar o uso de linguagens de programação para expressão de suas ideias musicais.

Desta forma, foi iniciada em 2011 uma linha de atuação em *livecoding* com a criação do duo *FooBar* composto por Vilson Vieira e o autor deste escrito. Este duo se desenvolveu no trio *FooBarBaz*/Variáveis Metasintáticas composto pelo duo e por Gilson Beck, este atuante na mesa de som e usando detecção de cor no *laptop* para incrementar a apresentação. Este trio realizou uma *performance* no *V Festival Contato*. É interessante ressaltar que, até onde se sabe, essa foi a primeira apresentação de *livecoding* no Brasil, e a maior apresentação internacional em tamanho de platéia, estima-se que entre 3 e 5 mil pessoas estavam presentes.

Foi usada a linguagem *ChucK* por apresentar os recursos que o duo inicial considerou mais apropriados, embora de forma alguma isso seja consensual na prática atual de *livecoding*. Além disso, a apresentação contou com recursos adicionais para agregar interesse, como a utilização do *cowsay* para enviar mensagens enquanto se desenrolava a música. Em especial, as trajetórias de um ponto branco no fundo do código sugeria o sono REM como uma experiência coletiva de alteração do estado de consciência. Estes recursos podem ser vistos em uso nos videos

demonstrativos[14] As linhas do *cowsay* e o *script* em linguagem Processing relacionados, assim como maiores detalhes sobre o duo e fotos da apresentação podem ser vistos em `http://wiki.nosdigitais.teia.org.br/FooBarBaz`. Estavam presentes no palco também outros membros do labMacambira.sourceforge.net, davam suporte em gravações e outros auxílios: Ricardo Fabbri, Alexandre 'Bzum', Daniel Penalva e Ivan Marin.

Especificamente sobre a prática do *livecoding*, duas pessoas executaram *scripts* em *Chuck* simultaneamente. A saber, Vilson Vieira executou ritmos, batidas bastante marcadas que serviam como base. Renato Fabbri, autor deste texto, executava linhas fluídas quasi-melódicas que formavam arcos maiores. Havia também interlúdios em que ambos se revesavam com músicas curtas e inusitadas, como em um duelo. A mixagem e espacialização dos dois canais de áudio foram controladas por um *patch* em Puredata que permitia o *cross-fading* entre os canais através da detecção do movimento das mãos. Isso foi possível através do objeto em Puredata/GEM chamado *color classify*, criado por Ricardo Fabbri e posto em uso por Gilson Beck durante a apresentação. Esse mesmo objeto foi utilizado no instrumento AirHackTable, discutido na seção E.2.3.

A prática de *livecoding* requer o uso de instruções curtas, que incentivem o improviso musical através do código. Ambos os *livecoders* usaram bons editores de texto para a escrita dos *scripts* em *ChucK*: Vim e Emacs. Renato utilizou aspectos estruturais de alto nivel, arcos longos e estruturas melódicas e minimalistas, como os que seguem:

Interface para controle de parâmetros em tempo real

```
// Em ChucK, soltar estas duas linhas em alguma musica
// bem ritmada e deixar os padroes aa vontade com este
// arco maior:

// crescimento continuo
SinOsc s1 => Gain g => g => s1 => dac;
2 => s1.sync;
```

[14] Disponíveis em `http://vimeo.com/33012735`, `http://vimeo.com/33018740`, `http://vimeo.com/33019291`, `http://vimeo.com/33025717` e `http://vimeo.com/33025913`.

```
while(second => now);

// vento
Noise s1 => Gain g1 => g1 => SinOsc s2 => dac;
2 => s2.sync;
while(second => now);
```

Ao mesmo tempo, Vilson fez uso de scripts para facilmente especificar a posição do arquivo de áudio em que deve ser iniciada a execução, assim como a velocidade de reprodução do áudio, duração e amplitude. Uma interface simples permitiu a alteração rápida desses parâmetros em tempo de execução.

Interface para controle de parâmetros em tempo real

```
["samples/fx/s20.wav"] @=> Foo.name;
[0.] @=> Foo.prop;
[.25, .15] @=> Foo.rate;
[2., 1., 1., 4.] @=> Foo.du;
[.8] @=> Foo.gain;
```

Classe para *sampling* de arquivos de áudio

```
public class Foo {
    static string name[];
    static float prop[];
    static float rate[];
    static float du[];
    static float gain[];
}

["samples/fx/s22.wav"] @=> Foo.name;
[.0] @=> Foo.prop;
[1.] @=> Foo.rate;
[4.] @=> Foo.du;
[0.] @=> Foo.gain;

TimeGrid tg;
tg.set(1::minute/60/2, 8, 10);
tg.sync();

SndBuf buf => JCRev j => dac;
.5 => j.gain;
.2 => j.mix;

0 => int i;

while (true) {
    Foo.name[0] => buf.read;
```

```
27      Math.trunc(buf.samples()*Foo.prop[i%Foo.prop.size()]) $ int => buf.pos;
28      Foo.gain[i%Foo.gain.size()] => j.gain;
29      Foo.rate[i%Foo.rate.size()] => buf.rate;
30      tg.beat*Foo.du[i%Foo.du.size()] => now;
31      i++;
32  }
```

Estes desenvolvimentos inspiraram a criação de uma linguagem específica para *livecoding*, chamada *Vivace*, que está em constante desenvolvimento e tem seu código e maiores detalhes disponíveis online de forma livre[15]. Já foram realizadas novas apresentações utilizando essa linguagem, a notar: Hacklab do Velho, em São Carlos (SP), Semana da Comunicação FAAP (SP), Palco UFSCar (SP) e Semana Nacional da Ciência e Tecnologia (SP). Nelas houve a predominância do apelo às mídias populares, em uma reinauguração do gênero da *pop art*: trechos de vídeos de uma novela brasileira: 'Avenida Brasil' eram rearranjados em tempo real em sincronia com linhas rítmicas e melódicas[16].

Catalizadas pela aproximação de "Gera" Magela, Caleb Mascarenhas Luporini e Guilherme Lunhani as apresentações e amadurecimeentos de 2012 desembocaram na escrita coletiva de códigos (acesso e controle da platéia no código que está sendo escrito), utilização de monstros e *ascii art* e e sonoridades bizarras. Com isso, está sendo observado e proposto um gênero de *livecoding* batizado de '*Freakcoding*', possivelmente o primeiro subgênero de *livecoding*. Foi escrito um *manifesto* sobre o *freakcoding* e um artigo acadêmico. Estão sendo planejados mais apresentações e desenvolvimentos tecnológicos, todos motivados pelo *livecoding* e este subgênero brasileiro que está despontando.

[15] Veja: `http://automata.github.com/vivace`.

[16] Uma amostra está disponível *online* para ser utilizada por qualquer em qualquer computador com o navegador *Google-Chrome*: `http://pulapirata.com/skills/vivace`.

ABeatTracker (ABT)

O ABT é uma linguagem que dispara linhas rítmicas através de macros. Nestas, especifica-se as células rítmicas, amostras sonoras que são utilizadas como conteúdo sonoro destas linhas, modos de leitura destas amostras sonoras e variáveis randômicas utilizadas para execução da linha. Além disso, o ABT dispõe de variáveis globais que podem ser alteradas a qualquer momento pelo usuário também com macros, como BPM, volume, velocidade de leitura das amostras e variáveis randômicas globais (que se somam às individuais). As macros pré-estabelecidas constituem um conjunto de recursos pré-estabelecidos bem definido, o que contrasta com a ideia de uma 'linguagem de programação' que tenha capacidades mais amplas. De qualquer forma, linguagens com domínios específicos não são raras e por vezes o ABT foi descrito como uma linguagem por usuários e interessados.

O ABT é um instrumento computacional essencialmente rítmico. Junto a esta proposta, está a necessidade da utilização em conjunto com outros instrumentos, externos ao ABT, ao computador em que o ABT está sendo executado e possivelmente externo com relação a qualquer computador. Para isso foi elaborado o ABD (ABeatDetector), no qual o usuário tamborila no teclado do computador os rítmos que estiver ouvindo ou imaginando para que o ABT sincronize o pulso e utilize células rítmicas relacionadas. A análise feita pelo ABD resulta em uma série de rítmos explicitados por sequências de compassos que encapsulem durações regulares do rítmo tamborilado. Estes rítmos relacionados ao tamborilar do usuário são chamados de harmônicos e podem ser selecionados prontamente para o disparo de linhas melódicas no ABT. De fato, atualmente o ABD é parte do ABT. Todo o código do ABT está disponivel online junto à documentação para uso[17].

[17] Veja o repositório: git://labmacambira.git.sourceforge.net/labmacambira/audioArt/ABT

E.2.3 Música na matéria: EKP e AHT

Embora o foco deste trabalho seja a exploração musical através de códigos computacionais, nesta seção está uma explicação simples, clara e factual de como estas investidas transcendem o código e até mesmo a música em si. Por vezes, mais relevantes que os próprios desenvolvimentos são as mobilizações criadas nos entornos e os engajamentos. A seguir estão dois trabalhos que geraram alguma mobilização, resultando em trocas, reuniões, desenvolvimentos, pesquisas e apresentações propriamente ditas. O primeiro utiliza o estado do hardware como entrada, o segundo consiste na flutuação de origamis para a obtenção de um instrumento musical lúdico.

Emotional Kernel Panic (EKP)

Em 2008, em colaboração intensa com o CDTL (Centro de Desenvolvimento de Tecnologias Livres)[18] foi lançada a ideia de utilizar o estado do sistema operacional - especialmente o kernel linux - para gerar de sons. Surge então o '*Emotional Kernel Panic*' (EKP) na parceria de Felipe Machado (então coordenador do CDTL), Ricardo Brazileiro (artista conhecedor de tecnologias livres) e o primeiro autor do presente trabalho.

Foram reconhecidas três finalidades para esta exploração do sistema operacional:

- Pedagógica, através da utilização de um sentido que não o visual para a tarefa de analisar o sistema.
- Artística, para apresentações musicais/audiovisuais.
- Monitoramento do sistema operacional, pela emissão de sons periódicos relativos à carga de processamento, memória, uso de rede, etc.

Esta empreitada se desdobrou em apresentações na Conferência Internacional de PureData

[18] Uma associação civil formada e desmembrada em 2008, sediada em Recife, PE.

(2009), SESC (2009), e no III Festival Contato (2009). No Festival Internacional de Software Livre de 2010 foi apresentada uma pesquisa sobre o EKP com diversos patches feitos por Ricardo Brazileiro e Felipe Machado, assim como amadurecimentos da proposta. Uma implementação conceitual do EKP está aqui: `http://trac.assembla.com/audioexperiments/browser/ekp-base`.

AirHackTable

A AirHackTable (AHT) é um instrumento musical eletrônico controlado por origamis (dobraduras de papel), construída na forma de uma mesa. Nela, uma rede de *coolers* reciclados faz flutuar origamis de geometria e cores variadas. Os movimentos dos origamis são captados por *webcam* e interpretados em tempo real por software de processamento de imagens, gerando padrões que controlam a transformação sonora da música. Desta forma, pode-se dizer que os sons gerados refletem o vôo dos origamis de acordo com suas geometrias (que geram trajetórias de vôo caracteristicas). Dito de maneira mais teórica, as estruturas fora do tempo (geometrias dos origamis) estão sendo mapeadas em estruturas temporais (trajetórias dos origamis em seus vôos), o que é um macete da música erudita desde tempos antigos.

Na versão atual da AirHacktable, cada cor (vermelho, amarelo, azul, verde, preto, ou branco) controla uma voz. Já a posição do origami na mesa controla aspectos do som. Como um exemplo, pode-se configurar a AHT de forma que, se o origami está flutuando para a esquerda, o som também se move para esquerda, se o origami está mais próximo da câmera, o volume é maior, e se está mais afastando do operador, o som se torna mais agudo.

A AHT segue a filosofia de desenvolvimento contínuo, comum ao *software* livre, que atualmente foi incorporado ao movimento chamado *hardware* livre ou aberto. A concepção inicial foi de Chico Simões (mestre de maracatu e conhecedor de tecnologias livres) e o autor deste texto, através do amadurecimento de possiveis instrumentos musicais de papel. Logo neste estágio

inicial, foram aproximados Vilson Vieira, Ricardo Fabbri, Fábio Simões e Daniel Penalva em reuniões presenciais. Destas reuniões saíram ideias diversas, escritas em um *pad* aberto[19], e, por fim, a AHT em si.

A primeira realização da mesa se deu no V Festival Contato, no *Espaço Macambira*, com apresentação na Teia Casa de Criação, durante o mesmo festival. A estrutura da mesa é também de papel (papelão), e foi feita por Francisco Simões por pesquisa própria. O uso do reticulado de ventoinhas foi fruto de discussões coletivas e de experimentos do coletivo labMacambira.sourceforge.net, com especial empenho do Francisco e uso de sua luthieria de percussão: *Tora Tambores*.

Em 2012 houve a aproximação de Caleb Mascarenhas Luporini e "Gera" Magela nesta frente de atuação. Foram feitas oficinas no SESC Pinheiros e SESC Belenzinho e apresentação no AVAV[20], todas com o foco na construção e uso da AHT.

E.2.4 Música no tecido social: Sabrina Kawahara, Audioexperiments, EstudioLivre.org, CDTL, juntaDados.org, Devolts.org, MSST, LabMacambira.sf.net

A singularidade do cruzamento artístico, cultural e tecnológico dos trabalhos expostos também é desenvolvida por diferentes grupos. Os propósitos de compartilhamento e apropriação tecnológica estão no cerne destas comunidades, de forma que as investidas naturalmente tomam rumos engajados socialmente. Em especial, o empoderamento civil e a criação de um patrimônio tecnológico da humanidade são consequências imediatas das posturas de compartilhamento e apropriação citados. Segue uma lista parcial dos grupos mais relevantes para o presente trabalho, tanto pela influência que estes grupos/redes tiveram na formação e atuação do primeiro autor quanto pela ressonância que os trabalhos aqui expostos encontram.

[19] Veja: `http://pontaopad.me/origami-sensores`.
[20] Evento mensal que ocorre na cidade São Paulo: AudioVisual Ao Vivo.

- Sabrina Kawahara: formado pelo autor deste texto, Guilherme Lunhani, Guilherme Rebecchi, Israel Laurindo, Rodrigo Felício e Otávio Martigli, este grupo dedicou-se à realização de apresentações públicas com peças musicais, bem como à escrita de peças e textos através de dinâmicas coletivas.

- Audioexperiments (Æ): iniciado por Tiago Tavares, o autor do texto e aproximações de Daniel Pastore. O grupo se dedicou à publicação de textos didáticos e produção de código em repositórios públicos. Em destaque, o repositório no *Assembla* possui códigos em Python e C/C++ que realizam diversos experimentos em áudio, música e inteligência artificial[21].

- Estudios Livres: são ambientes de estudo, produção e distribuição de mídias livres. Formam, em conjunto, um movimento e uma rede social brasileira, tida como única no mundo, embora o conceito seja conhecido internacionalmente[22]. Tanto a comunidade quanto a lista operacional e plataforma da comunidade foram essenciais nos amadurecimentos de questões relacionadas às mídias livres e ao uso de Python para áudio e música, como pode ser visto na Carta Mídias Livres e no Tutorial de Python para Áudio e Música, abordados abaixo em E.3.1 deste mesmo Apêndice.

- CDTL (Centro de Desenvolvimento de Tecnologias Livres): foi uma associação civil com base em Recife/PE, dedicada ao desenvolvimento e difusão de tecnologias livres. Esta associação proporcionou suporte para os primeiros materiais didáticos e investidas em plugins de áudio através do convênio estabelecido com o Ministério da Cultura de Pontão de Cultura Digital.

- JuntaDados.org: é um grupo com membros em todo o território brasileiro e ainda ativo. Embora tenha havido um momento de maior institucionalidade, durante o convênio de

[21] Veja a página principal do espaço: `https://www.assembla.com/spaces/audioexperiments`.
[22] Há uma plataforma aberta com capacidades de *wiki*, blog e midiateca em `http://estudiolivre.org`.

Pontão de Cultura Digital com o Ministério da Cultura e parceria com a Universidade Estadual da Bahia (UNEB), o grupo é marcado pela descentralização e autonomia dos envolvidos. O autor é membro fundador e ativo do juntaDados, em conjunto, especialmente, com Fabiana "Goa" Sherine e Marcelo Soares Souza. Neste grupo foram desenvolvidas ideias como o EKP, o ABT, os tutoriais de Python para Áudio e escritos socialmente engajados.

- Devolts.org: responsável por Esporos de Pesquisa e Experimentação do programa Cultura Viva, os "Jardins Devolts" cultivam listas nacionais de trocas de conhecimentos livres e cultivo de informações voltadas para música com usos especiais de *PureData*, *Arduinos* e *Python*. Este é um meio de troca rico e alternativo a alguns meios mais politizados, como a lista do *estudiolivre.org*.

- MSST (Movimento dos Sem Satélite): com um manifesto e propostas coerentes, mas nada óbvias, como a demanda específica por satélites civis, abordagens tecnomágicas, o grupo é uma instância iniciática em termos tecnológicos e conceituais.

- LabMacambira.sf.net: espaço virtual de programadores dedicados ao audiovisual e propósitos socialmente engajados. Fundado em parceria do autor deste trabalho e de Vilson Vieira, Daniel Marostegan e Carneiro e Ricardo Fabbri e de parte dos investimentos da segunda etapa do convênio de Pontão de Cultura Digital Nós Digitais, da Teia Casa de Criação. Em um primeiro momento, membros foram remunerados como mentores e como aprendizes[23]. Em um segundo momento, aproximações e parcerias diversas ocorreram. Detalhes estão mas subseções seguintes de *Web* e materiais didáticos.

- Submidialogia: grupo bastante relacionado aos encontros de '*submidialogias*, ao Descen-

[23] Renato Fabbri, Ricardo Fabbri, Vilson Vieira, Marcos Mendonça e Gilson Beck mentoraram; Alexandre "Bzum" Negrão, Lucas Zambianchi, Larissa Arruda, Nivaldo Henrique Bondança, Fernando Gorodscy, Andres Martano desenvolveram como bolsistas; Danilo Shiga e Marcos Murad foram parceiros colaboradores; Francisco Simões, Fábio Simões, Caleb Mascarenhas Luporini, "Gera" Magela, Edson "Presto" Correia, Daniel Penalva se aproximaram para atividades específicas, cruciais em todo ano de 2012 já sem os investimentos do convênio.

tro e às ideias de submídia. A lista de emails do grupo possui membros de diversos dos outros grupos citados.

- Outros grupos relacionados: o MuSa foi um grupo de Joinville importante neste trabalho por ter sido fundado por Vilson Vieira, que possui presença forte no LabMacambira.sf.net e nos trabalhos aqui descritos. A Metareciclagem é uma rede engajada em apropriação tecnológica e empoderamento civil com bastante presença dos outros grupos aqui citados e muito movimentada. A lista de emails ligada ao movimento Transparência Hacker foi também de importância central como espaço focado em questões de transparência e dados abertos. A Casa de Cultura Tainã é uma entidade cultural de Campinas/SP, referência no uso de tecnologias livres e atividades socialmente engajadas, e onde muito do que está aqui descrito foi inspirado. O Hacklab do Velho foi fundado por pessoas ligadas ao labMacambira.sf.net e por moradores do alojamento velho da USP, campus de São Carlos, e acolheu diversas experimentações e articulações. Outros grupos com alguma importância foram a Casa Fora do Eixo de São Carlos, a Teia Casa de Criação, a Nuvem estação rural de experimentação tecnológica e o grupo que organiza o AVAV Áudiovisual Ao Vivo, ligado ao Epicentro Cultural.

E.3 Materiais didáticos

E.3.1 Tutoriais em texto e código: python, filtros e nyquist, plugins lv2, metrics, carta mídias livres, contra-cultura digital

- **Tutorial de python para áudio e som**

  Este tutorial foi apresentado parcialmente em Berlim no LAC 2007 e, desde então, foi melhorado algumas vezes. Esta primeira versão ficou resumida em forma de texto na plataforma Estúdio Livre[24]. Um agradecimento especial para Fábio Furlanete e Marília

[24] `http://estudiolivre.org/python-e-som-tutorial`.

Chiozo pelas contribuições. Em 2010 a Associacao Python Brasil escolheu este trabalho, então já mais maduro, para ser apresentado no Festival Internacional de *Software* Livre, em Porto Alegre. Como consequência, foi feita uma série de video-tutoriais[25]. Os vídeos são tidos como iniciáticos em Python por diferentes pessoas ligadas ao movimento de *software* livre.

- **Tutoriais de filtros e amostragem via python**

  Voltados para explicitar fundamentos de áudio digital, estes tutoriais são baseados em pequenos *scripts* escritos em Python que exploram conceitos puntuais. Pequenas explicações são dadas com o intuito de orientar a exploração inteligente destes *snippets*. *Teorema de Amostragem*: estes *scripts* executam experimentações ilustrativas com o Teorema de Nyquist e figuras. *Filtros*: além de filtros FIR e IIR simples, duas utilizações clássicas destes filtros estão implementadas de forma didática: Wavelets (FIR) e Quad (IIR). Estes códigos didáticos podem ser baixados do repositório *audioArt*[26].

- **Tutorial de plugins lv2**

  Dadas as dificuldades que o desenvolvimento dos *plugins* de áudio apresenta, foi feito um tutorial passo a passo com *plugins* operantes em todas as etapas. Os códigos e os textos estão todos em repositório[27].

- **Microtutoriais Django**[28]

  Estes 'microtutoriais' são baseados nos conceitos de *scripts mínimos* e *alterações puntuais*. O primeiro conjunto de microtutoriais é dedicado a reconstruir o tutorial oficial do django de forma condensada e não prolixa. O segundo destes conjuntos é dedicado a instrumentalizar de fato o leitor com o entendimento do funcionamento dos princípios funda-

[25] `http://estudiolivre.org/tiki-index.php?page=Video+Tutoriais`.
[26] Veja `http://labmacambira.git.sourceforge.net/git/gitweb.cgi?p=labmacambira/audioArt;`.
[27] Veja `http://labmacambira.git.sourceforge.net/git/gitweb.cgi?p=labmacambira/lv2Tut;`.
[28] Veja: `https://vimeo.com/channels/labmacambira`.

mentais deste *framework*. A importância deste material reside no fato de ser uma das primeiras incidências dos *scripts* mínimos, muito usados no labMacambira.sourceforge.net para passagens de tecnologias de forma precisa.

- **Figusdevpack (FDP)**

  Idealizado como um meio de interação da comunidade de Python e Música para compartilhamento de *scripts* inteiros e excertos, o FDP foi Baseado principalmente em documentação organizada sobre as práticas e as bibliotecas existentes para Python. Parte desta documentação é proposta como *scripts* que fazem uso de objetos e módulos de forma isolada, puntual. Este trabalho foi aceito na maior conferência de áudio em linux, a Linux Audio Conference de 2008 (LAC2008) e foi reativado algumas vezes pelo autor deste texto em conjunto com Vilson Vieira, Ivan Marin e outros desenvolvedores. Este projeto está documentado na plataforma do Estúdio Livre[29].

- **Carta mídias livres**

  Texto criado em decorrência da participação do autor do presente escrito na comissão de seleção no "Prêmio Mídias Livres", a convite do Ministério da Cultura por "notório saber". A participação consistiu em avaliar os inscritos no Prêmio Mídias Livres e distribuir 4 milhões de reais dentre categorias regionais e nacionais. Esta carta é um documento único, deixando às claras o conceito de Mídias Livres como não aprisionadas pelo conceito de propriedade, ou seja, que priorizam a sua livre circulação e a possibilidade de geração de materiais derivados. Há o viés de priorizar processos colaborativos, comunitários e setores da sociedade menos contemplados na geração e circulação midiática[30].

- **Philosometrics**

[29] Veja a página do PDF `http://estudiolivre.org/tiki-index.php?page=fdp&highlight=fdp`, o artigo aceito no LAC2008 `http://www.estudiolivre.org/el-gallery_view.php?arquivoId=8221` e o repositório do FDP `http://sourceforge.net/projects/fdpack/developfdpsf`.

[30] Veja: `http://www.estudiolivre.org/carta-pts-midias-livres`

Em decorrência deste trabalho, surgiu o Musimetrics, o Cinemetrics e o Literametrics. Uma publicação no *Journal of Statistical Mechanics: Theory and Experiment* condensa e compara as análises de filosofia e música.(69) Além disso, é uma utilização das ciências duras para assunto mais incidente em ciências humanas.

- **Textos socialmente engajados**

  O uso de pseudônimos é um costume apreciado em diversos meios. As pesquisas informais confirmam vantagens desta prática[31]. Em especial utiliza-se pseudônimos para auxiliar a despersonificação, gerando textos menos presos à satisfação da auto-imagem. Como resultado, além de aumento de produtividade, constuma-se conseguir também compreensões diferentes. Destes textos, dois foram publicados em duas publicações relevantes: O Contracultura Digital e o Peixe Morto, este último relacionado aos submidialogias[32].

E.3.2 *Screencasts* e outros materiais em video

- **Python para áudio e música:** Relacionado ao tutorial em texto citado acima, foram feitos videos sobre a utilização da linguagem Python, em grande parte através de usos em *IPython*, para leitura e escrita de arquivos de áudio com bibliotecas padrão e externas, assim como uso básico de wavelets,OSC e outros recursos.

- **Canal Macambira no Vimeo** Atualmente, são mais de 700 videos sobre sessões de *hacking* e experiências de arte e tecnologia. Feitos pelo LabMacambira.sourceforge.net, o canal tem videos do autor deste escrito, e de: Larissa Arruda, Alexandre Negrão, Lucas Zambianchi, Andres Martano, Fernando Gorodscki, Chico Simões, Daniel Marcicano, Daniel Pizetta, Daniel Penalva, Nivaldo Bondança, Danilo Shiga, Marcos Murad, Ricardo Fabbri e Vilson Vieira[33]. Exemplos especiais dos videos disponibilizados no canal

[31] Veja: `http://disqus.com/research/pseudonyms/`

[32] Veja: `http://mutgamb.org/blog/Submidialogias-Peixe-Morto-para-Baixar` e `http://culturadigital.br/contraculturadigital/2012/02/01/publicacao-contraculturadigital/`.

[33] O canal pode ser acessado em: `http://vimeo.com/channels/labmacambira`.

são:

- *Livecoding*: foram expostos os processos utilizados na apresentação de *livecoding* com ChucK, vacas e sono REM[34].
- Raspagem de dados: menos artístico e mais engajado, é uma breve explicação sobre o processo de raspagem de dados em páginas HTML[35].
- Outros videos incluem gambiarras artísticas, uso de scripts/ferramentas para audio-visual, etc.

E.4 Web

Esta dissertação é focada em uma descrição física do áudio digital enquanto música e usos em código desta abordagem. Como pode-se observar acima, neste mesmo Apêndice, este trabalho, disponibilizado e concebido como tecnologias livres, tem dimensões bastante consideráveis em aspectos socialmente engajados e artísticos. Esta seção exibe os feitos que extrapolam a música por completo.

E.4.1 Tecnologias sociais: Sítios, Conteúdos e Articulação

Sítios

Alguns *sites* foram criados, como parte de trabalhos engajados e tentativa de manutenção das estruturas do labMacambira.sourceforge.net. O primeiro site com alguma relevância seja talvez o site para a Casa dos Meninos / LIDAS, em que foi feita uma estrutura para acolher as conferências setoriais e municipais dos direitos da criança e do adolescente da cidade de São Paulo. Outros exemplos, mais recentes, são as páginas para a Rede Nacional de Cultura Ambiental

[34] Veja: `https://vimeo.com/33012735`, `https://vimeo.com/33018740`, `https://vimeo.com/33019291`, `https://vimeo.com/33025717`, `https://vimeo.com/33025913`.
[35] Veja: `vimeo.com/channels/labmacambira/2681879`.

Afrobrasileira e os blogs para a Rio+20 das secretarias do Ministério da Cultura.

Nest meio tempo também foram criadas várias ferramentas *web*, que rodam em navegadores comuns, como o *Firefox*. Destes, vale citar[36]:

- SOS: sistema para coleta e difusão de conhecimentos populares e étnicos ligados à saúde.
- Ágora Communs, Novo Ágora: sistema de deliberações *online*.
- Economia Criativa: rascunho de sistema para facilitar circulação de bens me 4 eixos: venda de produtos, aluguél de equipamentos, aluguél de lugares e prestação de serviços.
- Catálogo de Ideias: ferramenta simples para catálogo de ideias sobre temas diferentes, junto a palestras, etc.
- Mapeamento dos 301 pontos de SP: mapeamento dos 301 pontos de cultura estaduais.
- Mapas Coletivos: ferramenta para criação e publicação de mapas com possibilidades colaborativas. Bastante usada por pessoas de meios diferentes pela usabilidade e proposta pertinente.
- Leitura de *shapefiles* no Mapas de Vista: implementação de leitura de arquivos do tipo *shapefile* com dados geográficos em um plugin e tema de *Wordpress*.
- Maper: atualmente, um rascunho de uma ferramenta de publicação de Mapas com Vistas a publicações no facebook e outros formatos.

[36] Os arquivos fonte estão nos repositórios em `http://labmacambira.git.sourceforge.net/git/gitweb-index.cgi`.

Conteúdos

Os conteúdos estão em diversas páginas da *wiki* do Nós Digitais e do Estúdio Livre.[37] Tratam principamente de documentações de *software* e receitas para acesso a repositórios ou instalações. Nestas documentações também estão apontamentos de recursos artísticos e apresentações realizadas com a AHT, *livecoding* e outras propostas. Chama a atenção a quantidade de instâncias de *etherpads* usadas, o labMacambira.sf.net chega a criar vários destes epads por dia, para escrita simultânea a várias mãos[38].

Articulação, disponibilização e desenvolvimento conjunto

As articulações se dão principalmente através do IRC, do AA e de emails, além de conversas presenciais. O IRC é uma forma de comunicação leve e focada na comunicação coletiva. O AA está explicado abaixo. Não menos importante são os *Etherpads* e as páginas *wiki* criadas, permitindo proposições e escritas rápidas por diferentes pessoas.

AA

Uma das tecnologias desenvolvidas nesta empreitada se propõe a estabelecer formas de compartilhar andamentos e processos. As motivações incluem transparência civil e organização assíncrona de grupos descentralizados. O sistema se baseia no envio de mensagens periódicas sobre as atividades que estão sendo desempenhadas, evitando foco em produtos e propaganda. A ferramenta pode ser usada no IRC ou em linha de comando. Existem algumas interfaces *web* para exibir as mensagens e dados do AA[39]. Estão planejados desenvolvimentos do AA para que seja utilizável através do *chat* em diferentes redes sociais. Também está sendo considerada a utilização do AA para distribuição de verba de acordo com as dedicações comprovadas pelos

[37] Veja: `http://wiki.nosdigitais.teia.org.br/Lab_Macambira`, `http://www.estudiolivre.org/el-user.php?view_user=gk`.
[38] Veja: `http://pontaopad.me/epads`
[39] Veja: `http://wiki.nosdigitais.teia.org.br/AA`.

mecanismos do AA, tornando possível que se trabalhe de forma remunerada pelo bem comum.

E.5 Momento atual e previsões

Está sendo escrito sobre *livecoding*, especialmente ligado ao Vivace e ao subgênero *Freakcoding*, que recebeu um manifesto recentemente. Um artigo explicando os vínculos entre a proposta estética e o programa Vivace está sendo escrito por várias mãos. O georeferenciamento, perene na atuação do labMacambira.sf.net, tomou agora a direção de arrumar animações de fluxos em arestas de grafos dispostos em mapas. Apresentações artísticas estão previstas para uso da AHT e experimentações de *livecoding*. Um *crowdfunding* foi feito para experimentação de difusão na rede e respostas, com estudos em andamento sobre comportamentos no *Facebook* e listas de *emails*. O Cultura Viva, programa do governo federal que apoiou parte destas iniciativas, indica fortalecimento, tornando-se política de estado, não somente do governo que a implantou. As aproximações federais são observadas também por parte do MEC (Ministério da Educação), com a prestação de uma acessoria em 2012, e do MinC, devido à Lei Cultura Viva. Comunicações mais moderadas continuam com o MCTi, MC e MMA. Os canais de IRC e emails do labMacambira.sf.net estão bastante ativos e focados em execução de tarefas, e, dentre os desenvolvimentos em código previstos, está o aproveitamento desta dissertação para implementações em JavaScript, experimentações, apresentações, publicações, usos didáticos e estudo propriamente dito. O alto rendimento do labmacambira.sourceforge.net no GSoC de 2012, e o forte empenho de alunos do IPRJ/UERJ, de Nova Friburgo, através do suporte oferecido pelo prof. Ricardo Fabbri, da mesma instituição, apontam para uma vida própria que estes andamentos tomaram. Em São Paulo, a atuação de Geraldo Magela e Caleb Luporini se propõe a difundir estas práticas em circuitos artísticos e de ensino. Por emails privados, listas ou IRC, os participantes do grupo estabelecem trocas com pessoas em diversas localidades no Brasil e no mundo para fins de desenvolvimento tecnológico, trabalhos, elaborações didáticas e convivên-

cias, mesmo que brandas, por protocolos de comunicação. A proposta no momento é focar em algo que contemple estas coisas todas da melhor forma, aliando cobertura dos interesses e precisão no aprofundamento, o que está despontado nos estudos de *logs* de IRC e listas de *emails*, com bancos de dados massivos e públicos. As possibilidades artísticas, engajadas, científicas e de instrução e interesse pessoal são confluentes através da modelagem por redes complexas, o que sugere a unificação do conhecimento, traço marcante da área e observado com frequência.