UNIVERSIDADE FEDERAL DE SÃO CARLOS — UFSCAR
Campus Sorocaba

Carlos Moraes de Freitas Junior

REDUÇÃO DE RUÍDOS NA MÚSICA *ASH TRAY BLUES*: UM ESTUDO SOBRE PROCESSAMENTO DIGITAL DE SINAIS COM FOCO EM TRANSFORMADAS DE FOURIER

Sorocaba
Setembro – 2019

Carlos Moraes de Freitas Junior

REDUÇÃO DE RUÍDOS NA MÚSICA *ASH TRAY BLUES*: UM ESTUDO SOBRE PROCESSAMENTO DIGITAL DE SINAIS COM FOCO EM TRANSFORMADAS DE FOURIER

UNIVERSIDADE FEDERAL DE SÃO CARLOS — UFSCAR
Campus Sorocaba

Orientador: Prof. Dr. Renato Fernandes Cantão

Sorocaba
Setembro – 2019

Moraes de Freitas Junior, Carlos

Redução de Ruídos na Música "Ash Tray Blues" Através do Método Noise Gate: Um Estudo Sobre Processamento Digital de Sinais com Foco em Transformadas de Fourier / Carlos Moraes de Freitas Junior. -- 2019.
76 f. : 30 cm.

Trabalho de Conclusão de Curso (graduação)-Universidade Federal de São Carlos, campus Sorocaba, Sorocaba
Orientador: Prof. Dr. Renato Fernandes Cantão
Banca examinadora: Prof. Dr. Antonio Luís Venezuela, Prof.ª Dra. Maria Lúcia Pereira Antunes
Bibliografia

1. Processamento Digital de Sinais. 2. Filtros digitais. 3. Transformadas de Fourier. I. Orientador. II. Universidade Federal de São Carlos. III. Título.

Ficha catalográfica elaborada pelo Programa de Geração Automática da Secretaria Geral de Informática (SIn).
DADOS FORNECIDOS PELO(A) AUTOR(A)
Bibliotecário(a) Responsável: Maria Aparecida de Lourdes Mariano – CRB/8 6979

FOLHA DE APROVAÇÃO

CARLOS MORAES DE FREITAS JUNIOR

REDUÇÃO DE RUÍDOS NA MÚSICA ASH TRAY BLUES:
UM ESTUDO SOBRE PROCESSAMENTO DIGITAL DE SINAIS COM FOCO EM TRANSFORMADAS DE FOURIER

Trabalho de Conclusão de Curso

Universidade Federal de São Carlos – *Campus* Sorocaba

Sorocaba, 13/09/2019.

Orientador _____
Prof. Dr. Renato Fernandes Cantão

Membro _____
Prof. Dr. Antonio Luís Venezuela

Membro _____
Prof.ª Dr.ª Maria Lúcia Pereira Antunes

Este trabalho é dedicado à minha mãe Rita, que desde o início de minha vida esteve ao meu lado, fornecendo todo o necessário para meu sustento e educação.

Agradecimentos

Agradeço primeiramente ao Deus Triúno em quem deposito minha fé.

Agradeço imensamente ao Prof. Dr. Renato Fernandes Cantão por aceitar o convite de ser meu orientador e pela atenção despendida durante todo o período de elaboração do presente trabalho, desde as primeiras ideias até a defesa diante da banca examinadora.

Agradeço à Universidade Federal de São Carlos por oferecer cursos de qualidade e de suma importância para a sociedade. Aos professores que fazem parte do corpo docente do curso de Licenciatura em Matemática e que dedicam-se avidamente à formação de professores e pesquisadores, o meu muito obrigado.

Meus agradecimentos aos familiares que acompanharam toda minha trajetória, em especial, agradeço à minha noiva Mariana, por me levantar todas as vezes que me deixei cair, por acreditar em mim e por também sonhar os meus sonhos.

Por fim, agradeço aos meus colegas de curso, que mais do que ninguém sabem dos obstáculos que tivemos que superar na graduação. Mesmo por vezes estando separados pela grade horária, conquistamos tudo isso juntos.

‘‘A esperança tem asas.
Faz a alma voar.
Canta a melodia mesmo sem saber a letra.
E nunca desiste – nunca.”
(Emily Dickinson)

Resumo

Este trabalho apresenta um estudo sobre processamento digital de sinais, análise de Fourier e filtros digitais. O tema é de grande importância pois está presente no cotidiano de grande parte da população, sendo usado em TVs, *smartphones*, rádios e redes telefônicas. O objetivo central é reduzir o ruído presente na música *Ash Tray Blues*, gravada ao final da década de 1920. Para tanto, foram realizados estudos acerca do processamento de sinais, com foco nas Transformadas de Fourier e seu uso na construção de filtros digitais, em especial, o método *noise gate*, utilizado como base para o desenvolvimento do algoritmo *noise reduce*, de Tim Sainburg. Com o auxilio do software Audacity®, obteve-se uma faixa contendo o ruído da música de forma isolada. O algoritmo foi implementado com a utilização da linguagem de programação Python e contou com pacotes como NumPy e Librosa. Da implementação do algoritmo resultou um arquivo contendo a música com ruído reduzido e espectrogramas que ilustram o processo de filtragem do sinal. Ao final da aplicação, obteve-se um resultado positivo e observou-se uma melhora na qualidade do som em relação à faixa original. Pôde-se concluir que é possível realizar a redução de ruído em gravações antigas, mesmo com um curto intervalo de ruído isolado.

Palavras-chave: Processamento de sinais. Análise de Fourier. Filtros digitais. *Noise gate*. Redução de ruído.

Abstract

This work presents a study about digital signal processing, Fourier analysis and digital filters. These subjects are extremely relevant, given their ubiquity in the daily life of most of the population, being present on TVs, smartphones, radios and phone networks. The main objective is to reduce the noise present in the song *Ash Tray Blues*, recorded at the end of the 1920s. Studies on signal processing were carried with focus on Fourier Transforms and their use in the design of digital filters. Of particular interest is the *noise gate method*, used as a basis for the development of Tim Sainburg's *noise reduce* algorithm. With the aid of Audacity® software, a track containing the isolated music noise was obtained. The algorithm was implemented using the Python programming language with the help of packages like NumPy and Librosa. The implementation resulted in a file containing the noise-reduced music and spectrograms that illustrate the signal filtering process. In the end a positive result was obtained and an improvement in sound quality over the original music was observed. It can be concluded that it is possible to perform noise reduction on older recordings even with a short sample of the original noise.

Keywords: Signal processing. Fourier analysis. Digital filters. Noise gate. Noise reduction.

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e siglas

m	Metro – unidade de medida de comprimento
s	Segundo – unidade de medida de tempo
u	Unidade de medida sem especificação
Hz	Hertz – Unidade de medida de frequência
dB	Decibel – unidade de medida de magnitude (nível de som)
DFT	Discrete Fourier Transform (Transformada Discreta de Fourier)
IDFT	Inverse Discrete Fourier Transform (Transformada Discreta de Fourier Inversa)
FFT	Fast Fourier Transform (Transformada Rápida de Fourier)
STFT	Short-Time Fourier Transform (Transformada de Fourier de Tempo Curto)
ISTFT	Inverse Short-Time Fourier Transform (Transformada de Fourier de Tempo Curto Inversa)
FIR	Finite Impulse Response (Resposta ao Impulso Finita)
IIR	Infinite Impulse Response (Resposta ao Impulso Infinita)
SNR	Signal-Noise Ratio (Relação Sinal-Ruído)
CD	Compact Disc (Disco Compacto)
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
BOVESPA	Bolsa de Valores de São Paulo

Sumário

1 Introdução

Define-se sinal como todo processo físico que ocorre no mundo real em função do tempo. Um sinal pode ser analógico, ou seja, contínuo durante o tempo ou digital, ou seja, o sinal é quantizado em instantes discretos de tempo (LYONS, 2011).

O processamento de sinais é a ciência que estuda, interpreta, analisa, modifica e aprimora esses processos através do tratamento analógico ou digital de suas informações (dados). Exemplos deste tema tornaram-se cada vez mais presentes no decorrer da história, tendo início há aproximadamente quatro mil e quinhentos anos, no Egito. Os egípcios tinham a necessidade de prever as cheias do Rio Nilo. Esta necessidade vinha do fato das margens do Nilo serem férteis e ali terem se juntado muitos grupos em busca de boas condições para o plantio e a subsistência. Diante desta necessidade, o povo egípcio criou um mecanismo de registro de dados para prever os períodos de cheia do Nilo e, consequentemente, a fertilidade do solo. Conhecido como *Pedra de Palermo*, o mecanismo funcionava como uma tabela de dados, registrando o nível de água do Nilo e o nome do Faraó em cada período anual. Toda lógica ali empregada remete ao processamento de sinais, pois eles colheram sinais analógicos (nível do Rio Nilo), os interpretaram e propuseram uma expectativa futura, fundamentada nos dados analisados (VETTERLI; PRANDONI, 2008).

Hoje, a presença do processamento digital — e analógico — de sinais pode ser notada, por exemplo, logo ao ser despertado por um aplicativo de alarme instalado em um *smartphone*, que reproduz um arquivo de áudio que foi gravado digital ou analogicamente, codificado, quantizado, analisado, filtrado, decodificado, normalizado e guardado em algum repositório digital do *smartphone*.

Dentro do processamento digital de sinais, a redução de ruídos mostra-se como uma preocupação recorrente em diferentes aplicações, em especial, a redução de ruídos em arquivos digitais de áudio.

O estudo do tema aqui apresentado tem grande relevância para a indústria musical, bem como na própria análise de sinais, como em reconhecimento de voz, utilizado em investigações policiais ou redução de ruídos causados por maquinários na área médica, por exemplo.

Neste contexto, o objetivo central deste trabalho foi reduzir o ruído em uma música gravada na década de 1920 e dimensionar e avaliar a relação entre custo e benefício deste procedimento. Para atingir este objetivo central, os seguintes objetivos específicos devem ser listados:

- Apresentar o histórico dos estudos sobre funções periódicas, séries de Fourier, trans-

formadas de Fourier e filtros digitais;

- Analisar métodos de redução de ruído em arquivos digitais de áudio;
- Implementar um algoritmo que atenue a amplitude de frequências indesejáveis;
- Obter diferentes testes com os parâmetros do algoritmo implementado;
- Avaliar os resultados obtidos, observando a qualidade do som filtrado.

A pesquisa foi desenvolvida em um período de seis meses. As atividades aqui elencadas buscaram atingir os objetivos desejados, compondo a metodologia deste trabalho. Foram utilizados recursos bibliográficos e computacionais para a implementação do algoritmo que foi utilizado para a realização da redução de ruídos na canção selecionada.

A primeira atividade foi a pesquisa bibliográfica acerca dos assuntos que são abordados neste texto e que fizeram parte da execução do projeto de redução de ruídos. Essa atividade foi realizada através de pesquisa em acervos digitais, disponibilizados pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior — CAPES e por meio de buscas em sites relacionados.

Em seguida, foram pesquisados métodos de filtragem de sinais já existentes. Utilizou-se os mesmos recursos da primeira atividade. Esta fase serviu como embasamento para o produto final pretendido, que foi um algoritmo de redução de ruídos em áudio digital, utilizando as transformadas de Fourier e o método *Noise Gate*. Como consequência desta atividade, foram gerados arquivos de áudio com o som filtrado.

Como terceira atividade, foi analisado o software *Audacity*[(1)], em especial a ferramenta *noise reduction* (redução de ruído) e um algoritmo criado por (SAINBURG, 2018) — utilizando a linguagem de programação *Python*[(2)] —, utilizado neste trabalho. O algoritmo foi escolhido com o intuito de receber um sinal de áudio ruidoso como entrada e, por meio do método *noise gating*, atenuar os níveis de ganho de frequências indesejáveis, reduzindo assim o ruído existente no sinal original. O software *Audacity*®, também serviu como uma potente ferramenta de comparação e auxílio para cortes de faixas da música analisada.

A música analisada, chamada *Ash Tray Blues*, foi comprada através da loja virtual *Document Records*[(3)] por US$ 0,99.

(1) O software Audacity®, copyright © 1999-2019 *Audacity Team*. Site: `https://audacityteam.org/`, é um software livre, distribuído sob os termos da Licença Pública Geral GNU. O nome Audacity® é uma marca registrada de Dominic Mazzoni.

(2) Python Software Foudation, Beaverton, Oregon, Estados Unidos.

(3) `https://thedocumentrecordsstore.com/product/DOCD-5088/`.

Em seguida, foram executados testes de desempenho no algoritmo, modificando numericamente os parâmetros de entrada e verificando seus efeitos. Com esta atividade, foi possível dimensionar a relação entre custo e benefício da redução de ruídos em arquivos digitais de áudio.

Por fim, avaliou-se a qualidade do áudio gerado pela utilização do algoritmo implementado. Esse procedimento foi realizado computacionalmente, através de testes em hardwares reprodutores de áudio.

Na Seção 2 é apresentada a análise de Fourier, dando um olhar geral acerca do trabalho deixado por Jean Baptiste Joseph Fourier e sua relação com a música e o processamento de sinais.

A Seção 3 conta com um estudo acerca de alguns tipos de filtros digitais disponíveis dentro do processamento de sinais e que utilizam as transformadas de Fourier em sua arquitetura.

A aplicação que faz parte do objetivo geral do trabalho está na Seção 4, apresentando o funcionamento do algoritmo utilizado, sua implementação e os espectrogramas produzidos.

Na Seção 5 encontram-se os resultados obtidos através da aplicação realizada, apresentando uma quantização da redução de ruído obtida.

Na Seção 6 são apresentadas as considerações finais acerca do trabalho realizado e as pesquisas futuras.

2 Análise de Fourier

Este trabalho tem sua fundamentação teórica baseada no trabalho do matemático e físico francês Jean Baptiste Joseph Fourier (1768-1830). Ele foi responsável por um importante estudo sobre condução de calor, tendo sua principal obra publicada em 1822, *"Théorie Analytique de la Chaleur"* (FOURIER, 1822). Como exemplo dos resultados de suas pesquisas, credita-se a Fourier o descobrimento do efeito estufa. Ele deixou não somente um legado nesta área, mas também contribuiu para o crescimento do processamento de sinais, contando com a colaboração de outros pesquisadores para a manutenção e difusão de suas pesquisas.

Seus estudos sobre calor e funções periódicas resultaram no que hoje denominamos *séries de Fourier*, cuja finalidade é descrever funções periódicas por meio de aproximações, utilizando somas de funções trigonométricas (senos e cossenos) (SANTOS, 2004).

Para dar início ao estudo sobre séries de Fourier, é importante ter alguns conceitos apre-

sentados previamente, como por exemplo os de funções trigonométricas, apresentados a seguir.

2.1 Funções periódicas

As funções periódicas, como o próprio nome sugere, estão relacionadas a um certo *período*, que pode ser considerado comprimento (m) ou tempo (s), dependendo da aplicação (SANTOS, 2004). Tomando $T \in \mathbb{R}$ como o período da função periódica $g : \mathbb{R} \to \mathbb{R}$, tem-se

$$g(x) = g(x + T). \tag{1}$$

Da Equação (1), segue que a cada intervalo nT do domínio — com n sendo um número inteiro — a imagem $g(x + nT)$ será igual a imagem $g(x)$.

Sendo o período T o intervalo de tempo ou distância entre cada ciclo completo da função, seu conceito inverso, ou seja, a quantidade de ciclos completos da função por unidade de tempo ou comprimento denomina-se *frequência* e define-se como

$$f = \frac{1}{T}. \tag{2}$$

A frequência pode ser medida em ciclos por metro ou ciclos por segundo, dependendo da aplicação. Em especial, se a frequência for medida em ciclos por segundo, normalmente utiliza-se a unidade Hertz (Hz).

As funções trigonométricas valem-se da *frequência angular*, sendo empregadas nos estudos sobre séries de Fourier. A frequência angular é dada por

$$w_0 = \frac{2\pi}{T}. \tag{3}$$

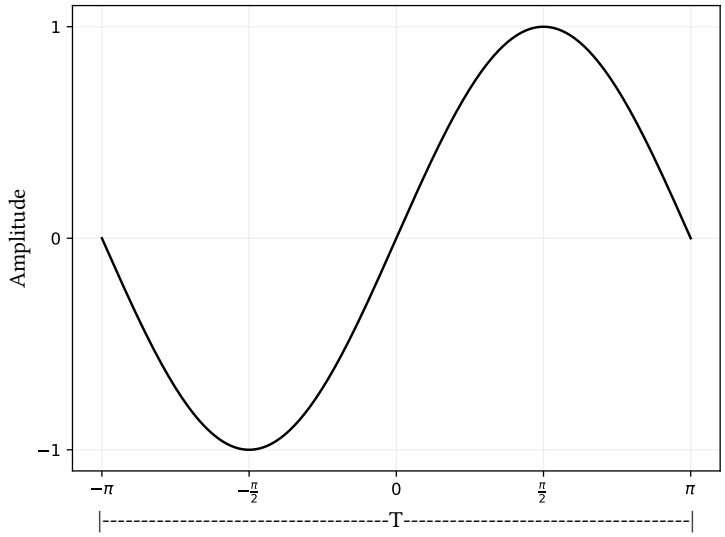
A Figura 1 traz um exemplo de função periódica, o que permite voltar a atenção à soma de funções trigonométricas, conhecidas como séries trigonométricas, em particular, as séries de Fourier. Os gráficos elaborados pelo autor foram construídos com o auxílio do pacote *Matplotlib* (HUNTER, 2007), implementado em Python.

2.2 Séries de Fourier

Seja a série trigonométrica

$$\frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2n\pi x}{T}\right) + b_n \operatorname{sen}\left(\frac{2n\pi x}{T}\right) \right]. \tag{4}$$

Figura 1 – Função trigonométrica $f(x) = \operatorname{sen} x$, de período $T = 2\pi$.



Fonte: o autor.

Pela Equação (4) pode-se perceber que as infinitas funções trigonométricas são periódicas de período T. Respeitadas algumas condições, essa série define uma função periódica f, também de período T, sendo garantida pelo Teorema (1).

Teorema 1 (Teorema da Convergência de Fourier) *Seja f uma função periódica, de período T e seccionalmente contínua em $\left[-\frac{T}{2}, \frac{T}{2}\right]$, então a série de Fourier converge para todo x real, tal que*

$$\frac{a_0}{2} + \sum_{n=1}^{\infty}\left[a_n \cos\left(\frac{2n\pi x}{T}\right) + b_n \operatorname{sen}\left(\frac{2n\pi x}{T}\right)\right] = \begin{cases} f(x), & \text{se } f \text{ é contínua em } x \\ \dfrac{f(x_+) + f(x_-)}{2}, & \text{se } f \text{ é descontínua em } x \end{cases} \tag{5}$$

sendo $f(x_+)$ e $f(x_-)$ os limites laterais de f em x.

A demonstração do teorema foge do escopo do projeto, mas pode ser encontrada em (FIGUEIREDO, 2000).

A Equação (5) abre uma outra discussão que é exatamente a principal característica das séries de Fourier: a aproximação de funções periódicas e seccionalmente contínuas quaisquer por infinitas somas de funções trigonométricas de mesmo período (SANTOS, 2004, p. 17). Para tanto, deve-se determinar os coeficientes a_0, a_n e b_n da série de Fourier a partir da função estudada.

2.2.1 Determinação dos coeficientes da série de Fourier

A determinação dos coeficientes da série de Fourier é aplicável para qualquer função periódica que respeite as condições impostas pelo Teorema da Convergência de Fourier. Tomando a frequência angular fundamental $w_0 = \frac{2\pi}{T}$, pode-se reescrever a série de Fourier, de modo que

$$f(x) \approx \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos(n w_0 x) + b_n \operatorname{sen}(n w_0 x) \right]. \tag{6}$$

Integrando ambos os lados no intervalo $[0, T]$, obtém-se

$$\begin{aligned}
\int_0^T f(x)\,dx &= \int_0^T \left\{ \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos(n w_0 x) + b_n \operatorname{sen}(n w_0 x) \right] \right\} dx \\
&= \int_0^T \frac{a_0}{2}\,dx + \sum_{n=1}^{\infty} \left[\int_0^T a_n \cos(n w_0 x)\,dx + \int_0^T b_n \operatorname{sen}(n w_0 x)\,dx \right] \\
&= \frac{a_0}{2} x \Big|_0^T + \sum_{n=1}^{\infty} \left[\frac{a_n}{n w_0} \operatorname{sen}(n w_0 x) \Big|_0^T - \frac{b_n}{n w_0} \cos(n w_0 x) \Big|_0^T \right] \\
&= \frac{a_0}{2} T + \sum_{n=1}^{\infty} \left\{ \frac{a_n}{n w_0} \left[\operatorname{sen}(n w_0 T) - \operatorname{sen} 0 \right] - \frac{b_n}{n w_0} \left[\cos(n w_0 T) - \cos 0 \right] \right\} \\
&= \frac{a_0}{2} T + \sum_{n=1}^{\infty} \left\{ \frac{a_n}{n w_0} \operatorname{sen}(2\pi n) - \frac{b_n}{n w_0} \left[\cos(2\pi n) - 1 \right] \right\}.
\end{aligned}$$

Mas,

$$\operatorname{sen}(\pi n) = 0 \quad \text{e} \quad \cos(2\pi n) = 1, \quad \forall n \in \mathbb{Z}.$$

Então:

$$\begin{aligned}
\int_0^T f(x)\,dx &= \frac{a_0}{2} T + \sum_{n=1}^{\infty} \left[\frac{a_n}{n w_0} (0) - \frac{b_n}{n w_0} (1 - 1) \right] \\
&= \frac{a_0}{2} T + \sum_{n=1}^{\infty} \left[-\frac{b_n}{n w_0} (0) \right] \\
&= \frac{a_0}{2} T + \sum_{n=1}^{\infty} 0 \\
&= \frac{a_0}{2} T
\end{aligned}$$

o que implica em

$$\int_0^T f(x)\,dx = \frac{a_0}{2} T$$

e portanto

$$a_0 = \frac{2}{T} \int_0^T f(x)\,dx. \tag{7}$$

Para a determinação do coeficiente a_n, multiplica-se ambos os lados da Equação (6) por $\cos(mw_0x)$, com $m \in \mathbb{Z}_+^*$ e após, integra-se ambos os lados no intervalo $[0, T]$. Assim,

$$\begin{aligned}
&\int_0^T f(x)\cos(mw_0x)\,dx = \\
&= \int_0^T \left\{ \frac{a_0}{2}\cos(mw_0x) + \sum_{n=1}^{\infty}\left[a_n\cos(nw_0x)\cos(mw_0x) + b_n\,\mathrm{sen}(nw_0x)\cos(mw_0x)\right]\right\} dx \\
&= \int_0^T \frac{a_0}{2}\cos(mw_0x)\,dx + \sum_{n=1}^{\infty}\left\{ a_n\int_0^T \cos(nw_0x)\cos(mw_0x)\,dx + \right. \\
&\qquad \left. + b_n\int_0^T \mathrm{sen}(nw_0x)\cos(mw_0x)\,dx\right\}.
\end{aligned}$$

Pelas relações de ortogonalidade (FIGUEIREDO, 2000, p. 16-18), tem-se que a segunda integral dentro do somatório é nula e para a primeira integral, se $n \neq m$, ela também será nula, então recupera-se o resultado da Equação (7). Assim, faz-se $n = m$ e a primeira integral será igual a $\frac{T}{2}$. Segue que

$$\begin{aligned}
\int_0^T f(x)\cos(mw_0x)\,dx &= \frac{a_0}{2nw_0}\,\mathrm{sen}(nw_0x)\Big|_0^T + a_n\frac{T}{2} \\
&= \frac{a_0}{2nw_0}\left[\mathrm{sen}(nw_0T) - \mathrm{sen}\,0\right] + a_n\frac{T}{2} \\
&= \frac{a_0}{2nw_0}\,\mathrm{sen}(2n\pi) + a_n\frac{T}{2}.
\end{aligned}$$

Como $\mathrm{sen}(\pi n) = 0, \forall n \in \mathbb{Z}$,

$$\begin{aligned}
\int_0^T f(x)\cos(nw_0x)\,dx &= \frac{a_0}{2nw_0}(0) + a_n\frac{T}{2} \\
&= a_n\frac{T}{2}.
\end{aligned}$$

Portanto

$$a_n = \frac{2}{T}\int_0^T f(x)\cos(nw_0x)\,dx. \tag{8}$$

Resta a determinação do coeficiente b_n e para tanto, multiplica-se ambos os lados da Equação (6) por $\mathrm{sen}(mw_0x)$, com $m \in \mathbb{Z}_+^*$ e após, integra-se ambos os lados no intervalo

$[0, T]$. Assim,

$$\begin{aligned}
&\int_0^T f(x)\,\mathrm{sen}(mw_0x)\,dx = \\
&= \int_0^T \left\{ \frac{a_0}{2}\,\mathrm{sen}(mw_0x) + \sum_{n=1}^{\infty} \Big[a_n \cos(nw_0x)\,\mathrm{sen}(mw_0x) + b_n\,\mathrm{sen}(nw_0x)\,\mathrm{sen}(mw_0x) \Big] \right\} dx \\
&= \int_0^T \frac{a_0}{2}\,\mathrm{sen}(mw_0x)\,dx + \sum_{n=1}^{\infty} \left\{ a_n \int_0^T \cos(nw_0x)\,\mathrm{sen}(mw_0x)\,dx + \right. \\
&\qquad \left. + b_n \int_0^T \mathrm{sen}(nw_0x)\,\mathrm{sen}(mw_0x)\,dx \right\}.
\end{aligned}$$

Pelas relações de ortogonalidade (SANTOS, 2004), tem-se que a primeira integral dentro do somatório é nula e para a segunda integral, se $n \neq m$, ela também será nula, então recupera-se o resultado da Equação (7). Assim, faz-se $n = m$ e a primeira integral será igual a $\frac{T}{2}$. Vem que

$$\begin{aligned}
\int_0^T f(x)\,\mathrm{sen}(nw_0x)\,dx &= -\frac{a_0}{2nw_0}\cos(nw_0x)\Big|_0^T + b_n\frac{T}{2} \\
&= \frac{a_0}{2nw_0}\big[\cos 0 - \cos(nw_0T)\big] + b_n\frac{T}{2} \\
&= \frac{a_0}{2nw_0}\big[1 - \cos(2n\pi)\big] + b_n\frac{T}{2}.
\end{aligned}$$

Como, $\cos(2n\pi) = 1, \forall n \in \mathbb{Z}$,

$$\begin{aligned}
\int_0^T f(x)\,\mathrm{sen}(nw_0x)\,dx &= \frac{a_0}{2nw_0}(1-1) + b_n\frac{T}{2} \\
&= \frac{a_0}{2nw_0}(0) + b_n\frac{T}{2} \\
&= b_n\frac{T}{2}.
\end{aligned}$$

Portanto

$$b_n = \frac{2}{T}\int_0^T f(x)\,\mathrm{sen}(nw_0x)\,dx. \tag{9}$$

2.2.2 Representação em série de Fourier: exemplo

Como dito anteriormente, a determinação dos coeficientes das séries de Fourier é a mesma para qualquer função periódica estudada. Seja, por exemplo, uma função periódica f, com período $T = 2\pi$ dada pela expressão:

$$f(x) = \begin{cases} 1, & \text{se } -\pi \leq x < 0 \\ 0, & \text{se } 0 \leq x < \pi, \end{cases} \tag{10}$$

com $f(x+2\pi) = f(x)$.

A frequência angular $w_0 = \frac{2\pi}{T}$ para este caso, com $T = 2\pi$, é igual a 1 ($w_0 = 2\pi/2\pi = 1$). Então, calculando os coeficientes de Fourier, tem-se:

$$\begin{aligned} a_0 &= \frac{2}{T}\int_0^T f(x)\,dx = \frac{2}{2\pi}\int_0^{2\pi} f(x)\,dx \\ &= \frac{1}{\pi}\left[\int_{-\pi}^0 f(x)\,dx + \int_0^{\pi} f(x)\,dx\right] = \frac{1}{\pi}\left\{x\Big|_{-\pi}^0 + 0\Big|_0^{\pi}\right\} \\ &= \frac{1}{\pi}(0+\pi) = \frac{\pi}{\pi} = 1 \qquad\qquad \therefore a_0 = 1. \end{aligned}$$

Para a_n:

$$\begin{aligned} a_n &= \frac{2}{T}\int_0^T f(x)\cos(nw_0x)\,dx \\ &= \frac{2}{2\pi}\int_0^{2\pi} f(x)\cos(nx)\,dx \\ &= \frac{1}{\pi}\left\{\int_{-\pi}^0 f(x)\cos(nx)\,dx + \int_0^{\pi} f(x)\cos(nx)\,dx\right\} \\ &= \frac{1}{\pi}\left\{\frac{1}{n}\operatorname{sen}(nx)\Big|_{-\pi}^0 + 0\Big|_0^{\pi}\right\} \\ &= \frac{1}{\pi n}\big[\operatorname{sen}0 - \operatorname{sen}(-\pi n)\big]. \end{aligned}$$

Como, $\operatorname{sen}(-\pi n) = \operatorname{sen}0 = 0, \forall n \in \mathbb{Z}$, vem que

$$\begin{aligned} &\frac{1}{\pi n}\big[\operatorname{sen}(0) - \operatorname{sen}(-\pi n)\big] = \frac{1}{\pi n}(0) = 0 \\ &\therefore a_n = 0. \end{aligned}$$

Para b_n:

$$\begin{aligned} b_n &= \frac{2}{T}\int_0^T f(x)\operatorname{sen}(nw_0x)\,dx \\ &= \frac{2}{2\pi}\int_0^{2\pi} f(x)\operatorname{sen}(nx)\,dx \\ &= \frac{1}{\pi}\left\{\int_{-\pi}^0 f(x)\operatorname{sen}(nx)\,dx + \int_0^{\pi} f(x)\operatorname{sen}(nx)\,dx\right\} \\ &= \frac{1}{\pi}\left\{-\frac{1}{n}\cos(nx)\Big|_{-\pi}^0 + 0\Big|_0^{\pi}\right\} \\ &= \frac{1}{\pi n}\{\cos(-\pi n) - \cos 0 + 0\}. \end{aligned}$$

Como $\cos(x)$ é uma função par (SANTOS, 2004, p. 18), então

$$\cos(-\pi n) = \cos(\pi n), \qquad \forall n \in \mathbb{Z}.$$

Vem que, se n for um número par

$$\frac{1}{\pi n}\big[\cos(-\pi n) - \cos(0) + 0\big] = \frac{1}{\pi n}(1-1)$$
$$\therefore b_n = \frac{1}{\pi n}(0) = 0$$

e se n for ímpar,

$$\frac{1}{\pi n}\big[\cos(-\pi n) - \cos(0) + 0\big] = \frac{1}{\pi n}(-1-1)$$
$$= \frac{1}{\pi n}(-2) = -\frac{2}{\pi n}$$
$$\therefore b_n = -\frac{2}{\pi n}.$$

Logo, a aproximação em série de Fourier de f é dada por

$$f(x) \approx \frac{1}{2} + \sum_{n=1}^{\infty} b_n \operatorname{sen}(nx).$$

Ignorando as parcelas nulas do somatório — em que n é par —, supõe-se n ímpar, ou seja,

$$n = 2k+1, \qquad \forall k \in \mathbb{Z}_+,$$

temos

$$f(x) \approx \frac{1}{2} + \sum_{k=0}^{\infty} -\frac{2}{(2k+1)\pi} \operatorname{sen}\big[(2k+1)x\big].$$

Na Figura 2 verifica-se graficamente o resultado da série de Fourier para este exemplo, truncada em 1, 5, 10 e 30 termos.

2.2.3 Série de Fourier complexa

A partir da *Identidade de Euler*, dada por

$$e^{x+iy} = e^x\big(\cos y + i \operatorname{sen} y\big), \tag{11}$$

se desenvolve a forma complexa da série de Fourier. O desenvolvimento da fórmula foge do escopo do projeto, mas pode ser visto em (SANTOS, 2004, p. 33-35).

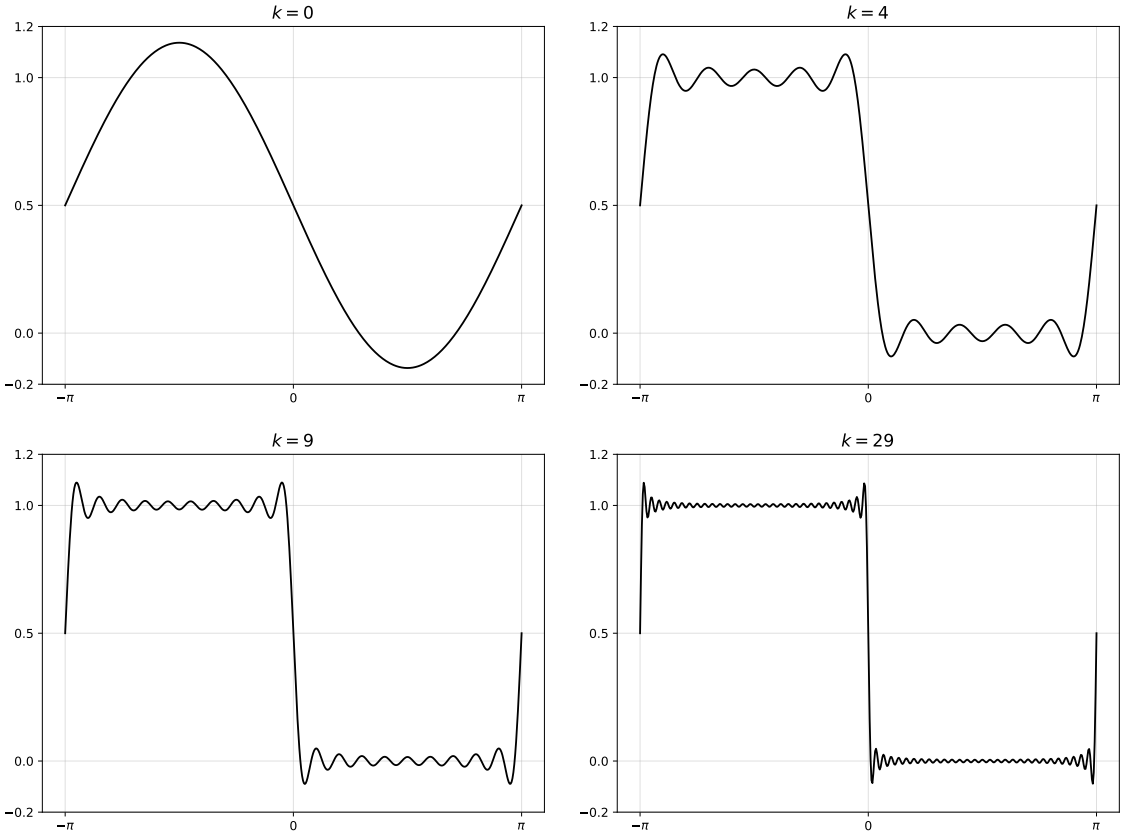
A forma complexa da série de Fourier como aproximação de uma função periódica f de período T é dada por

$$f(x) \approx \sum_{n\in\mathbb{Z}} c_n e^{inw_0 x}, \tag{12}$$

com

$$c_n = \frac{1}{T}\int_0^T f(x) e^{-inw_0 x}\, dx. \tag{13}$$

Figura 2 – Aproximações de f dada pela Equação (10) em série de Fourier truncada em 1, 5, 10 e 30 termos.



Fonte: o autor.

Conhecidas como harmônicos, funções da forma e^{inw_0x}, $n \in \mathbb{Z}$ — compostas por senoides e cossenoides — são importantes objetos de estudo para a representação complexa da série de Fourier, em especial, no estudo de música (ZANATO, 2017). E é exatamente o estudo sobre música a primeira e grande motivação para o desenvolvimento desse trabalho.

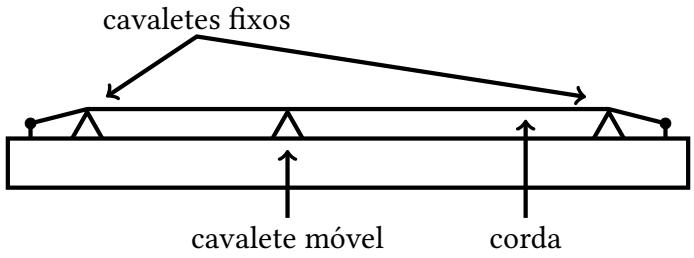
2.3 Séries de Fourier e música

Muito antes de Fourier, na Grécia antiga, o matemático e filósofo Pitágoras descobrira relações matemáticas que descrevem a harmonia dos sons (BEZERRA, 2018).

Ao utilizar um instrumento chamado monocórdio (como mostrado no desenho esquemático da Figura 3), Pitágoras descobriu que quando cordas com comprimentos específicos eram tensionadas, os sons produzidos eram agradáveis e harmonizavam entre si. Por

exemplo, se a corda tinha um comprimento ℓ, quando ele a fazia vibrar tensionada em ℓ e tensionada em $\frac{1}{2}\ell$, os sons produzidos eram praticamente os mesmos, porém mais agudos em $\frac{1}{2}\ell$.

Figura 3 – O monocórdio.



Fonte: (BESSADA, 2014, p. 106).

Voltando à Equação (2), vê-se que a frequência é inversamente proporcional ao período. Se este período é compreendido como intervalos de comprimento, como no caso estudado, então a frequência será inversamente proporcional ao comprimento da corda.

Sendo assim, quando a corda está tensionada em ℓ, a frequência f_0 é igual a $\frac{1}{\ell}$. Quando se tensiona a corda exatamente em $\frac{\ell}{2}$ a frequência f é igual ao dobro da frequência f_0, ou seja, $f = \frac{2}{\ell}$. Assim, a frequência f_0 da nota musical alcançada é conhecida como primeiro harmônico (ou harmônico fundamental), a frequência f é o segundo harmônico e assim por diante. Este fato tem relação direta com as séries de Fourier, em especial com a sua forma complexa, Equação (12).

Pitágoras continuou seus estudos e percebeu que quando tensionava a corda em $\frac{1}{3}\ell$, deixando vibrar $\frac{2}{3}\ell$ da corda, o som produzido não era igual e nem parecido com os dois primeiros. Contudo, era harmônico em relação a eles, ou seja, formavam uma combinação agradável.

Ele repetiu esse processo, deixando soar $\frac{2}{3}$ da corda restante, ou seja,

$$\frac{2}{3}\left(\frac{2}{3}\right)\ell = \frac{4}{9}\ell,$$

e percebeu que o som produzido também harmonizava com os anteriores.

Prosseguindo com a técnica, Pitágoras acabou por definir a chamada *Escala Pitagórica*. Se com a corda tensionada em $\ell = 1\,\text{u}$ (unidade de medida, quando não especificado), a nota alcançada for Dó, por exemplo, a escala será como apresentado na Tabela 1.

É fácil observar, na segunda coluna da Tabela 1, que o comprimento não é exatamente uma potência de $\frac{2}{3}$ como o raciocínio de Pitágoras apontava. A multiplicação por potências

Tabela 1 – Escala Pitagórica.

Nota musical	Fator de variação no comprimento da corda em relação a Dó	Fator de aumento na frequência produzida pela corda tensionada em relação a Dó
Dó	1	1
Sol	$2^0 \times \left(\frac{2}{3}\right)^1 \approx 0{,}667$	1,500
Ré	$2^1 \times \left(\frac{2}{3}\right)^2 \approx 0{,}889$	1,125
Lá	$2^1 \times \left(\frac{2}{3}\right)^3 \approx 0{,}593$	1,688
Mi	$2^2 \times \left(\frac{2}{3}\right)^4 \approx 0{,}790$	1,266
Si	$2^2 \times \left(\frac{2}{3}\right)^5 \approx 0{,}527$	1,898
Fá#	$2^3 \times \left(\frac{2}{3}\right)^6 \approx 0{,}702$	1,424
Dó#	$2^4 \times \left(\frac{2}{3}\right)^7 \approx 0{,}936$	1,068
Sol#	$2^4 \times \left(\frac{2}{3}\right)^8 \approx 0{,}624$	1,602
Ré#	$2^5 \times \left(\frac{2}{3}\right)^9 \approx 0{,}832$	1,201
Lá#	$2^5 \times \left(\frac{2}{3}\right)^{10} \approx 0{,}555$	1,802
Fá	$2^6 \times \left(\frac{2}{3}\right)^{11} \approx 0{,}740$	1,352
Dó*	$2^6 \times \left(\frac{2}{3}\right)^{12} \approx 0{,}493$	2,027

de 2 deriva da primeira observação dele: os sons produzidos pela corda tensionada em ℓ e $\frac{\ell}{2}$ são tão parecidos, que soam praticamente da mesma forma, porém um mais agudo que outro (dobro da frequência). Assim, eles são considerados sons equivalentes e levam o mesmo nome. A nota Fá, por exemplo, pelo raciocínio de potências de $\frac{2}{3}$, seria produzida com a corda tensionada em $\frac{\ell}{86{,}498}$, com $\ell = 1\,\text{u}$. Como o som produzido pela corda tensionada com o dobro desse comprimento é muito parecido com ele próprio, então o dobro do dobro também o será e assim por diante. Assim, o som produzido com a corda tensionada com um comprimento de 0,74 u também será uma nota Fá, porém mais grave (menor frequência) que a anterior.

Nota-se também que ao reposicionar as notas de modo que a frequência aumente gradativamente, as notas seguirão a ordem apresentada na Tabela 2.

A última nota da Tabela 2, Dó*, tem frequência muito próxima ao dobro da frequência

Tabela 2 – Escala Pitagórica ordenada.

Nota musical	Fator de variação no comprimento da corda em relação a Dó	Fator de aumento na frequência produzida pela corda tensionada em relação a Dó
Dó	1	1
Dó#	$2^4 \times \left(\frac{2}{3}\right)^7 \approx 0{,}936$	1,068
Ré	$2^1 \times \left(\frac{2}{3}\right)^2 \approx 0{,}889$	1,125
Ré#	$2^5 \times \left(\frac{2}{3}\right)^9 \approx 0{,}832$	1,201
Mi	$2^2 \times \left(\frac{2}{3}\right)^4 \approx 0{,}790$	1,266
Fá	$2^6 \times \left(\frac{2}{3}\right)^{11} \approx 0{,}740$	1,352
Fá#	$2^3 \times \left(\frac{2}{3}\right)^6 \approx 0{,}702$	1,424
Sol	$2^0 \times \left(\frac{2}{3}\right)^1 \approx 0{,}667$	1,500
Sol#	$2^4 \times \left(\frac{2}{3}\right)^8 \approx 0{,}624$	1,602
Lá	$2^1 \times \left(\frac{2}{3}\right)^3 \approx 0{,}593$	1,688
Lá#	$2^5 \times \left(\frac{2}{3}\right)^{10} \approx 0{,}555$	1,802
Si	$2^2 \times \left(\frac{2}{3}\right)^5 \approx 0{,}527$	1,898
Dó*	$2^6 \times \left(\frac{2}{3}\right)^{12} \approx 0{,}493$	2,027

de Dó, mas ao contrário do agradável som produzido pelo exato dobro da frequência de Dó, o som produzido por Dó* não é harmônico com a nota Dó. Assim, em um período posterior ao de Pitágoras, criou-se a escala temperada, a partir da procura de um número $c \in \mathbb{R}$ tal que $c^{12} = 2c^0$.

Assim,

$$c^{12} = 2 \iff \sqrt[12]{c^{12}} = \sqrt[12]{2} \iff c \approx 1{,}059.$$

Este número c é chamado de semitom temperado (ou apenas semitom). Em música, os semitons são vistos como uma medida linear, ou seja, ouve-se que "a nota Mi está quatro semitons acima de Dó e três semitons abaixo de Sol", por exemplo. Porém, matematicamente, as notas distam potências de semitom entre si. A Tabela 3 mostra a escala temperada, tendo as doze notas musicais na escala, cada uma seguida pela razão de intervalo (matemática) e pela quantidade de semitons (música).

Tabela 3 – Escala Temperada.

Nota musical	Razão de Intervalo	Quantidade de semitons
Dó	$c^0 = 1$	0
Dó#	$c^1 \approx 1{,}059$	1
Ré	$c^2 \approx 1{,}122$	2
Ré#	$c^3 \approx 1{,}189$	3
Mi	$c^4 \approx 1{,}260$	4
Fá	$c^5 \approx 1{,}335$	5
Fá#	$c^6 \approx 1{,}414$	6
Sol	$c^7 \approx 1{,}498$	7
Sol#	$c^8 \approx 1{,}587$	8
Lá	$c^9 \approx 1{,}682$	9
Lá#	$c^{10} \approx 1{,}782$	10
Si	$c^{11} \approx 1{,}888$	11
Dó2	$c^{12} = 2$	12

Na Tabela 3 verifica-se que Dó2 tem exatamente o dobro da frequência de Dó. Em música, diz-se que Dó2 está uma oitava acima de Dó — ou a doze semitons de distância.

Voltando à harmonia dos sons, nota-se uma estreita relação com a série harmônica

$$\sum_{x=1}^{\infty} \frac{1}{x}.$$

Se por um lado ℓ, $\frac{\ell}{2}$ e $\frac{\ell}{3}$ (deixando soar $\frac{2}{3}\ell$) produzem sons harmônicos, de outro lado, na Página 24 verifica-se que a aproximação de uma função por série de Fourier apresenta muita semelhança com a série harmônica. Os três primeiros termos daquela série são, por exemplo,

$$\frac{1}{2} + \frac{-2\,\mathrm{sen}(x)}{\pi}; \frac{1}{2} + \frac{-2\,\mathrm{sen}(3x)}{3\pi}; \frac{1}{2} + \frac{-2\,\mathrm{sen}(5x)}{5\pi},$$

que são iguais, respectivamente a

$$\frac{1}{2} - \left[\left(\frac{1}{1}\right)\frac{2\,\mathrm{sen}(x)}{\pi}\right]; \frac{1}{2} - \left[\left(\frac{1}{3}\right)\frac{2\,\mathrm{sen}(3x)}{\pi}\right]; \frac{1}{2} - \left[\left(\frac{1}{5}\right)\frac{2\,\mathrm{sen}(3x)}{\pi}\right].$$

As frações $\frac{1}{1}$, $\frac{1}{3}$ e $\frac{1}{5}$ são os três primeiros termos da série harmônica

$$\sum_{k=0}^{\infty} \frac{1}{x}, \quad x = 2k+1.$$

Dadas as relações entre séries de Fourier e a música e a forte ligação entre período e frequência, presente em todo o decorrer do texto até aqui, surge o questionamento de como criar uma correspondência entre essas duas grandezas. É neste momento que a Transformada de Fourier mostra sua importância.

2.4 Transformada discreta de Fourier

Um sinal digital de áudio é interpretado pelo computador como sendo uma sequência numérica $x(t)$ de amplitudes do sinal em função do tempo (BLACK; EDSON, 1947; PETZOLD, 2000).

Com a Transformada de Fourier (BRACEWELL, 1989), tornou-se possível descrever esse mesmo sinal como uma sequência numérica de amplitudes $X(f)$ em função da frequência. Assim, os principais harmônicos dos instrumentos em uma música, por exemplo, passaram a ser percebidos e estudados com facilidade.

Conceitualmente, a principal característica da Transformada de Fourier é permitir analisar um sinal em função da frequência e essa é uma potente ferramenta para o processamento de sinais, em particular, para a redução de ruídos, aplicação a que este trabalho se dedica.

A Transformada de Fourier permite que o sinal seja dividido em bandas de frequência e cada uma dessas bandas pode ser analisada para identificar ruídos contidos no sinal.

Como o computador trabalha no mundo digital, é necessária uma discretização dos dados analisados. Assim, tanto a sequência de dados de entrada, quanto a sequência de saída devem ser discretas. Por isso computacionalmente faz-se uso da *Transformada Discreta de Fourier* (ou DFT[4]) que é definida por

$$X(m) = \sum_{n=0}^{N-1} x(n) e^{\frac{-i2\pi nm}{N}}, \tag{14}$$

onde:

- m = o índice de saída da DFT no domínio da frequência, $m = 0, 1, 2, 3, \ldots, N-1$;
- $X(m)$ = o m-ésimo componente de saída da DFT;

[4] Discrete Fourier Transform, em inglês.

- n = o índice das amostras de entrada no domínio do tempo, $n = 0, 1, 2, 3, \ldots, N-1$;
- $x(n)$ = sequência de amostras de entrada;
- N = número de amostras da sequência de entrada e o número de pontos de saída da DFT (LYONS, 2011).

Utilizando a Identidade de Euler, é possível reescrever a DFT em termos de senos e cossenos

$$X(m) = \sum_{n=0}^{N-1} x(n)\left[\cos\left(\frac{2\pi nm}{N}\right) - i\,\mathrm{sen}\left(\frac{2\pi nm}{N}\right)\right]. \tag{15}$$

2.4.1 DFT: exemplo

Pode-se tomar como exemplo a realização de uma DFT com oito pontos. A Tabela 4 mostra a sequência de entrada da DFT, $x(n) = \mathrm{sen}\, n$, amostrada a uma taxa $f_s = 1$ amostras por segundo, tais que $-1 < x(n) < 1$.

Tabela 4 – Sequência de entrada da DFT.

$x(0) = 0$	$x(4) = -0,7568025$
$x(1) = 0,84147098$	$x(5) = -0,95892427$
$x(2) = 0,90929743$	$x(6) = -0,2794155$
$x(3) = 0,14112001$	$x(7) = 0,6569866$

A saída da DFT retornará valores em função da frequência, de modo que cada harmônico $X(m)$ conterá a parte real acrescida de sua parte imaginária. Os harmônicos podem ser determinados a partir da taxa de amostragem:

$$f_m = \frac{mf_s}{N}. \tag{16}$$

A Tabela 5 mostra os *bins*[5] da frequência para o exemplo apresentado.

Assim, a sequência de saída da DFT será dada como mostra a Tabela 6.

A Figura 4 traz os gráficos de $x(n)$ em função do tempo e a parte real de $X(m)$ em função da frequência. Este último recebe o nome de espectro de frequências. Analisando-a nota-se uma simetria nos valores de $X(m)$, excluindo-se seu primeiro ponto, $\big(f(0), X(0)\big)$. Esta simetria ocorre quando a sequência de entrada apresenta valores reais, ainda que a saída da DFT tenha sua parte imaginária. Assim, é necessário calcular somente os $\frac{N}{2} + 1$ primeiros termos da DFT. Os outros termos serão os seus respectivos números conjugados[6].

[5] *bins* é um termo utilizado para se referir às amostras no domínio da frequência (LYONS, 2011, p. 92-93).

[6] O conjugado de um número complexo $z = x + iy$ é igual a sua parte real mais o oposto de sua parte

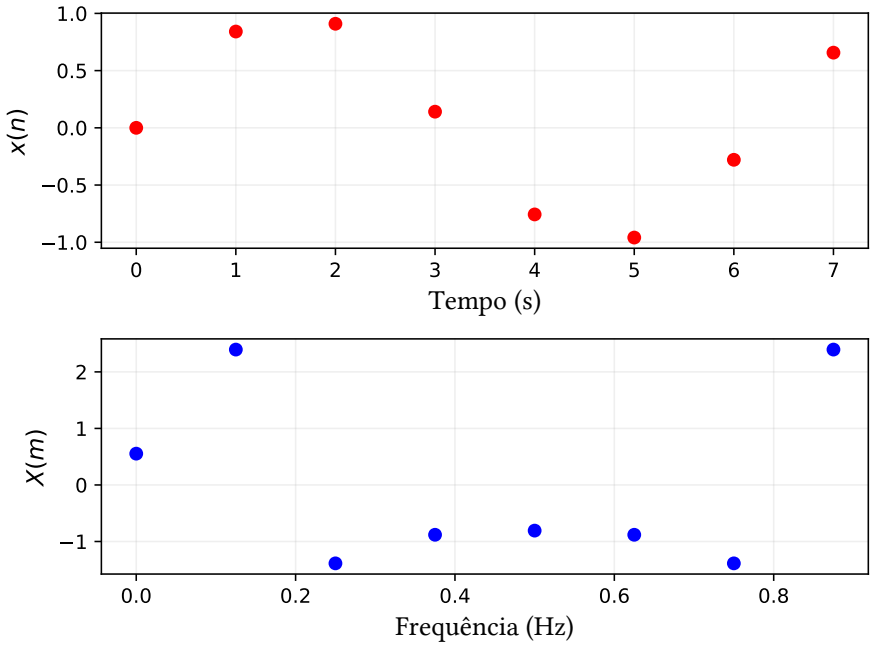
Tabela 5 – Valores do domínio (frequência), em Hz.

$f_0 = \frac{0\times1}{8} = 0$	$f_1 = \frac{1\times1}{8} = 0{,}125$
$f_2 = \frac{2\times1}{8} = 0{,}25$	$f_3 = \frac{3\times1}{8} = 0{,}375$
$f_4 = \frac{4\times1}{8} = 0{,}5$	$f_5 = \frac{5\times1}{8} = 0{,}625$
$f_6 = \frac{6\times1}{8} = 0{,}75$	$f_7 = \frac{7\times1}{8} = 0{,}875$

Tabela 6 – Sequência de saída da DFT.

$X(0) = 0{,}553\,732\,75 + 0i$	$X(4) = -0{,}807\,573\,88 + 0i$
$X(1) = 2{,}394\,646\,96 - 2{,}097\,011\,86i$	$X(5) = -0{,}881\,041\,97 - 0{,}280\,413\,99i$
$X(2) = -1{,}386\,684\,42 + 0{,}915\,559\,9i$	$X(6) = -1{,}386\,684\,42 - 0{,}915\,559\,9i$
$X(3) = -0{,}881\,041\,97 + 0{,}280\,413\,99i$	$X(7) = 2{,}394\,646\,96 + 2{,}097\,011\,86i$

Figura 4 – $x(n)$ em função do tempo e parte real de $X(m)$ em função da frequência.



Fonte: o autor.

2.4.2 Magnitude

Quando se fala em Transformada de Fourier, frequentemente procura-se analisar a resposta em frequência do objeto em estudo. Esta resposta em frequência é dada pela DFT,

imaginária, ou seja, $z^* = x - iy$.

mais especificamente pela magnitude de cada uma de suas saídas no domínio da frequência.

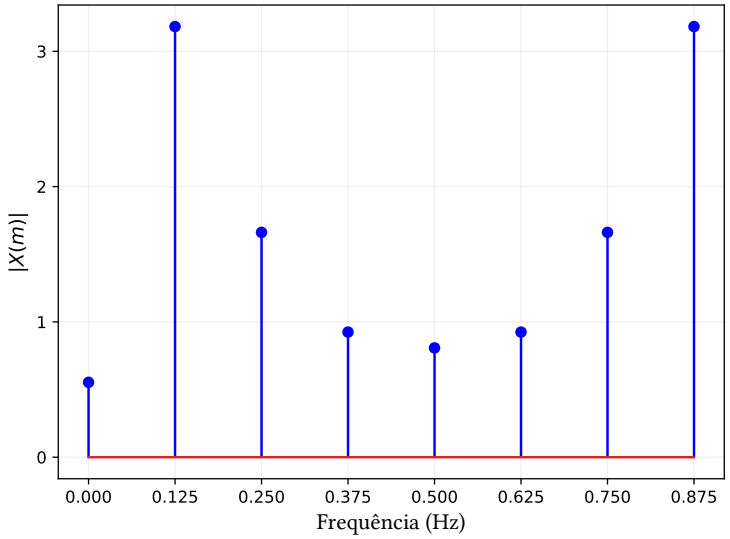
A magnitude para uma saída $X(m)$ da DFT nada mais é do que o conjunto de valores absolutos apresentados pela DFT e é dada pela Equação (17),

$$X_{\text{mag}}(m) = |X(m)| = \sqrt{X^2_{\text{real}}(m) + X^2_{\text{imag}}(m)}, \tag{17}$$

sendo $X_{\text{real}}(m)$ a parte real da saída da DFT e $X_{\text{imag}}(m)$ a parte imaginária da saída da DFT.

A Figura 5 apresenta o espectro de magnitudes da DFT da Figura 4.

Figura 5 – Espectro de magnitudes da DFT (em função da frequência).



Fonte: o autor.

Em se tratando de áudio, a magnitude dos sinais é manipulada em decibéis. O decibel é uma unidade logarítmica que indica a proporção entre uma quantidade física (Energia) em relação a um nível de referência. Sua conversão é dada pela Equação (18) (LYONS, 2011).

$$X_{\text{dB}}(m) = 20\log_{10}\left(\frac{|X(m)|}{|X(0)|}\right)\text{dB}. \tag{18}$$

Esta representação garante que $X_{\text{dB}}(0) = 0\,\text{dB}$, pois

$$X_{\text{dB}}(0) = 20\log_{10}\left(\frac{|X(0)|}{|X(0)|}\right) = 20\log_{10}(1) = 20 \times 0 = 0\,\text{dB}.$$

A conversão para decibéis ajuda na visualização do comportamento da DFT e na comparação entre diferentes respostas em frequência de filtros digitais.

2.4.3 Transformada inversa de Fourier

Em sua definição, a DFT essencialmente é usada para criar um espectro de frequências a partir de uma sequência discreta de amostras no domínio do tempo, passando a tratar da sequência no domínio da frequência (LYONS, 2011).

A partir disto, define-se a *Transformada Discreta Inversa de Fourier* (ou IDFT[7]) como a operação inversa da DFT. Tendo uma sequência no domínio da frequência $X(m)$, a IDFT de m pontos é calculada e retorna uma sequência $x(m)$ no domínio do tempo.

A IDFT é muito importante principalmente em reconstrução de sinais, pois toda a manipulação do sinal ocorre — tradicionalmente — no domínio da frequência, porém o sinal deve estar no domínio do tempo para poder passar para o mundo real — e analógico — sem perdas ou modificações.

2.4.4 Transformada rápida de Fourier

Computacionalmente, o cálculo da DFT tem um custo muito alto, causado pela necessidade de realizar várias multiplicações. Esse custo aumenta à medida em que o número de pontos da DFT aumenta. Assim, apesar de seu poder matemático, sua eficiência computacional era muito baixa. Porém, em 1965, James W. Cooley e John W. Tukey publicaram um artigo intitulado *An algorithm for the machine calculation of complex Fourier series*, apresentando a todos a *Transformada Rápida de Fourier* (ou FFT[8]) (COOLEY; TUKEY, 1965).

A FFT nada mais é do que um algoritmo para a implementação computacional da DFT. Seu único requisito é que N seja uma potência de 2 (COOLEY; TUKEY, 1965).

Assim, se em uma sequência de entrada $x(n)$ não houver um número em potência de 2 de amostras, a sequência é preenchida com zeros, até que se tenha o tamanho necessário (em potência de 2). Por exemplo, um sinal com 188 amostras passará por uma FFT, porém 188 não é uma potência de 2. Então, a esse sinal, são acrescidas 68 amostras com valor zero e assim, uma FFT de 256 pontos é realizada.

É possível controlar qual a resolução que a FFT retornará, ou seja, o espaçamento entre cada um dos *bins*. Se deseja-se, por exemplo, uma resolução de 20 Hz na FFT e tem-se uma taxa de amostragem $f_s = 40000$ amostras por segundo,

$$N = \frac{f_s}{\text{resolução desejada}}, \tag{19}$$

$$\Rightarrow N = \frac{40000}{20} = 2000.$$

[7] *Inverse Discrete Fourier Transform*, em inglês

[8] *Fast Fourier Transform*, em inglês.

Como 2000 não é potência de 2, deverá ser feita uma FFT de 2048 pontos.

Neste trabalho, a FFT foi realizada utilizando o pacote *NumPy* (VAN DER WALT; COLBERT; VAROQUAUX, 2011) do software de código aberto *SciPy* (JONES; OLIPHANT; PETERSON et al., 2001) que foi implementado com linguagem de programação *Python* (VAN ROSSUM; DRAKE JR, 1995) em um ambiente chamado *Jupyter Notebook* (KLUYVER et al., 2016).

É importante pontuar que a IDFT também pode ser obtida através da FFT, incrementando sua velocidade, assim como no caso da DFT.

3 Filtros digitais

Filtros digitais são sistemas de processamento de sinais que recebem a entrada de um sinal de tempo discreto e o manipulam no domínio da frequência, de modo que os níveis de som de suas amostras podem ser amplificadas ou atenuadas, dependendo da aplicação e do espectro de frequências desejado. A saída do filtro é um sinal também de tempo discreto. O que diferencia os filtros de outros sistemas digitais é a utilização de uma correspondência entre entrada e saída do sinal. Um exemplo de processo de filtragem digital é a diferenciação (ANTONIOU, 2006, p. 8).

Em processamento de sinais, os filtros digitais são aplicados a sinais de áudio e de imagem, como pode ser verificado em cursos e aplicações de engenharia de telecomunicações. A principal característica desses filtros é sua capacidade de reduzir componentes indesejáveis dos sinais de entrada. Em outras palavras, os filtros digitais permitem que algumas frequências passem sem alteração por eles enquanto outras frequências têm sua magnitude atenuada.

A história dos filtros digitais começa no século XVII, quando matemáticos em seus estudos sobre áreas de formas geométricas e astrônomos em suas tentativas de entender e racionalizar as formas das órbitas dos planetas perceberam a necessidade de interpolar dados numéricos vindos de suas observações. Com essa necessidade, muitas propostas de fórmulas de interpolação numérica foram propostas, como por exemplo, por Newton (1642-1727) e Lagrange (1736-1813). Nos anos cinquenta e sessenta foram desenvolvidos algoritmos capazes de processar sinais representados por dados numéricos. Esse avanço foi possível graças ao computador digital, responsável desde então pela aplicação em massa de diferentes ferramentas computacionais para o aprimoramento do processamento de sinais. Após um constante crescimento dos estudos em torno do assunto os algoritmos, os programas de

computador e os sistemas que fossem usados para processamento digital de sinais em forma de dados numéricos passaram a ser denominados como filtros digitais (ANTONIOU, 2006).

De uma forma geral, os filtros procuram eliminar componentes indesejáveis de um sinal, conhecidos como *ruído*.

3.1 Ruídos

Ruído é tudo aquilo que causa distúrbios aleatórios indesejáveis em um sinal. Eles são produzidos de diversas formas e são praticamente inevitáveis no processamento de sinais.

No caso estudado, os ruídos apresentam-se como um conjunto de ondas aleatórias com frequências que, ao serem ouvidas em conjunto com uma música, produzem sons desarmônicos, desagradáveis ao ouvido humano.

Neste trabalho, a proposta foi atenuar ruídos de fundo, que podem ser produzidos por sussurros durante a gravação, silvos ou zumbidos. Estes ruídos também podem ser produzidos pelo equipamento de gravação utilizado.

A música analisada é da década de 1920 e apresenta bastante ruído de fundo, devido aos equipamentos e tecnologia disponíveis e utilizados à época da gravação. Alguns tipos de ruídos tem características únicas e assim, recebem um nome específico:

Ruído branco apresenta igual intensidade em todo seu espectro de frequências e tem média de intensidade de suas amostras igual a zero (VETTERLI; PRANDONI, 2008, p. 227).

Ruído rosa a intensidade em cada *bin* de frequência f_n é igual a $\frac{1}{f_n}$, ou seja, é inversamente proporcional à frequência (DUTTA; HORN, 1981).

Ruído marrom apresenta uma atenuação mais acentuada que o ruído rosa, a intensidade em cada *bin* de frequência f_n é igual a $\frac{1}{f_n^2}$ (HALLEY, 1996).

Outro fato inconveniente no processamento de sinais decorre do processo de amostragem do sinal analógico, que depende basicamente da capacidade de memória e processamento do hardware utilizado.

3.2 O efeito *aliasing*

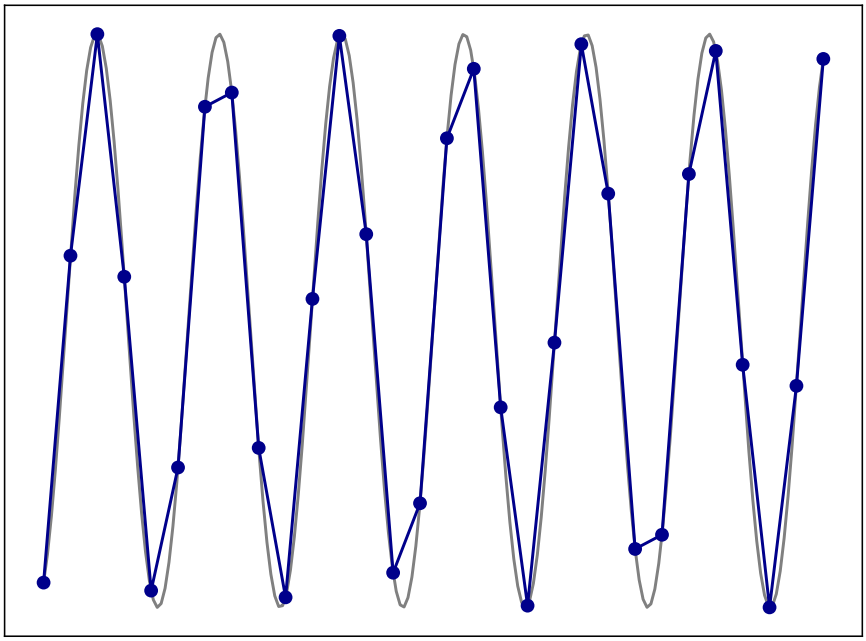
O fenômeno de *aliasing* ocorre quando a taxa de amostragem é menor que a ideal e algumas informações do sinal não são amostradas corretamente. Seja, por exemplo, um

sinal $x(n) = \text{sen}(20n)$, amostrado a uma frequência de amostragem f_s de

$$\frac{30}{2\pi} = 4{,}775\text{s}^{-1}.$$

A Figura 6 ilustra o exemplo apresentado. Fica claro que a onda formada pelos pontos amostrados não descreve satisfatoriamente o sinal de entrada. Pior que isso: supõe um sinal diferente, que passa pelos mesmos pontos que a sequência real de entrada.

Figura 6 – Exemplo do efeito de *aliasing*.



Fonte: o autor.

O *aliasing* está ligado à periodicidade das ondas mais simples, como a senoide, por exemplo. Assim, para uma sequência de entrada $x(n)$, tem-se:

$$\text{sen}(2\pi f_0 n t_s), \qquad n \in \mathbb{Z}_+, \tag{20}$$

sendo f_0 a frequência fundamental do sinal e $t_s = \frac{1}{f_s}$ (período de amostragem e frequência de amostragem, respectivamente).

Pela periodicidade da função seno, tem-se que

$$\begin{aligned} x(n) &= \operatorname{sen}(2\pi f_0 n t_s) = \operatorname{sen}(2\pi f_0 n t_s + 2\pi m), \qquad m \in \mathbb{Z} \\ &= \operatorname{sen}(2\pi(f_0 n t_s + m)) = \operatorname{sen}\left(2\pi(f_0 n t_s + m)\frac{n t_s}{n t_s}\right) \\ &= \operatorname{sen}\left(2\pi\left(f_0 + \frac{m}{n t_s}\right) n t_s\right). \end{aligned}$$

Supõe-se $m = kn$, $k \in \mathbb{Z}$. Como $t_s = \frac{1}{f_s}$, vem que

$$\operatorname{sen}\left(2\pi(f_0 + \frac{m}{n t_s}) n t_s\right) = \operatorname{sen}\left(2\pi\left(f_0 + \frac{kn}{n\frac{1}{f_s}}\right) n t_s\right).$$

Logo

$$x(n) = \operatorname{sen}(2\pi(f_0 + k f_s) n t_s). \tag{21}$$

Igualando as Equações (20) e (21), conclui-se que

$$x(n) = \operatorname{sen}(2\pi f_0 n t_s) = \operatorname{sen}(2\pi(f_0 + k f_s) n t_s).$$

Portanto

$$\operatorname{sen}(2\pi f_0 n t_s) = \operatorname{sen}(2\pi(f_0 + k f_s) n t_s). \tag{22}$$

Desta forma, o sinal deve ser amostrado a uma taxa com, no mínimo, o dobro do valor da maior frequência do sinal analógico.

Em palavras, "ao amostrar a uma taxa de f_s amostras/segundo, se k é qualquer inteiro positivo ou negativo, não podemos distinguir entre os valores amostrados de uma onda senoidal de f_0 Hz e uma onda senoidal de $(f_0 + k f_s)$ Hz" (LYONS, 2011, p. 54).

Outro efeito indesejável ocorre com a maioria dos sinais do mundo real ao realizar a Transformada de Fourier. Esse efeito chama-se *leakage* e será apresentado a seguir.

3.3 O efeito *leakage*

É inegável a contribuição que a Transformada de Fourier exerce no processamento de sinais. Porém, com ela, podem surgir problemas que influenciam diretamente o sinal estudado.

O *leakage* é um problema causado durante a Transformada de Fourier e faz com que o espectro de frequências gerado seja apenas uma aproximação do espectro do sinal original (ainda que possa ser uma ótima aproximação).

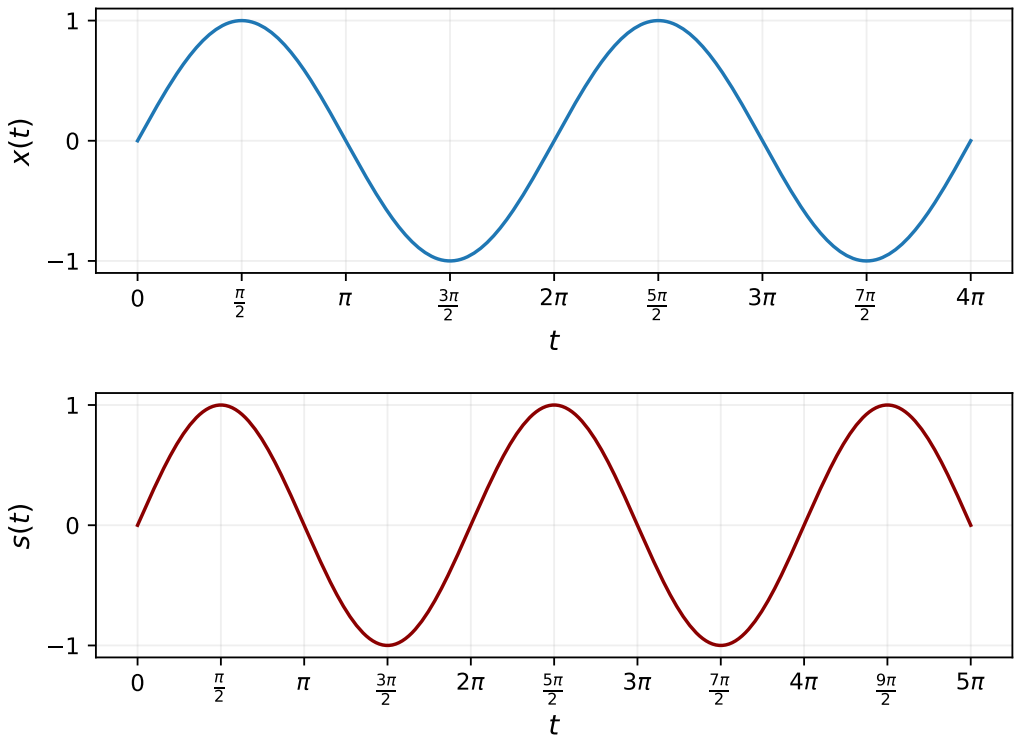
Os *bins* de frequência — calculados pela Equação (16) — mostram os harmônicos do sinal estudado. Porém, se no sinal existe um componente em

$$\frac{2,5 f_s}{N},$$

por exemplo, este não terá sua energia exatamente em algum dos *bins*. Como este componente não pode ser ignorado pela DFT, sua energia será mostrada em todas as saídas da DFT, afetando assim o espectro de frequências (LYONS, 2011, p. 92-93).

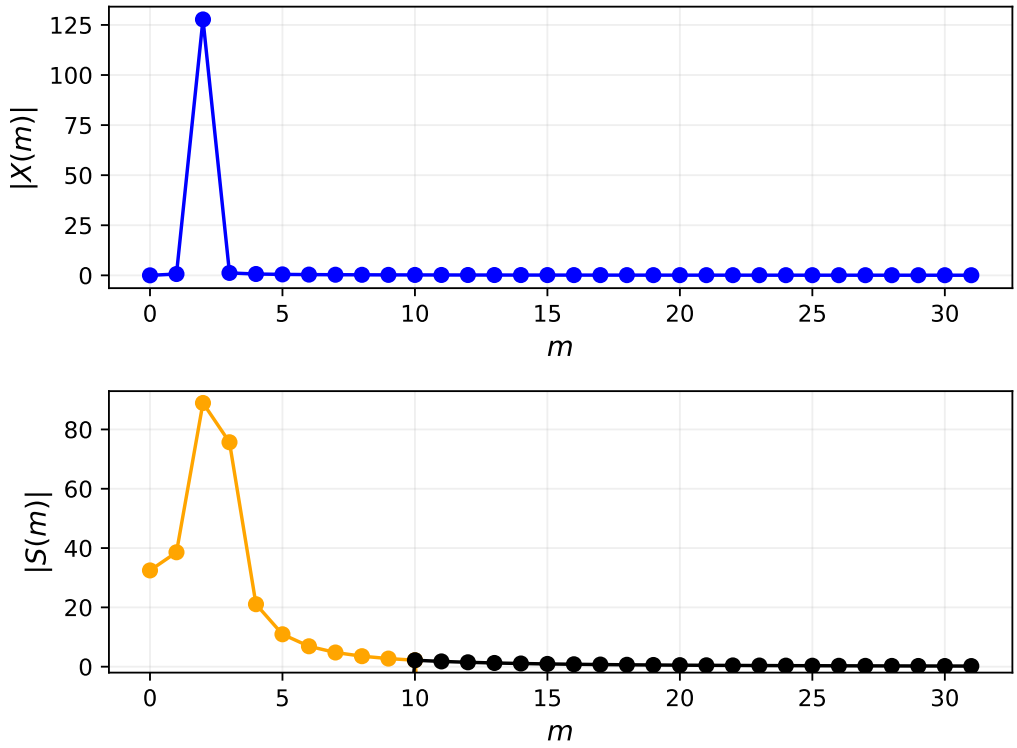
A Figura 7 mostra duas senoides como sequências de entrada e a Figura 8 mostra as saídas reais das DFTs de 64 pontos das duas sequências de entrada, respectivamente. A primeira sequência de entrada apresenta 2 ciclos completos e assim, em seu espectro de frequências, sua energia está centrada em apenas um *bin* de frequência. Isto já era esperado, uma vez que somente uma senoide está na composição do sinal. Já a segunda sequência não apresenta ciclos completos e isso causa um *leakage*, tornando evidente o efeito causado por essa distorção na Figura 8.

Figura 7 – $x(t)$, com 2 ciclos completos e $s(t)$, com 2,5 ciclos.



Fonte: o autor.

Figura 8 – $|X(m)|$ e $|S(m)|$, com *leakage*.



Fonte: o autor.

Para minimizar as distorções ocasionadas pelo *leakage*, são usadas as *janelas*.

3.4 Janelas

As janelas são utilizadas na redução do *leakage* e também podem ser combinadas com filtros para aprimorar sua resposta em frequência. O *leakage* é reduzido através da redução da magnitude dos *lobos laterais* da função

$$\text{sinc(x)} = \frac{\text{sen}(x)}{x},$$

que será apresentada com mais detalhes na Subseção 3.5.2. Esse processo ocorre fazendo com que as amplitudes da sequência de entrada (em função do tempo) sejam suavizadas e terminem em um único valor, comum ao início e ao final do intervalo de amostragem.

Como o próprio nome sugere, as janelas são funções que permitem que a magnitude de certas frequências permaneçam inalteradas enquanto outras são atenuadas.

As janelas mais comuns são:

Janela retangular é dada por $w(n) = 1$, para $0 \leq n \leq (N-1)$ (LYONS, 2011, p. 101).

Janela triangular é dada por

$$w(n) = \begin{cases} \dfrac{n}{N/2}, & \text{se } 0 \leq n \leq \dfrac{N}{2} \\ 2 - \dfrac{n}{N/2}, & \text{se } \dfrac{N}{2} + 1 \leq n \leq N-1. \end{cases}$$

Janela de Hanning é dada por

$$w(n) = 0{,}5 - 0{,}5\cos\left(\frac{2\pi n}{N}\right), \quad 0 \leq n \leq (N-1).$$

Janela de Hamming é dada por

$$w(n) = 0{,}54 - 0{,}46\cos\left(\frac{2\pi n}{N}\right), \quad 0 \leq n \leq (N-1).$$

Em todos os casos n é um número inteiro não negativo e N é o número de pontos da DFT a ser realizada.

A aplicação das janelas se dá pela DFT da multiplicação entre a sequência de entrada e a janela (LYONS, 2011), como mostra a Equação (23)

$$X_w(m) = \sum_{n=0}^{N-1} w(n)x(n)e^{\frac{-i2\pi nm}{N}}. \tag{23}$$

As janelas tem uma aplicação muito importante em filtros digitais. A seguir são apresentados os Filtros de Resposta ao Impulso Finita (FIR).

3.5 Filtros de resposta ao impulso finita

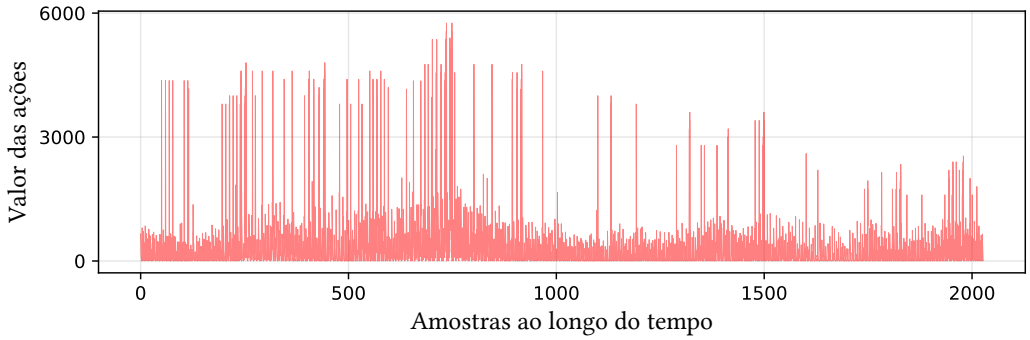
Os filtros FIR — *Finite Impulse Response* (do Inglês, Filtros de Resposta ao Impulso Finita), que juntamente com os filtros IIR — *Infinite Impulse Response* (do Inglês, Filtros de Resposta ao Impulso Infinita) são os dois tipos de filtros digitais lineares, tradicionalmente utilizados em processamento de sinais.

Os filtros FIR recebem um número limitado de valores de entrada não nulos e após seu processamento, retorna um número finito de valores não nulos como saída. Se os valores de entrada tornarem-se nulos em um dado momento, a saída desses valores também será nula. Um exemplo simples de utilização de um filtro FIR é a média (LYONS, 2011).

Seja, por exemplo, a variação do valor das ações da empresa VALE na bolsa de valores da BOVESPA no ano de 2008 — alguns meses após sua privatização. O preço das ações teve grande taxa de variação durante o ano.

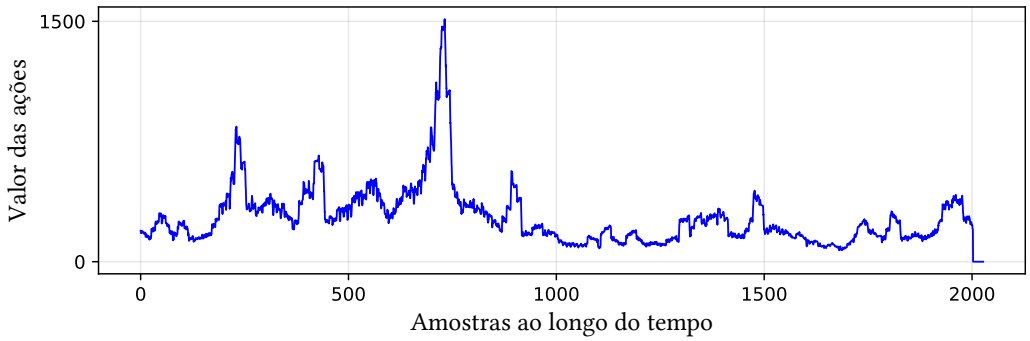
O filtro média é capaz de suavizar a curva formada por essa variação e ajuda a compreender melhor o comportamento dos dados. A Figura 9 mostra o gráfico dos dados de entrada do filtro média com a variação dos valores das ações ao longo do ano de 2008. A Figura 10 mostra o gráfico do sinal com a aplicação de um filtro média de 100 pontos e a Figura 11 mostra o gráfico do sinal com a aplicação de um filtro média de 400 pontos.

Figura 9 – Variação das cotas da VALE em 2008.



Fonte: `http://www.bmfbovespa.com.br/pt_br/servicos/market-data/historico/mercado-a-vista/series-historicas/`.

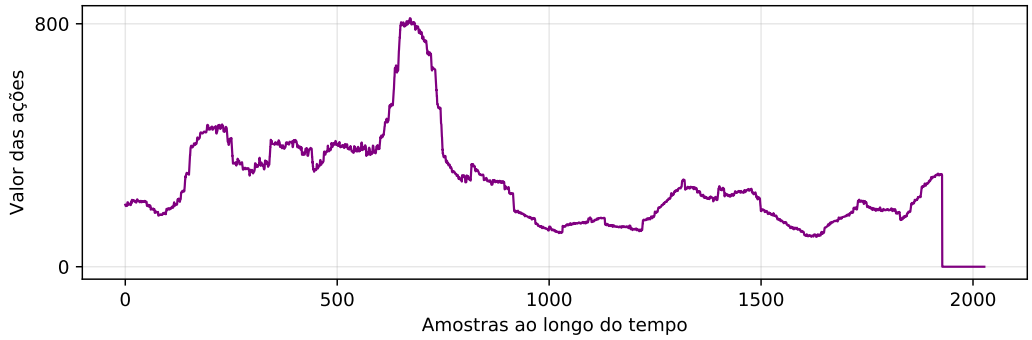
Figura 10 – Aplicação de um filtro média com 100 pontos.



Fonte: O autor.

Pelas Figuras 9, 10 e 11, pode-se notar que o gráfico fica mais suave à medida em que o número de pontos do filtro aumenta. Isso explica-se pelo fato das médias calculadas necessitarem de um número cada vez maior de dados, ou seja, um número menor de médias é calculado e assim, há uma variação menor de valores. A saída desse filtro tende a ficar cada vez mais linear, aproximando-se da média de todos os valores de entrada.

Figura 11 – Aplicação de um filtro média com 400 pontos.



Fonte: O autor.

3.5.1 Convolução

No exemplo anterior, dois filtros média foram utilizados, um com 100 pontos e outro com 400 pontos. Supõe-se agora o mesmo tipo de filtro, porém com 10 pontos. Observa-se que os coeficientes do filtro são todos iguais a $\frac{1}{10}$, ou seja, a soma de todos os coeficientes do filtro é igual a 1.

Para uma sequência de entrada $x(n)$, a saída do filtro média vai ser igual a

$$y(n) = \frac{1}{10}x(n-9) + \frac{1}{10}x(n-8) + \cdots + \frac{1}{10}x(n-1) + \frac{1}{10}x(n), \quad n \in \mathbb{Z}_+. \tag{24}$$

Tomando os coeficientes do filtro como $h(k)$, $k \in \mathbb{Z}_+$, tem-se

$$y(n) = h(9)x(n-9) + h(8)x(n-8) + \cdots + h(1)x(n-1) + h(0)x(n).$$

Generalizando, a n-ésima saída de um filtro com M pontos será dada pela Equação (25). Esta Equação é chamada de convolução (LYONS, 2011) e é aplicável em filtros digitais FIR.

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k). \tag{25}$$

A convolução é uma operação matemática e leva um símbolo específico ($*$). Então a Equação (25) pode ser reescrita pela Equação (26),

$$y(n) = h(k) * x(n). \tag{26}$$

Neste momento, é necessário voltar as atenções novamente à DFT — Transformada Discreta de Fourier. Isto porque existe uma estreita relação entre a DFT e convolução, como

mostra a Equação (27).

$$h(k) * x(n) \underset{\text{IDFT}}{\overset{\text{DFT}}{\longleftrightarrow}} H(m)X(m). \tag{27}$$

Em palavras, a DFT da convolução entre $h(n)$ e $x(n)$ é igual ao produto entre os espectros de frequência $H(m)$ e $X(m)$ e a IDFT do produto entre os espectros de frequência $H(m)$ e $X(m)$ é igual a convolução entre $h(n)$ e $x(n)$ (LYONS, 2011).

Os coeficientes $h(k)$ do filtro não necessariamente precisam ser iguais. É possível dar ganhos específicos a diferentes amostras, dependendo da aplicação.

3.5.2 Filtro passa-baixas

O filtro passa-baixas é amplamente utilizado em processamento de sinais. Sua principal função é atenuar altas frequências, enquanto mantém as baixas frequências com ganho unitário.

A arquitetura do filtro passa-baixas pode ser feita pelo método ótimo e pelo método de janelas (LYONS, 2011). Para este trabalho, será utilizado o método de janelas. Para tanto, os coeficientes do filtro que estarão na convolução com o sinal de entrada na Equação (26) não serão todos iguais, como no exemplo apresentado anteriormente.

Primeiramente, pode-se dizer que um filtro ideal é aquele que mantém o ganho das frequências interessantes inalterado e que dá ganho zero às frequências indesejáveis, ou seja, multiplica os valores por zero ou um.

Existe uma função trigonométrica que é capaz de fazer justamente isso, a função sinc, dada pela Equação (28).

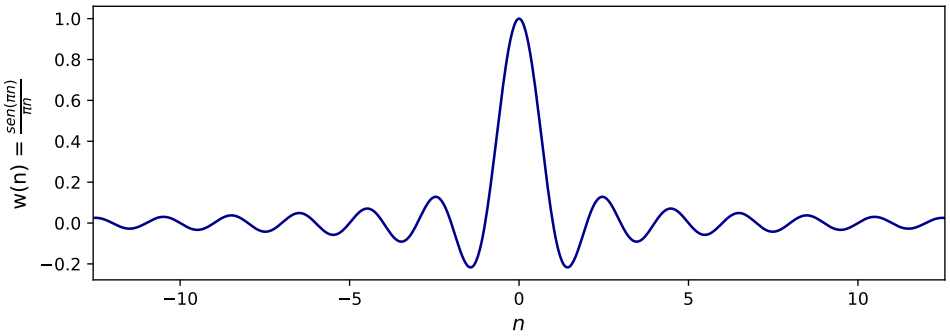
$$\operatorname{sinc} x = \frac{\operatorname{sen} x}{x}. \tag{28}$$

Porém, isso acontece somente quando se tem um número infinito de pontos amostrados no domínio do tempo. Como o mundo digital não é capaz disso, a função sinc acaba apresentando distorções em seu sinal, como mostra a Figura 12. Essas distorções são chamadas de lobos laterais e são causadas pelas descontinuidades apresentadas pela função sinc ao transicionar entre um e zero, periodicamente.

Outro problema pode ser observado analisando seu espectro de frequências (DFT). A Figura 13 mostra uma função sinc da forma $\frac{\operatorname{sen}(\pi x)}{\pi x}$ e seu espectro de frequências.

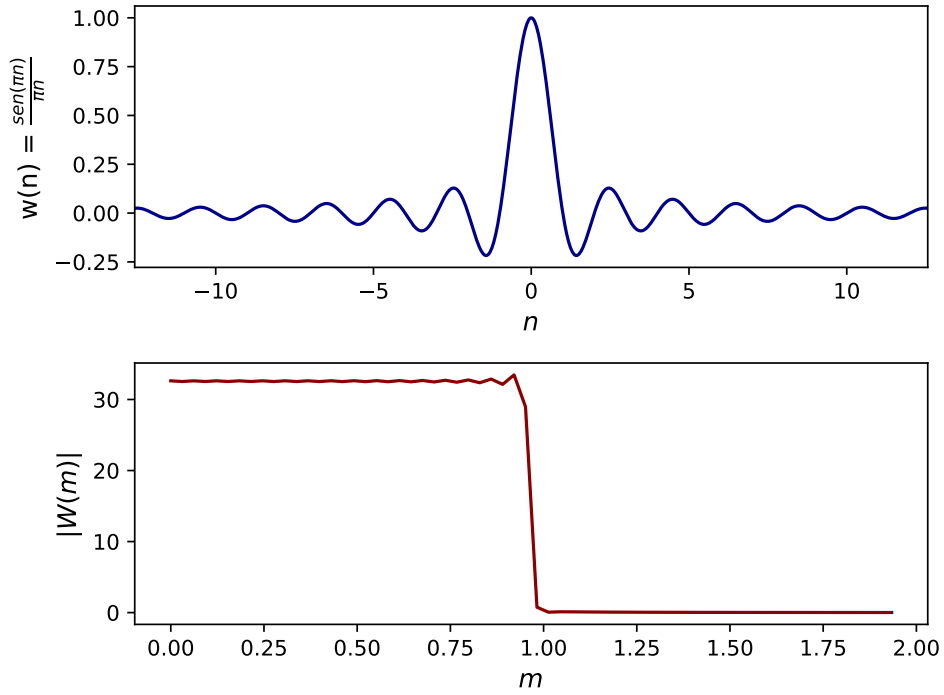
As ondulações localizadas no espectro de frequências, onde seu ganho é maior, são ocasionadas pelos lobos laterais sendo que ambos podem ser minimizados com o uso das janelas, aprimorando os coeficientes do filtro (LYONS, 2011, p. 188-189).

Figura 12 – A função sinc.



Fonte: o autor.

Figura 13 – Lobos laterais e ondulações no espectro de frequências.



Fonte: o autor.

Uma propriedade muito importante da convolução é utilizada para definir os coeficientes do filtro a ser utilizado.

Na Equação (27) vemos que a convolução no domínio do tempo equivale a multiplicação no domínio da frequência. Contudo, agora será usado o caminho inverso para os filtros, ou

seja, a multiplicação no domínio do tempo equivale a convolução no domínio da frequência (LYONS, 2011, p. 185). A Equação (29) mostra essa nova relação,

$$h(k)x(n) \underset{\text{IDFT}}{\overset{\text{DFT}}{\longrightarrow}} H(m) * X(m). \tag{29}$$

A janela tem como função truncar os termos da função sinc, fazendo com que a resposta em frequência do filtro esteja próxima da ideal e com o mínimo de ondulações possível. Desta forma, os coeficientes do filtro FIR nada mais são que a multiplicação entre os termos da função sinc truncada e os termos da janela. Logo, pela Equação (29), a resposta em frequência final do filtro será a convolução entre a janela e a função sinc no domínio da frequência, como mostra a Equação (30),

$$H(m) = H^{\infty} * W(m), \tag{30}$$

com H^{∞} sendo a resposta em frequência de uma função sinc infinitamente longa e $W(m)$ a resposta em frequência de uma janela (LYONS, 2011).

Neste sentido, é importante saber que:

> Podemos tornar a região de transição do filtro mais estreita usando coeficientes adicionais de filtro $h(k)$, mas não podemos eliminar a ondulação da banda passante. Essa ondulação, conhecida como fenômeno de Gibbs, se manifesta sempre que uma função ($w(k)$ neste caso) com uma descontinuidade instantânea é representada por uma série de Fourier (LYONS, 2011, p. 188).

4 Aplicação

A produção musical transformou-se em uma indústria milionária, trazendo grandes investimentos em pesquisas neste campo e visando a melhoria da qualidade do produto entregue ao consumidor de música.

Antigamente, por conta da limitação técnica dos equipamentos utilizados na gravação, o resultado final das músicas continha ruídos provenientes desta restrição. Com o passar do tempo, o montante de ruído reduziu-se gradativamente, devido ao avanço dos métodos de redução de ruído e filtragem de som (DE MARCHI, 2005).

O principal objetivo deste trabalho foi reduzir o ruído em uma música gravada na década de 1920, época esta em que as músicas eram gravadas em discos de sete polegadas e 78 rotações, feitos de um material chamado goma-laca. A duração (capacidade de armazenamento) desses discos era de aproximadamente três minutos e eram reproduzidos no gramofone, patenteado em 1887 por Emile Berliner (GOMES, 2014).

É verdade que a sofisticação trazida pelo Disco Compacto (CD) à industria musical é notável. "O fato, contudo, é que o gramofone revolucionou a música popular a tal ponto que podemos afirmar que a duração consagrada das canções populares, até hoje — em torno de três minutos — é resultado de seu advento" (GOMES, 2014, p. 76).

A música escolhida foi *Ash Tray Blues*, de Papa Charlie Jackson, gravada em 1928. A música fez parte do álbum "Vol. 2" e tem a duração de dois minutos e cinquenta e oito segundos. A Figura 14 mostra a capa do disco de Papa Charlie Jackson.

Figura 14 – Capa do albúm "Vol. 2", de Papa Charlie Jackson.

Fonte: `https://thedocumentrecordsstore.com/product/DOCD-5088/`.

A música foi comprada através da loja virtual *Document Records*[9] por US$ 0,99. A remasterização da música buscou conservar as características originais do som, gravado originalmente em um disco goma-laca.

[9] `https://thedocumentrecordsstore.com/product/DOCD-5088/`.

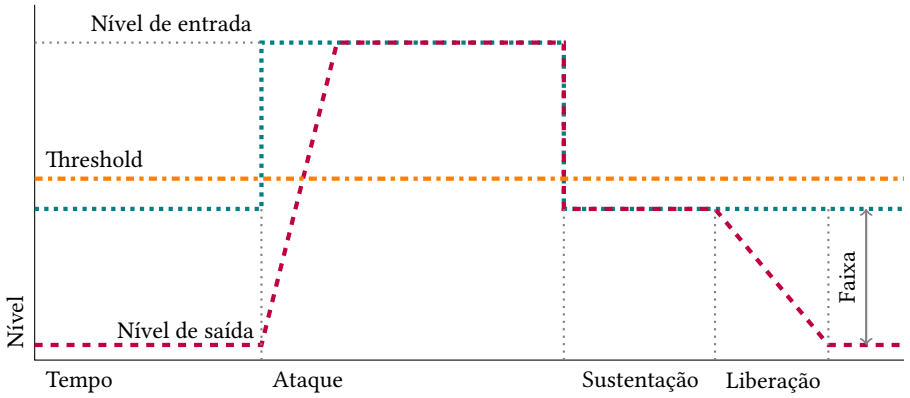
4.1 O método *noise gate*

Existem métodos de redução de ruído para cada tipo de aplicação. Por esse motivo, é importante que haja uma investigação acerca do objeto de estudo. Para o caso exposto neste trabalho, o método *noise gate* foi escolhido devido à garantia de uma redução significativa da quantidade de ruído contido em um sinal digital de áudio.

O método *noise gate* consiste na análise e comparação dos níveis de som do ruído e do sinal de entrada para que os componentes ruidosos desse sinal sejam atenuados. Através da magnitude média e desvio padrão (GOUVEIA, 2019) do ruído é criado um limiar de níveis de som. Quando a magnitude do sinal está acima desse limiar (conhecido como *threshold*) ele se mantém inalterado e quando está abaixo do *threshold* ele é atenuado.

Na prática, o método funciona como um portão, como o próprio nome sugere. O portão será fechado quando o nível de magnitude não ultrapassar o *threshold*. A Figura 15 mostra um desenho esquemático do método utilizado.

Figura 15 – Desenho esquemático do *noise gate*.



Fonte: adaptado de (BORGES, 2019).

Apesar da manipulação do sinal ocorrer no domínio da frequência, a saída se dá no domínio do tempo. Por essa razão, a Figura 15 mostra o nível de som (magnitude) do sinal em função do tempo (a linha pontilhada representa o sinal de entrada e a linha ponto-traço representa o sinal de saída do filtro).

O parâmetro *threshold* — representado por uma linha tracejada na Figura 15 — é responsável pela determinação do limiar de nível de som utilizado no algoritmo implementado, como já foi comentado.

O parâmetro *ataque* é utilizado para que não haja descontinuidade entre o fechar e abrir do portão, fazendo com que o processo seja suavizado.

Sustentação é o parâmetro que diz qual a tolerância de tempo que o sinal terá se ficar abaixo do *threshold*. Decorrido esse tempo de sustentação, o nível de som do sinal é atenuado sem que haja descontinuidade, sendo suavizado por um filtro controlado pelo parâmetro *liberação* (BORGES, 2019).

O principal ponto positivo do uso do *noise gate* é sua grande capacidade de eliminar sons indesejáveis que estão espalhados pelo sinal, não sendo possível sua remoção utilizando os filtros já mencionados na Seção 3.

A má notícia em relação ao *noise gate* é que seu uso causa danos ao sinal, pois existem tons puros (ou harmônicos) que acabam sendo atenuados juntamente com os ruídos, causando uma leve distorção. Quanto melhor forem ajustados os parâmetros escolhidos, maior será a qualidade do som entregue.

4.2 Audacity®

O Audacity® é muito utilizado por editores de áudio digital, tanto profissionais como amadores. O software tem uma ferramenta chamada *noise reduction*, que utiliza o método *noise gate* para reduzir os ruídos em um arquivo digital de áudio (AUDACITY, 2015). Esta ferramenta ajudou no desenvolvimento do trabalho, inicialmente como inspiração e posteriormente como apoio para comparar a eficácia do algoritmo implementado.

O software Audacity® também foi utilizado para a obtenção de um corte do arquivo de áudio contendo apenas ruído, requisito para que o método possa ser aplicado.

4.3 *Noise reduce*: o algoritmo de Tim Sainburg

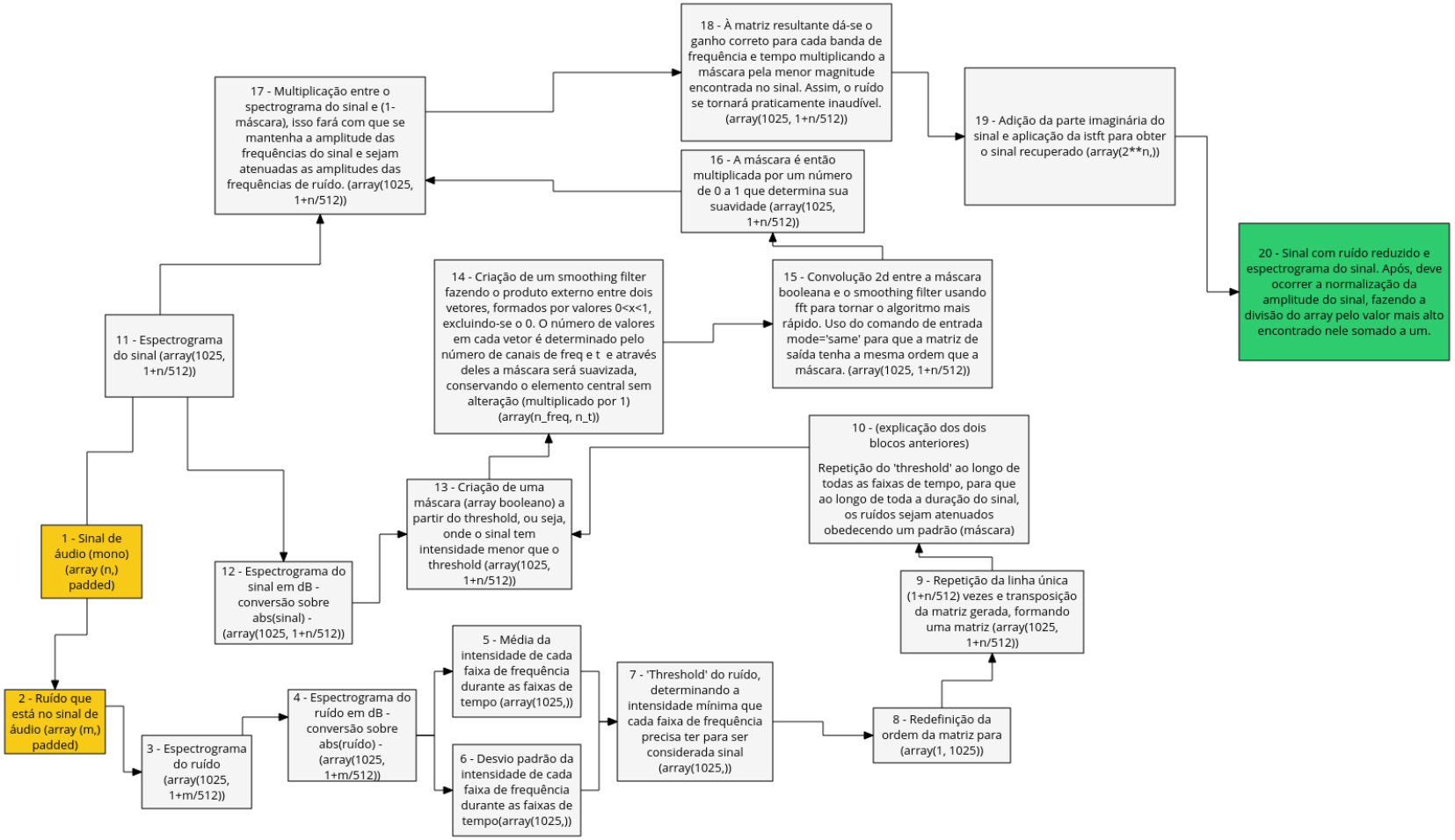
Tim Sainburg é um estudante PhD do Programa de Psicologia e Antropogenia do laboratório de neurofisiologia auditiva Tim Gentner na *University of California San Diego* (SAINBURG, 2018).

O algoritmo de Tim (SAINBURG, 2018) foi inspirado pela ferramenta de redução de ruído do software Audacity®. A linguagem de programação utilizada foi Python, fazendo-se uso de um pacote chamado Librosa (MCFEE et al., 2015) que apresenta muitas ferramentas de processamento de sinais para implementação com Python. A Figura 16 traz um diagrama de blocos do algoritmo de Tim, apresentando um resumo dos passos do algoritmo para executar a redução de ruído em um arquivo digital de áudio.

Para que o algoritmo possa ser aplicado, é necessário carregar um arquivo de áudio contendo o sinal com ruído (bloco 1 da Figura 16) e um segundo arquivo de áudio contendo

apenas um corte desse sinal ruidoso em que haja somente ruído (bloco 2 da Figura 16).

Figura 16 – Diagrama de blocos do algoritmo *noise reduce*.



Fonte: o autor.

O algoritmo foi analisado passo a passo para que fosse garantida sua eficácia na aplicação pretendida.

O algoritmo inicia executando uma FFT na amostra do sinal que contém somente ruído (a partir daqui será utilizado somente o termo "ruído" para referir-se a esta amostra). A FFT é executada por meio de uma ferramenta do Librosa chamada `stft` (bloco 3 da Figura 16).

4.3.1 STFT

A *Transformada de Fourier de Tempo Curto* (ou STFT[10]) é uma variação da DFT que tem como entrada um sinal em função do tempo e transforma-o em um sinal em função do tempo e da frequência, ou seja, dá-se ao sinal um tratamento bidimensional (SMITH, 2011).

O sinal pode estar ou não com um número 2^k, $k \in \mathbb{Z}_+$ de amostras. Caso não esteja, o usuário pode executar o *padding* que corresponde a completar o sinal com amostras nulas ou pode permanecer como está, já que o algoritmo faz uma espécie de arredondamento para que todas as amostras estejam em alguma banda de tempo e frequência.

A ferramenta STFT executa então uma FFT com um número N de pontos determinado pelo usuário. Como pretendeu-se uma resolução de aproximadamente 20 Hz e a taxa de amostragem utilizada foi de 44100 amostras por segundo, pela Equação (19), tem-se:

$$N = \frac{44100}{20} = 2205.$$

Sendo assim, a potência de 2 mais próxima é 2048 e, portanto, será executada uma FFT de 2048 pontos, resultando em 1025 bandas de frequência ($f = \frac{N}{2} + 1$), como mostrado na Página 31.

O número de bandas de tempo é determinado pela Equação (31)

$$t = \frac{n}{N/4} + 1, \tag{31}$$

sendo n o número de amostras no sinal. $\frac{N}{4}$ é o número de amostras que cada banda de tempo armazenará. O número t será ímpar, quando for executado o *padding* no sinal, ou seja,

$n = 2^k, \quad k \in \mathbb{N}$ (garantido pelo *padding*);

$N = 2^l, \quad l \in \mathbb{N}$ (número de pontos da DFT que é determinado pelo usuário);

$$\Rightarrow t = \frac{n}{N/4} + 1 = \frac{2^k}{2^l/4} + 1 = \frac{2^{k+2}}{2^l} + 1 = 2^{k-l+2} + 1 = 2 \times 2^{k-l+1} + 1.$$

[10] *Short-Time Fourier Transform*, em inglês.

Tomando $2^{k-l+1} = m$,

$$t = 2m + 1,$$

logo, t é ímpar.

Assim, pode-se também afirmar que f é ímpar, pois

$$f = \frac{N}{2} + 1 = \frac{2^l}{2} + 1 = 2^{l-1} + 1 = 2 \times 2^{l-2} + 1.$$

Tomando $2^{l-2} = s$,

$$f = 2s + 1,$$

logo, f é ímpar.

O número de quadros de tempo t poderá ser par caso não seja executado o *padding* no sinal e se tenha t na forma

$$t = 2k + \frac{p}{q}, \qquad k, p \text{ e } q \in \mathbb{Z}_+^*.$$

Se esse for o caso, o algoritmo tomará $\frac{p}{q} = 0$ e assim

$$t = 2k,$$

e portanto, nesse caso, t será par.

É executada ainda uma janela de Hanning sobre cada banda de tempo para a curva ser suavizada.

O processo de passar um sinal pela STFT resulta em uma matriz A, como mostra a Equação (32), sendo $a_{(f,t)} \in \mathbb{C}$ cada uma das amostras,

$$A = \begin{bmatrix} a_{(1,1)} & a_{(1,2)} & \cdots & a_{(1,\frac{n}{N/4}+1)} \\ a_{(2,1)} & a_{(2,2)} & \cdots & a_{(2,\frac{n}{N/4}+1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{(\frac{N}{2}+1,1)} & a_{(\frac{N}{2}+1,2)} & \cdots & a_{(\frac{N}{2}+1,\frac{n}{N/4}+1)} \end{bmatrix}. \tag{32}$$

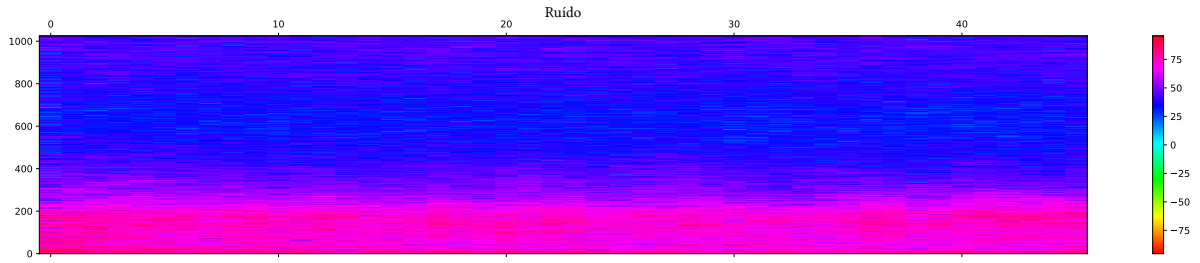
Para as manipulações do sinal através do *noise reduce*, são utilizadas as magnitudes do sinal. As magnitudes são obtidas com a ferramenta `abs` do pacote NumPy (VAN DER WALT; COLBERT; VAROQUAUX, 2011). Desta forma, `numpy.abs(`$a[f,t]$`)` é a magnitude do *bin* de frequência f no quadro de tempo t.

Para visualização da magnitude em decibéis, utilizou-se uma ferramenta do pacote Librosa chamada `core.amplitude_to_dB` (bloco 4 da Figura 16).

Como resultado visual da STFT, é obtido um espectrograma (SMITH, 2011) do sinal carregado.

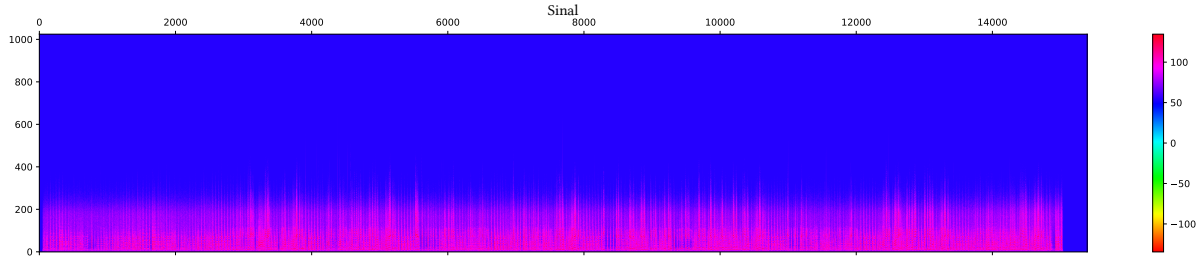
A Figura 17 mostra o espectrograma do ruído da música *Ash Tray Blues* e a Figura 18 mostra o espectrograma da música *Ash Tray Blues* contendo ruído.

Figura 17 – Espectrograma obtido pela STFT do ruído.



Fonte: O autor.

Figura 18 – Espectrograma obtido pela STFT do sinal com ruído.



Fonte: O autor.

Esses espectrogramas mostram as bandas de frequência nas linhas e as bandas de tempo nas colunas. A magnitude do sinal é representada por diferentes cores e os números na barra à direita do espectrograma são as magnitudes do sinal em decibéis.

Após a execução da STFT sobre o ruído, alguns parâmetros foram analisados.

4.3.2 Magnitude média

O objetivo é estabelecer um padrão para a redução do ruído em todas as bandas de frequência, durante todos os quadros de tempo da música. Para isso, se a magnitude das frequências que estão presentes no ruído fossem atenuadas sem que houvesse um padrão, provavelmente a música sofreria sérios danos, pois partes importantes do sinal podem estar emaranhadas com o ruído.

Para que haja o mínimo de dano, combinado com o máximo de redução de ruído possível, é preciso estudar os parâmetros do ruído. Um deles é a magnitude média. Para seu cálculo, basta utilizar a ferramenta `mean` do pacote NumPy sobre as magnitudes (em decibéis) da matriz obtida (bloco 5 da Figura 16).

Assim, é calculada a média das magnitudes de cada uma das bandas de frequência ao longo dos quadros de tempo, resultando em uma matriz coluna de ordem 1025.

4.3.3 Desvio padrão

Calcular o desvio padrão da magnitude do sinal em cada banda de frequência ao longo dos quadros de tempo (bloco 6 da Figura 16) permite traçar melhor o comportamento do ruído, uma vez que se o desvio padrão for muito alto, o dano causado à música pela redução do ruído será maior do que o desejado.

A ferramenta `std` do pacote NumPy é responsável por calcular o desvio padrão sobre o ruído, produzindo uma matriz coluna de ordem 1025.

4.3.4 Threshold

Threshold é um dos pontos mais importantes para a eficácia da implementação. Ele é o limiar do ruído, ou seja, é o que separa o que é considerado ruído do que é considerado sinal limpo.

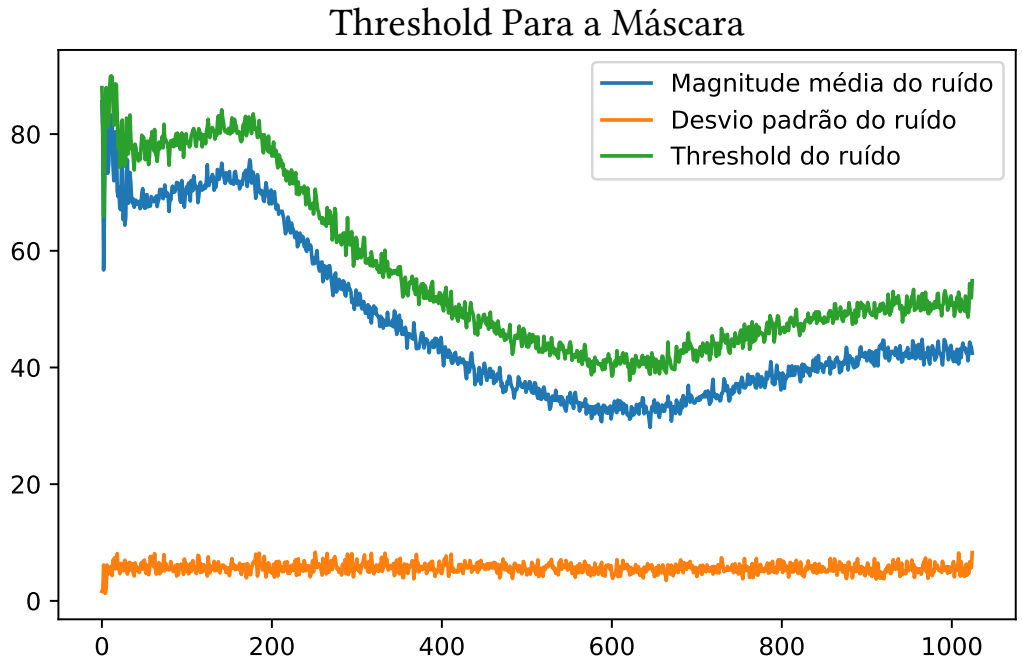
Para calcular o *threshold* (bloco 7 da Figura 16), o vetor magnitude média é somado ao produto entre o vetor desvio padrão e um escalar que é determinado pelo usuário na entrada do algoritmo (o `n_std_thresh`). Este número determina o excedente do *threshold*, ou seja, aumenta o nível de intensidade (em dB) que será considerado ruído. O padrão para o `n_std_thresh` é 1,5, ou seja, 150% do desvio padrão é somado à magnitude média.

Quanto mais próximo de 1 o `n_std_thresh` for, maior será o número de amostras consideradas ruído. A partir de 2, o dano causado ao sinal é muito grande, pois boa parte dele será considerado ruído e consequentemente terá seus níveis atenuados.

Se `n_std_thresh` < 1 algumas amostras que já foram consideradas ruído não serão atenuadas, ou seja, ainda farão parte do sinal, contaminando-o.

A Figura 19 mostra o espectro de frequências da magnitude média, desvio padrão e *threshold* para o ruído. O eixo horizontal apresenta as bandas de frequência e o eixo vertical apresenta os níveis do ruído em decibéis.

Figura 19 – Espectro de frequências da magnitude média, desvio padrão e *threshold* do ruído.



Fonte: O autor.

Após esse cálculo, o vetor gerado é transposto e repetido nas linhas abaixo até que se tenha $\frac{n}{N/4} + 1$ linhas. Logo em seguida, a matriz resultante é transposta, tendo a mesma forma da matriz A da Equação (32).

Esse processo é feito para que se tenha o mesmo *threshold* em todos os quadros de tempo, mantendo o controle sobre o ruído durante toda a duração da música (blocos 8, 9 e 10 da Figura 16).

A partir da matriz que contém as magnitudes do sinal em decibéis (blocos 11 e 12 da Figura 16) e do *threshold* que acaba de ser obtido, é criada uma máscara para o sinal.

4.3.5 Máscara

Em processamento de sinais, *máscara* é uma matriz booleana[11] que pode ser manipulada e filtrada para atender ao que se necessita. Para o *noise reduce* a condição aplicada à máscara é "quais amostras do espectrograma do sinal tem magnitude menor que o *threshold*?"

[11] A matriz booleana é aquela cujos elementos pertencem ao conjunto $\{0, 1\}$, sendo que — obedecendo a algum critério determinado — 0 representa *Falso* e 1 representa *Verdadeiro*.

Às amostras cuja resposta à condição for *Verdadeiro*, será designado o valor 1. Caso contrário, será designado o valor 0 (bloco 13 da Figura 16).

Para que não haja descontinuidade no sinal quando for executado analogicamente, a máscara é filtrada por um filtro de suavização.

4.3.6 Filtro de suavização

Filtro de suavização (ou *smoothing filter*, em inglês, como aparece no bloco 14 da Figura 16) já foi mencionado nesse trabalho, quando discutiu-se sobre os filtros digitais. Em especial, o filtro média aritmética, apresentado na Página 42. Naquele exemplo, os dez coeficientes do filtro tinham pesos iguais, como mostra a Equação (24), ou seja,

$$y(n) = \frac{1 \cdot x(n-9) + 1 \cdot x(n-8) + \cdots + 1 \cdot x(n-1) + 1 \cdot x(n)}{10}, \qquad n \in \mathbb{Z}_+.$$

Mas, como mostra a Equação (25), os coeficientes do filtro podem ter pesos diferentes, desde que sua soma seja igual a 1 para que o resultado final tenha ganho unitário sobre o sinal filtrado, ou seja, para que o sinal não seja distorcido (O'HAVER, 1997).

Desta forma, pode-se chegar a um filtro média (ponderada), por exemplo, como mostra a Equação (33),

$$y(n) = \frac{1 \cdot x(n-4) + 2 \cdot x(n-3) + 3 \cdot x(n-2) + 2 \cdot x(n-1) + 1 \cdot x(n)}{9}, \quad n \in \mathbb{Z}_+. \tag{33}$$

É fácil observar que a soma dos coeficientes do filtro será

$$\frac{1}{9} + \frac{2}{9} + \frac{3}{9} + \frac{2}{9} + \frac{1}{9} = \frac{9}{9} = 1.$$

Para o *noise reduce* (bloco 14 da Figura 16), o filtro de suavização segue parâmetros determinados pelo usuário. As entradas `n_grad_freq` e `n_grad_time` determinam quantas bandas de frequência e quantas bandas de tempo, respectivamente, o filtro suavizará em torno da amostra em que o filtro está agindo. O padrão do algoritmo é `n_grad_freq` = 2 e `n_grad_time` = 4.

É conveniente que se tenha um número ímpar de coeficientes para que a amostra em que o filtro está agindo fique centralizada. Desta forma, o filtro de suavização terá 3 coeficientes na frequência e 5 coeficientes no tempo.

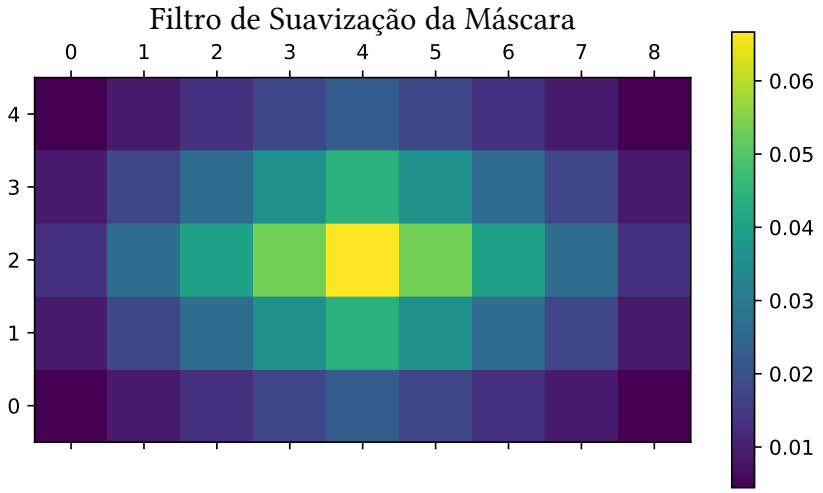
O *noise reduce* constrói o filtro de suavização em duas dimensões (frequência e tempo) a partir do cálculo do produto externo entre dois vetores. Esses vetores são

$$\vec{f} = \left(\frac{1}{\texttt{n\_grad\_freq+1}}, \frac{2}{\texttt{n\_grad\_freq+1}}, \ldots, 1, \ldots, \frac{2}{\texttt{n\_grad\_freq+1}}, \frac{1}{\texttt{n\_grad\_freq+1}} \right),$$
$$\vec{t} = \left(\frac{1}{\texttt{n\_grad\_time+1}}, \frac{2}{\texttt{n\_grad\_time+1}}, \ldots, 1, \ldots, \frac{2}{\texttt{n\_grad\_time+1}}, \frac{1}{\texttt{n\_grad\_time+1}} \right).$$

Como o ganho do filtro deve ser unitário, os coeficientes da matriz gerada devem ser divididos pela soma de todos os coeficientes da matriz.

Com isso, está pronto o filtro de suavização. A Figura 20 mostra o filtro de suavização obtido para aplicação na música *Ash Tray Blues*. Para que o filtro de suavização seja aplicado

Figura 20 – Espectro de intensidade do filtro de suavização obtido para a redução do ruído.



Fonte: O autor.

na máscara que foi criada, faz-se uma convolução bidimensional entre a máscara e o filtro.

4.3.7 Convolução bidimensional

Se a convolução em uma dimensão faz com que um sinal passe pelo filtro, suavizando o sinal no domínio do tempo ou da frequência, a convolução bidimensional faz os dois ao mesmo tempo (O'HAVER, 1997). O filtro agirá sobre cada amostra do sinal — elementos da matriz da Equação (32) — suavizando as amostras ao seu redor e mantendo um ganho unitário para a amostra central.

Partindo da Equação (25) e da Equação (26), a convolução bidimensional será dada pela Equação (34), sendo $y(f,t)$ a máscara após a convolução, $h(u,v)$ a sequência que contém os coeficientes do filtro de suavização e $x(f,t)$ a máscara antes da suavização,

$$y(f,t) = h(u,v) * x(f,t) \Rightarrow y(f,t) = \sum_{v=0}^{M-1} \sum_{u=0}^{N-1} h(u,v)x(f-u,t-v), \tag{34}$$

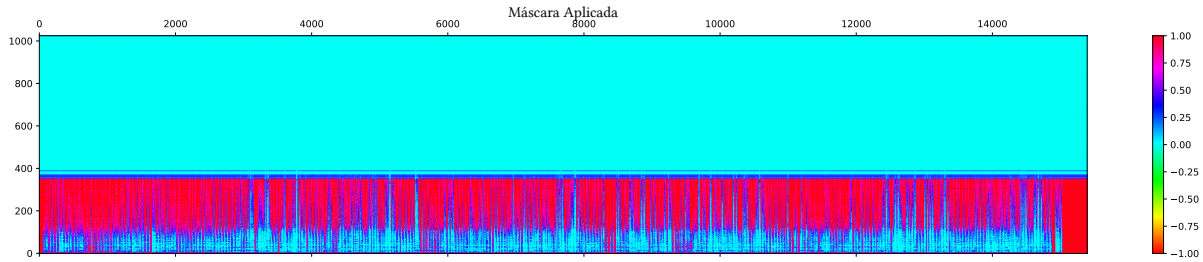
e é sobre o processo de convolução bidimensional que o bloco 15 da Figura 16 trata.

Como as dimensões do filtro são menores que as dimensões da máscara, a matriz resultante, ou seja, a máscara suavizada terá as mesmas dimensões da máscara antes da convolução bidimensional.

A máscara ainda é multiplicada por um fator (determinado pelo usuário) entre 0 e 1 que determina sua sensibilidade (bloco 16 da Figura 16). Para mascarar o sinal por completo (o que é mais indicado), deve-se multiplicar por 1, mantendo o ganho da máscara em 100%. Se multiplicar por 0, a máscara não surtirá efeito algum.

A Figura 21 mostra o resultado final da máscara desenvolvida para a redução do ruído na música *Ash Tray Blues*. Resta aplicar a máscara sobre a STFT do sinal para que as frequências

Figura 21 – Espectrograma da máscara que aplicou-se ao sinal.



Fonte: O autor.

ruidosas sejam atenuadas ao longo dos quadros de tempo.

4.3.8 Aplicação da máscara

A aplicação da máscara sobre o sinal (bloco 17 da Figura 16) é um dos últimos passos do algoritmo *noise reduce*. Para tanto, a STFT do sinal — que carrega as magnitudes em decibéis em função do tempo e da frequência — é multiplicada por $1 - y(f,t)$.

Essa diferença é muito importante pois carrega a informação do ganho exato que se atribuirá a cada amostra do sinal. A máscara que foi criada (Página 55) atribuiu valor 0

para as amostras que tivessem magnitude maior que o *threshold*. Essas amostras devem ser consideradas sinal e devem permanecer inalteradas.

Como neste passo faz-se a diferença $1 - y(f,t)$,

$$1 - y_i(f,t) = 1 - 0 = 1,\ \forall\, y_i(f,t) = 0.$$

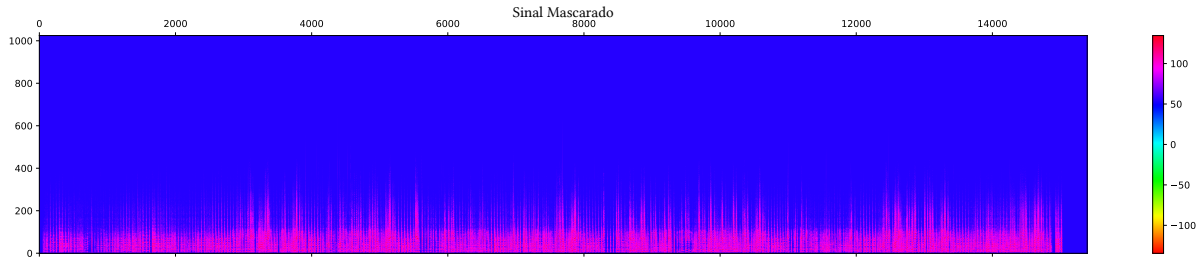
Em contra partida, para as amostras cuja magnitude for menor que o *threshold*, será dado um ganho menor que 1 e maior que 0, ou seja, a magnitude será reduzida em até 100%.

Como o *noise reduce* trabalha com as magnitudes do sinal, a redução do ruído será otimizada somando o sinal à máscara $y(f,t)$ multiplicada pela menor magnitude encontrada no sinal (bloco 18 da Figura 16).

Mais uma vez as amostras que tiverem magnitude maior que o *threshold* não sofrerão alteração, pois nesse caso, $y(f,t) = 0$. Caso contrário, a amostra sofrerá uma drástica redução em sua magnitude (em decibéis).

Após a aplicação da máscara, o espectrograma do sinal tornou-se mais "limpo", como mostra a Figura 22.

Figura 22 – Espectrograma do sinal após a aplicação da máscara.



Fonte: O autor.

4.3.9 Saída do algoritmo

Após a aplicação da máscara na parte real do sinal, faz-se o mesmo para a parte imaginária, aplicando a ferramenta `imag` do pacote NumPy sobre as amplitudes do sinal (não sua magnitude, como feito até agora) e multiplicando-a por $1 - y(f,t)$.

As partes real e imaginária são somadas para a reconstrução do sinal em amplitude. Para que isso possa acontecer, a parte real é convertida para amplitude pela ferramenta `core.db_to_amplitude` do Librosa e multiplicada por uma matriz de sinais[12], fazendo com

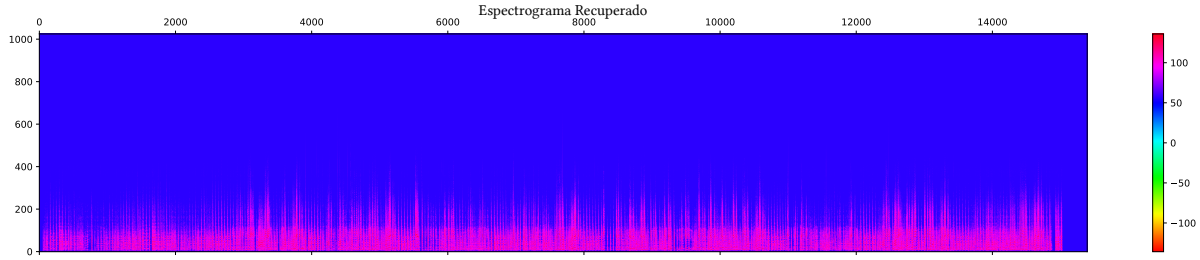
[12] Dada uma STFT $x(n,m)$, a ferramenta `sign` do pacote NumPy permite atribuir valor 1 quando $x_i(n,m) > 0$, 0 quando $x_i(n,m) = 0$ e -1 quando $x_i(n,m) < 0$.

que as amostras que antes de passarem pela ferramenta `abs` eram negativas voltem a sê-lo. A parte imaginária é multiplicada por $1j$ (sintaxe em Python para que um número seja considerado imaginário) e somada à parte real.

Como o sinal ainda está em função do tempo e da frequência graças a STFT, ele deve passar pela ferramenta `istft` do Librosa que executa a Transformada de Fourier de Tempo-Curto Inversa e assim, o sinal volta a estar em função do tempo.

O *noise reduce* ainda produz mais um espectrograma do sinal após ser recuperado (bloco 19 da Figura 16). A Figura 23 mostra o resultado final obtido.

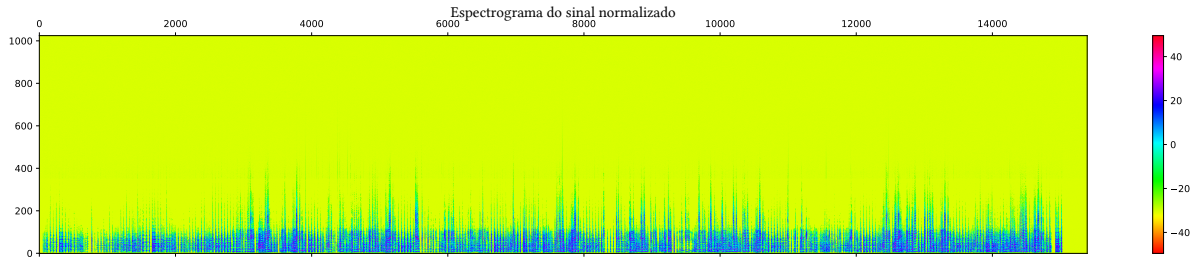
Figura 23 – Espectrograma do sinal recuperado.



Fonte: O autor.

Para que o sinal obtido possa ser salvo e reproduzido sem distorções, sua amplitude deve ser normalizada, de modo que cada amostra do sinal fique entre -1 e 1, fazendo com que quando ela for convertida de digital para analógica, não afete a audição humana. Para tanto, basta dividir o sinal pela amostra de maior valor somada ao número 1 (bloco 20 da Figura 16). A Figura 24 mostra o espectrograma do sinal normalizado.

Figura 24 – Espectrograma do sinal normalizado.



Fonte: O autor.

Esta foi a aplicação do algoritmo de Tim Sainburg para redução de ruído na música *Ash Tray Blues*, de Papa Charlie Jackson. Tanto o método, como o algoritmo utilizado nesta

aplicação são aplicáveis a uma série de outros sinais digitais de áudio, como em discursos, por exemplo. No Apêndice A é apresentada a implementação Python completa do algoritmo, contendo comentários sobre seu desenvolvimento.

4.4 Testes de desempenho

O algoritmo *noise reduce* solicita ao usuário 10 entradas para que o algoritmo seja executado, sendo apenas duas delas obrigatórias. Oito entradas apresentam um valor padrão, caso o usuário não deseje fazer alterações nesses parâmetros. A importância desses parâmetros de entrada está no desempenho do algoritmo, que pode pode sofrer alterações devido às entradas recebidas.

Para os testes de desempenho do algoritmo, utilizou-se um sinal contendo 5 segundos de uma nota Lá (440Hz)[13]. Optou-se por realizar os testes em um sinal mais simples para que as mudanças fossem apuradas com maior clareza e controle, uma vez que o sinal original não possuía ruído. O ruído foi adicionado através da ferramenta `generate_noise.band_limited_noise`, do mesmo pacote que contém o *noise reduce*, ambos produzidos por Tim Sainburg.

O ruído adicionado ao sinal tem uma banda de frequência entre 2kHz e 12kHz.

O principal ponto observado foi o tempo gasto para que o algoritmo fosse rodado em um computador comum. Para que o tempo fosse marcado, utilizou-se o pacote `time`, do Python.

Foram executados três tipos de testes: análise em relação ao tamanho das janelas e da FFT — ou seja, as entradas `n_fft`, `hop_length` e `win_length`; análise em relação às bandas afetadas pelo filtro de suavização — ou seja, as entradas `n_grad_freq` e `n_grad_time` — e análise em relação à suavidade do *threshold* (`n_std_thresh`).

Para cada um dos testes, o algoritmo foi aplicado 5 vezes para que fosse possível calcular a média dos tempos de execução com os parâmetros selecionados.

Como referência, executou-se o algoritmo com todas as entradas padrão. O tempo médio de execução foi de 0,368570s e observou-se que o ruído foi quase que totalmente retirado. Houve um dano pequeno no sinal, mas que não prejudicou sua clareza.

4.4.1 Teste 1: tamanho das janelas e FFT

No primeiro teste, foram analisadas as entradas `n_fft`, `hop_length` e `win_length`. As entradas padrão são 2048, 512 e 2048, respectivamente.

[13] Retirado de `https://www.mediacollege.com/audio/tone/files/440Hz_44100Hz_16bit_05sec.wav`.

Primeiramente optou-se por números pequenos:

$$\texttt{n\_fft} = 4;\ \texttt{hop\_length} = 1;\ \texttt{win\_length} = 4.$$

Essa escolha foi bem ruim, uma vez que a resolução da FFT precisa ser estreita e uma FFT com apenas 4 pontos resulta em bandas de frequência muito largas, deixando o algoritmo sem precisão.

A média de tempo para execução do algoritmo foi de $6,700489$s. O som ficou bastante ruidoso, tendo um péssimo resultado.

Após, realizou-se o teste com números grandes, bem maiores do que o padrão:

$$\texttt{n\_fft} = 2^{15};\ \texttt{hop\_length} = 2^{13};\ \texttt{win\_length} = 2^{15}.$$

A média de tempo para execução do algoritmo foi de $0,457351$s. O som ficou mais ruidoso que o padrão, apesar de estar menos ruidoso que na primeira tentativa.

Na terceira tentativa, optou-se por

$$\texttt{n\_fft} = 1024;\ \texttt{hop\_length} = 256;\ \texttt{win\_length} = 1024.$$

A média de tempo para execução do algoritmo foi de $0,362969$s. O som ficou mais ruidoso que o padrão, apesar de ser menos ruidoso que na primeira e segunda tentativas.

4.4.2 Teste 2: Bandas de frequência e tempo

No segundo teste, analisou-se as duas entradas que determinam como se dá a construção do filtro de suavização, que interfere diretamente na máscara que é aplicada ao sinal. Consequentemente, as entradas `n_grad_freq` e `n_grad_time` são determinantes para a atenuação dos níveis de ruído no sinal. Os valores padrões para `n_grad_freq` e `n_grad_time` são 2 e 4, respectivamente.

Na primeira tentativa, optou-se por zerar ambas as entradas, ou seja, fazer com que não haja suavização nas bandas de frequência, nem nos quadros de tempo.

A média de tempo para execução do algoritmo foi de $0,372231$s. O som ficou mais ruidoso que o padrão.

Na segunda tentativa, aplicou-se o valor 10 para ambas as entradas, fazendo com que houvesse um número grande de bandas de frequência e quadros de tempo suavizados.

A média de tempo para execução do algoritmo foi de $0,376071$s. O som ficou mais ruidoso que o padrão e menos que na tentativa anterior. O sinal filtrado apresentou danos maiores que o padrão.

Na terceira tentativa, foram testados valores próximos ao padrão: `n_grad_freq` = 4 e `n_grad_time` = 6, resultando em uma média de tempo de execução do algoritmo de 0,370614s e produzindo um som muito próximo do padrão.

4.4.3 Teste 3: Suavidade do threshold

Para o terceiro teste, observou-se a mudança no comportamento do algoritmo quando alterados os padrões do *threshold* do sinal. A entrada padrão do `n_std_thresh` é de 1,5.

Foram executadas duas tentativas. Na primeira, atribuiu-se o valor zero para o parâmetro analisado, fazendo com que o *threshold* ficasse exatamente a uma distância (da magnitude média do ruído) igual ao desvio padrão de cada banda de frequência durante cada quadro de tempo.

Como resultado, obteve-se um sinal mais ruidoso que o padrão e com danos. A média do tempo de execução do algoritmo foi de 0,369244s.

Na segunda tentativa, utilizou-se um valor bem grande para o `n_std_thresh`: 50. Isso fez com que o *threshold* ficasse muito acima da magnitude média do ruído.

O teste resultou em um sinal extremamente ruidoso, não sendo possível identificar o sinal original em meio ao ruído. O tempo de execução médio do algoritmo foi de 0,369244s.

Na Seção 5 são apresentados os resultados dos testes de desempenho realizados e no Apêndice B é apresentado um algoritmo elaborado pelo autor para o cálculo do SNR.

4.5 Relação Sinal Ruíso (SNR)

Para avaliar a qualidade do som filtrado da música *Ash Tray Blues*, utilizou-se uma métrica chamada *relação sinal-ruído* (ou SNR[14]). O SNR é a razão entre a potência do sinal livre de ruído e a potência do ruído (LYONS, 2011, p. 761), como mostra a Equação (35).

$$\text{SNR} = \frac{\text{Potência do sinal}}{\text{Potência do ruído}}. \tag{35}$$

Como o SNR pode variar em muitas ordens de magnitude, normalmente utiliza-se o decibel como unidade de medida. Assim,

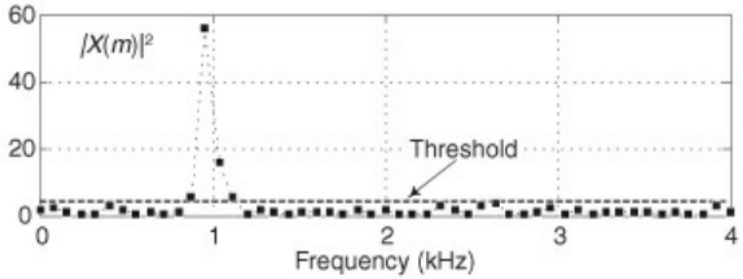
$$\text{SNR}_{\text{dB}} = 10\log_{10}(\text{SNR})\,\text{dB}. \tag{36}$$

No caso estudado, o ruído estava misturado ao sinal, não sendo possível o cálculo da potência do sinal e do ruído, separadamente.

[14] Signal-to-noise ratio, em inglês.

Para que haja a determinação do SNR do sinal estudado, é preciso que se analise seu espectro de frequências. A Figura 25 mostra um exemplo de sinal ruidoso sendo analisado para o cálculo do SNR.

Figura 25 – Espectro de frequências para determinação do SNR.



Fonte: (LYONS, 2011, p. 763).

Primeiramente calcula-se a DFT sobre o sinal desejado e, em seguida, o quadrado de suas magnitudes ($|X(m)|^2$). O *threshold* que aparece na Figura 25 é muito parecido com o utilizado no *noise reduce* (Página 54). Porém, esse *threshold* não sofrerá interferência do `n_std_thresh`, ou seja, `n_std_thresh` = 1.

Desta forma,

$$\textit{Threshold} = \text{Média do quadrado das magnitudes + Desvio padrão.} \tag{37}$$

Como o *threshold* é um número e não um vetor, com o auxilio do Python e do pacote NumPy, é possível repetir esse valor, produzindo um vetor de mesmo tamanho que o sinal analisado.

O SNR será calculado a partir da soma dos quadrados das magnitudes das amostras que estão acima do *threshold* sobre a soma dos quadrados das magnitudes das amostras que estão abaixo do *threshold*, como mostra a Equação (38).

$$\text{SNR} = \frac{\text{Soma das amostras } |X(m)|^2 \text{ acima do } \textit{threshold}}{\text{Soma das amostras } |X(m)|^2 \text{ abaixo do } \textit{threshold}}. \tag{38}$$

Desta forma, quanto menor a soma das amostras $|X(m)|^2$ abaixo do *threshold*, maior será o SNR e, consequentemente, o sinal será considerado menos ruidoso, com uma qualidade maior.

A seguir, são apresentados os resultados e discussões acerca do trabalho realizado, inclusive o SNR do sinal original e do sinal filtrado.

5 Resultados

O principal objetivo deste trabalho foi reduzir o ruído de fundo da música *Ash Tray Blues*, de Papa Charlie Jackson, gravada no ano de 1928.

Para tanto, utilizou-se o algoritmo *noise reduce*, de Tim Sainburg, baseado na ferramenta *noise reduction* do software Audacity® que por sua vez, foi desenvolvido com base no método *noise gate*, que determina um limiar de níveis de som para que o que ficar abaixo desse limiar seja considerado ruído e posteriormente tenha sua magnitude atenuada.

Chegou-se a conclusão de que a utilização de um método como o *noise gate* seria eficaz para o caso estudado após algumas tentativas com outros métodos, tais como o filtro média, filtro passa-baixas, filtro passa-altas[15], janelamento e suavização.

A Figura 18 mostra que a intensidade do sinal está concentrada até a 200ª banda de frequência, chegando até a 550ª banda — aproximadamente — com magnitudes acima de 60dB, de um total de 1025 bandas de frequência. Isso mostra que o ruído estava embuído no espectro de frequências do sinal e explica o porquê de filtros como o passa-baixas e o passa-altas não surtirem efeito.

De fato, quando usado, o filtro passa-baixas atenua os níveis de som de altas frequências a partir de um limiar determinado (o *cutoff*). Neste caso, tanto o ruído quanto a música são sinais de baixa frequência. Logo eles iriam continuar misturados e, consequentemente, o som continuaria ruidoso.

O mesmo vale para o filtro passa-altas. O sinal é composto em grande parte das frequências que seriam atenuadas, trazendo um prejuízo muito grande ao resultado final.

Os filtros média, janelamento e suavização se complementam. Porém, sem haver um nível de sofisticação mínimo, esses filtros acabariam por danificar o sinal, ainda que fosse perceptível uma leve queda nos níveis de ruído presente na música.

O *noise reduce* mostrou-se muito eficaz ao fazer uso do *noise gate*, pois a análise do sinal e sua manipulação foram executadas visando uma baixa nas magnitudes das frequências indesejáveis e não um tratamento direto nas bandas de frequência.

A STFT permitiu que o sinal fosse manipulado nos domínios da frequência e do tempo simultaneamente, possibilitando uma melhor visualização com os espectrogramas e uma análise mais apurada das bandas de frequência ao longo dos quadros de tempo determinados.

Ao realizar testes de desempenho do algoritmo *noise reduce*, foi possível constatar que

[15] Assim como o passa-baixas, o filtro passa-altas é do tipo FIR e, ao contrário do passa-baixas, permite que altas frequências passem por ele sem sofrer atenuação, enquanto as baixas frequências tem seus níveis de som atenuados.

os parâmetros de entrada utilizados para a aplicação na música *Ash Tray Blues* foram determinantes para o sucesso obtido na redução do ruído presente antes da filtragem.

Os parâmetros de entrada com valores padrões apresentaram a melhor qualidade de som com o menor tempo médio de execução do algoritmo. Uma das duas entradas obrigatórias exigidas pelo algoritmo é um sinal contendo apenas ruído. Como a música não tem grandes intervalos de tempo contendo ruído de forma isolada, definir essa entrada foi um grande obstáculo.

Com o auxilio da ferramenta de corte do software Audacity®, foi possível extrair dois trechos contendo apenas ruído, com 0,53s cada um. Esses trechos são muito pequenos quando comparados com a duração da música, que é de 2,58min. Ainda assim, foi possível gerar um resultado significativo e positivo.

Para quantizar a qualidade da música antes e após a redução de ruído, utilizou-se a métrica SNR (Subseção 4.5). O cálculo do SNR foi realizado utilizando-se a linguagem de programação Python, com o auxilio dos pacotes NumPy e Librosa.

Utilizou-se como referência de qualidade de redução de ruído uma faixa da música *Ash Tray Blues* após passar pelo *noise reduction*, do software Audacity®. A Tabela 7 mostra o resultado do método utilizado em comparação com a música original e com a utilização do software Audacity®.

Tabela 7 – SNR dos resultados obtidos.

Sinal de entrada — *Ash Tray Blues*	SNR (dB)
Versão original de gravação	6,31
Filtrada pelo Audacity®	6,827
Filtrada pelo *noise reduce*	6,972
Filtrada pelo *noise reduce**	7,1

A última linha da Tabela 7 refere-se ao sinal obtido pela aplicação do *noise reduce* sobre o sinal filtrado anteriormente — tendo como referência o primeiro trecho de ruído — e agora filtrado novamente, mas em relação ao segundo trecho de ruído.

A música filtrada pela ferramenta *noise reduction* do software Audacity® obteve um aumento de 8,19% do SNR, enquanto o algoritmo *noise reduce*, de Tim Sainburg conseguiu um aumento de 12,52% do SNR.

Assim sendo, o *noise reduce* fornece resultados significativos e até mais positivos que o software Audacity® (para o caso desta aplicação).

A relação custo-benefício dessa aplicação é positiva, uma vez que um computador comum pode executar o algoritmo em poucos segundos e obter um resultado satisfatório na redução de ruídos.

Esse algoritmo pode ajudar muito em gravações que não são musicais também, como em um discurso ou uma entrevista, por exemplo, desde que se tenha alguns segundos de gravação contendo o som ambiente (apenas ruído de fundo presente).

6 Considerações finais

Neste trabalho, procurou-se fazer um estudo acerca do processamento digital de sinais, que teve seu início há quatro mil e quinhentos anos, no Egito.

Dentro do processamento digital de sinais, o francês Jean Baptiste Joseph Fourier fez uma revolução, mesmo sem ter percebido. Após seus estudos acerca da condução de calor, consolidaram-se as séries que levam seu nome, fato que possibilitou o desenvolvimento das diversas Transformadas de Fourier e, consequentemente, um olhar diferente para a interpretação dos sinais digitais.

O processamento digital de sinais está presente no cotidiano da maioria das pessoas. Rádios, TVs, smartphones, telefones fixos, internet e tantos outros dispositivos que são utilizados pela grande maioria da população não seriam possíveis se não houvesse o desenvolvimento do processamento de sinais.

Como nem tudo é como se deseja, os ruídos estão presentes em praticamente todos os sinais. Mesmo sendo quase que inevitáveis, graças aos filtros analógicos e digitais é possível uma redução considerável na quantidade de ruído presente nos sinais.

A música *Ash Tray Blues*, de Papa Charlie Jackson, gravada em 1928, foi escolhida visando uma redução de ruído em sua reprodução. Sua escolha foi baseada na época de sua gravação, cujos equipamentos não tinham sofisticação suficiente para obter um som livre de ruídos.

Foi estudado o método *noise gate*, utilizado pelo software Audacity® na ferramenta *noise reduction*. Este software inspirou a criação de um algoritmo chamado *noise reduce*, de Tim Sainburg.

A música *Ash Tray Blues* foi filtrada utilizando as duas ferramentas apresentadas e após, seus resultados foram quantizados — através do SNR — e comparados, deixando claro a eficácia das ferramentas, em especial, o algoritmo de Sainburg.

Concluiu-se que é possível a redução de ruído de uma música dos anos 1920, mesmo com um curto intervalo de ruído isolado presente na música.

Todos os objetivos propostos (geral e específicos) foram alcançados, fazendo com que outros objetivos sejam abertos para novas pesquisas, como a identificação de notas musicais através da análise do espectro de frequências, o aprimoramento dos algoritmos de redução de ruídos e a criação de métricas específicas para redução de ruído.

A Algoritmo *Noise Reduce* Comentado

Pacotes utilizados

```
import matplotlib.pyplot as plt      % Gráficos
import numpy as np                   % Biblioteca numérica
import tqdm                          % Barra de progresso
import librosa                       % Análise de música e áudio
import scipy.signal                  % Processamento de sinais
```

Geração de gráficos especiais

```
# Retorna o espectrograma do sinal desejado
def plot_spectrogram(signal, title):
    fig, ax = plt.subplots(figsize=(20, 4))
    cax = ax.matshow(signal, origin="lower", aspect="auto",
                     cmap=plt.cm.hsv, vmin=-1*np.max(np.abs(signal)),
                     vmax=np.max(np.abs(signal)))
    fig.colorbar(cax)
    ax.set_title(title, fontname='Linux Libertine O', fontsize=16, y=1.06)
    plt.tight_layout()
    plt.show()
```

```
# Produz um gráfico com as estatísticas do ruído
def plot_statistics(mean_freq_noise, std_freq_noise, noise_thresh):
    plt_mean, = plt.plot(mean_freq_noise, label="Magnitude média do ruído")
    plt_std, = plt.plot(std_freq_noise, label="Desvio padrão do ruído")
    plt_std, = plt.plot(noise_thresh, label="Threshold do ruído")
    plt.title("Threshold Para a Máscara", fontname='Linux Libertine O', fontsize=16, y=1)
    plt.legend()
    plt.tight_layout()
    plt.show()
```

```
# Gráfico do filtro de suavização
def plot_filter(smoothing_filter):
    cax = plt.matshow(smoothing_filter, origin="lower")
    plt.tight_layout()
    plt.colorbar(cax)
    plt.title("Filtro de Suavização da Máscara",
              fontname='Linux Libertine O', fontsize=16, y=1.1)
    plt.show()
```

Transformadas e conversão de unidades

```
# Short-Time Fourier Transform
# Entra um sinal em função do tempo e retorna um sinal em função do tempo e da frequência
def _stft(y, n_fft, hop_length, win_length):
    return librosa.stft(y=y, n_fft=n_fft, hop_length=hop_length, win_length=win_length)
```

```
# Inverse Short-Time Fourier Transform;
# Retorna o sinal recuperado em função do tempo;
def _istft(y, hop_length, win_length):
    return librosa.istft(y, hop_length, win_length)
```

```
# Converte amplitude para decibéis, 20 * log10(S / ref);
# - S é um array com as amplitudes do sinal;
# - amin é o valor mínimo para S e ref;
# - top_db é o valor máximo para S e ref // max(20 * log10(S)) - top_db;
def _amp_to_db(x):
    return librosa.core.amplitude_to_db(x, ref=1.0, amin=1e-20, top_db=80.0)
```

```
# Converte decibéis para amplitude, 10.0**(0.5 * (S_db + log10(ref)/10));
# - ref: potência de referência;
def _db_to_amp(x,):
    return librosa.core.db_to_amplitude(x, ref=1.0)
```

Filtro de suavização e máscara

```
# Retorna os coeficientes do filtro de suavização que será aplicado à máscara;
def _smoothing_filter(n_grad_freq, n_grad_time):
    # - np.outer multiplica dois vetores a e b. Resulta em um array com len(a)
    #   linhas e len(b) colunas;
    #
    # - np.concatenate junta dois arrays em um só, colocando o segundo
    #   ao final do primeiro;
    #
    # - np.linspace retorna números espaçados uniformemente em um intervalo
    #   especificado nas duas primeiras entradas;
```

```
    #
    # - n_grad_freq é o número de canais a serem suavizados pela máscara. Aqui,
    #   ele serve para determinar quantos pontos o array vai ter. No caso,
    #   n_grad_freq +1-1, pois endpoint=False quer dizer que o último número não
    #   será incluído;
    smoothing_filter = np.outer(
        np.concatenate([
                # espaçamento de 1/(n_grad_freq+1)
                np.linspace(0, 1, n_grad_freq + 1, endpoint=False),
                # espaçamento de 1/(n_grad_freq+1)
                np.linspace(1, 0, n_grad_freq + 2),
            ] # Isso vai retornar array([0., 1/(n_grad_freq+1),
              #                           2/(n_grad_freq+1, ...,
              #                           1., ...,
              #                           2/(n_grad_freq+1, 1/(n_grad_freq+1), 0. ]);
        )[1:-1], # Descarta o primeiro e o último valor do array (ambos os 0)
        np.concatenate([
                np.linspace(0, 1, n_grad_time + 1, endpoint=False),
                np.linspace(1, 0, n_grad_time + 2),
            ])[1:-1], # Faz o mesmo para o tempo
    ) # Nesse momento, foi criada a "matriz produto"

    # Cada termo do array é dividido pela soma dos termos da "matriz produto"
    smoothing_filter = smoothing_filter / np.sum(smoothing_filter)
    return smoothing_filter
```

```
# Cria uma máscara booleana que será aplicada ao sinal
def mask_signal(sig_stft_db, sig_mask, mask_gain_dB, sig_stft):
    # Entradas;
    # - sig_stft_db, espectrograma do sinal em dB
    #     sig_stft_db = _amp_to_db(np.abs(sig_stft)),
    #   com sig_stft=_stft(audio_clip, n_fft, hop_length, win_length);
    #   onde audio_clip é o sinal de entrada, n_fft=2048, hop_length=win_length/4 e
    #   win_length=n_fft, ou seja, sig_stft_dp é um array com a potência (dB) do sinal
    #   original;

    # - sig_mask, máscara que será aplicada ao sinal;
    # - mask_gain_dB, controle de intensidade da redução de ruído da máscara;
    # - sig_stft, espectrograma do sinal;

    # Saídas:
    # - sig_stft_amp, sinal após a aplicação da máscara;
    # - sig_stft_db_masked, sinal após a aplicação da máscara em dB;

    # mask real;
    sig_stft_db_masked = (
        sig_stft_db * (1 - sig_mask)             # Dá ganho ao sinal de forma que o ruído
        + np.ones(np.shape(mask_gain_dB))        # seja atenuado;
        * mask_gain_dB * sig_mask                # Ajuste no ganho do sinal;
    )
    # mask imag;
```

```
    sig_imag_masked = np.imag(sig_stft) * (1 - sig_mask) # Parte imaginária do sinal;
    sig_stft_amp = (_db_to_amp(sig_stft_db_masked) * np.sign(sig_stft))
        + (1j * sig_imag_masked)                          # np.sign retorna -1 p/ x<0, 0 p/ x=0
                                                          # e 1 p/ x>0;
    return sig_stft_amp, sig_stft_db_masked
```

Remoção de ruído

```
# Remove ruído de um sinal através do método "noise gate";
def reduce_noise(audio_clip, noise_clip, n_grad_freq=2, n_grad_time=4, n_fft=2048,
                 win_length=2048, hop_length=512, n_std_thresh=1.5, prop_decrease=1.0,
                 verbose=False):
    # Entradas
    # - audio_clip: sinal original;
    # - noise_clip: sinal contendo apenas ruído;
    # - n_grad_freq: canais de frequência que serão suavizados na aplicação da máscara;
    # - n_grad_time: quadros de frequência que serão suavizados na aplicação da máscara;
    # - n_fft: determina quantos pontos a fft terá;
    # - win_length: tamanho da janela que será aplicada a cada quadro de tempo;
    # - hop_length: número de amostras por coluna;
    # - n_std_thresh: quão acima da média do ruído o threshold ficará (fator que
    #   multiplica o desvio padrão);
    # - prop_decrease: quanto o ruído será atenuado (1 = tudo, 0 = nada)
    # - verbose: se deseja que sejam mostrados os espectrogramas;

    # Saída
    # - Array com o sinal recuperado com redução de ruído;

    if verbose: # quantos espectrogramas serão mostrados se verbose='True';
        pbar = tqdm(total=7)
    else:
        pbar = None

    update_pbar(pbar, "STFT sobre o ruído")
    # STFT sobre o ruído;
    noise_stft = _stft(noise_clip, n_fft, hop_length, win_length)  # Espectrograma do ruído;
    noise_stft_db = _amp_to_db(np.abs(noise_stft))   # Conversão para dB;
    # Calculando as estatísticas do ruído;
    update_pbar(pbar, "STFT sobre o sinal")
    mean_freq_noise = np.mean(noise_stft_db, axis=1)  # - Média de cada linha (axis=1)
                                                      #   da matriz gerada;
    std_freq_noise = np.std(noise_stft_db, axis=1)    # - Calcula o desv. padrão em cada
                                                      #   linha da matriz;
    noise_thresh = mean_freq_noise + std_freq_noise * n_std_thresh
    # Limiar do ruído: intensidade média somada com o produto entre os desvios padrões e
    # quantas amostras com intensidade acima da média serão consideradas sinal;

    # STFT sobre o sinal;
    update_pbar(pbar, "STFT sobre o sinal")
```

```
    sig_stft = _stft(audio_clip, n_fft, hop_length, win_length)    # Espectrograma do sinal;
    sig_stft_db = _amp_to_db(np.abs(sig_stft))                     # Conversão para dB;
    update_pbar(pbar, "Geração da máscara")
    # Retorna o valor mínimo (abs) em dB do spectrograma do sinal;
    mask_gain_dB = np.min(_amp_to_db(np.abs(sig_stft)))
    # Calculando o threshold para cada quadro de tempo e bin de frequência;
    # - O limiar em dB será uma matriz que repete os valores do vetor noise_thresh
    #   (foi transformado em vetor pelo np.reshape) diretamente nas linhas abaixo;
    # np.shape(sig_stft_db)[1]xlen(mean_freq_noise). Após, a matriz é transposta;
    db_thresh = np.repeat(  # Repete valores de um array. Nesse caso(axis=0),
                            # será repetido o valor de cada linha em uma linha abaixo;
                            # np.reshape muda a ordem da matriz, que no caso é
                            # 1xlen(mean_freq_noise);
        np.reshape(noise_thresh, [1, len(mean_freq_noise)]), np.shape(sig_stft_db)[1],
                   axis=0,).T
    # Se o sinal estiver abaixo do threshold, será atenuado;
    sig_mask = sig_stft_db < db_thresh   # array lógico (True, False),
                                         # em que True vale 1 e False vale 0;
    update_pbar(pbar, "Máscara suavizada")
    # Retorna os coeficientes do filtro de suavização que será aplicado a máscara;
    smoothing_filter = _smoothing_filter(n_grad_freq, n_grad_time)
    # Executa uma convolução entre sinal e máscara;
    sig_mask = scipy.signal.fftconvolve(sig_mask, smoothing_filter, mode="same") # Convolução;
    sig_mask = sig_mask * prop_decrease # Multiplica a máscara por um valor entre 0 e 1;
    update_pbar(pbar, "Máscara aplicada")
    # Aplicando a máscara ao sinal;
    sig_stft_amp, sig_stft_db_masked = mask_signal(
        sig_stft_db, sig_mask, mask_gain_dB, sig_stft
    )
    update_pbar(pbar, "Sinal Recuperado")
    # Recuperando o sinal;
    # Recupera o sinal em função do tempo;
    recovered_signal = _istft(sig_stft_amp, hop_length, win_length)
    recovered_spec = _amp_to_db(
        np.abs(_stft(recovered_signal, n_fft, hop_length, win_length))
    ) # Espectrograma do sinal recuperado;
    if verbose:
    # Mostra os passos do algoritmo em espectrogramas;
        plot_reduction_steps(
            noise_stft_db,        # Espectrograma do ruído em decibéis;
            mean_freq_noise,      # Média de cada linha da matriz gerada;
            std_freq_noise,       # Desvio padrão de cada linha da matriz gerada;
            noise_thresh,         # Limiar do ruído;
            smoothing_filter,     # Filtro para suavizar máscara;
            sig_stft_db,          # Espectrograma do sinal;
            sig_mask,             # Espectrograma da máscara;
            sig_stft_db_masked,   # Espectrograma do sinal com a máscara aplicada;
            recovered_spec,       # Espectrograma do sinal recuperado.
        )
    return recovered_signal
```

B Algoritmo SNR

```
import numpy as np
```

```
def plot(x, y, t):
    plt.figure()
    plt.plot(x,y)
    plt.plot(x,t)
```

```
# Retorna a razão sinal-ruído em dB de um sinal;
def snr(signal, rate):
    pad = 2**(int(np.ceil(np.log2(signal.shape[0]))))
    signal_DFT = np.fft.fft(signal, n = pad)
    signal_mag2 = np.abs(signal_DFT)**2
    n = signal_mag2.shape[0]//2
    signal_mag2_nyquist = signal_mag2[0:n]
    f = np.arange(0, signal_mag2_nyquist.shape[0], 1.0)
        * (rate*1.0/signal_mag2_nyquist.shape[0])
    sig_mean = np.mean(signal_mag2_nyquist)
    sig_std = np.std(signal_mag2_nyquist)
    sig_thresh = sig_mean+sig_std
    sig_thresh *= np.ones(f.shape)
    sig_above = signal_mag2_nyquist > sig_thresh
    sig_below = signal_mag2_nyquist < sig_thresh
    sig_above = sig_above * 1.0
    sig_below = sig_below * 1.0
    sig_above *= signal_mag2_nyquist
    sig_below *= signal_mag2_nyquist
    snr = 10*np.log10(np.sum(sig_above)/np.sum(sig_below))
    graph = plot(x = f, y = signal_mag2_nyquist, t = sig_thresh)
    return snr, graph
```

Referências

ANTONIOU, Andreas. **Digital signal processing: Signals, systems, and filters**. New York: McGraw-Hill, 2006.

AUDACITY. **How Audacity Noise Reduction Works**. https://wiki.audacityteam.org/wiki/How_Audacity_Noise_Reduction_Works. 2015.

BESSADA, Dennis. A Gênese da Harmonia das Esferas no Antigo Pitagorismo. **Revista Música**, v. 14, p. 85, mai. 2014. DOI: 10.11606/rm.v14i1.115248.

BEZERRA, Juliana. **Pitágoras**. 2018. Disponível em: <https://www.todamateria.com.br/pitagoras/>.

BLACK, Harold S.; EDSON, J. O. Pulse code modulation. **Transactions of the American Institute of Electrical Engineers**, IEEE, v. 66, n. 1, p. 895–899, 1947.

BORGES, Tiago. **O que é e como usar o (Noise) Gate**. 2019. Disponível em: <https://tiagoborges.net/gate/>.

BRACEWELL, Ronald N. The Fourier transform. **Scientific American**, JSTOR, v. 260, n. 6, p. 86–95, 1989.

COOLEY, James W; TUKEY, John W. An algorithm for the machine calculation of complex Fourier series. **Mathematics of computation**, JSTOR, v. 19, n. 90, p. 297–301, 1965.

DE MARCHI, Leonardo. A Angústia do Formato: uma História dos Formatos Fonográficos. **E-Compós**, v. 2, jan. 2005. DOI: 10.30962/ec.29. Disponível em: <https://www.e-compos.org.br/e-compos/article/view/29>.

DUTTA, P.; HORN, P. M. Low-frequency fluctuations in solids: $\frac{1}{f}$ noise. **Rev. Mod. Phys.**, American Physical Society, v. 53, p. 497–516, 3 jul. 1981. DOI: 10.1103/RevModPhys.53.497. Disponível em: <https://link.aps.org/doi/10.1103/RevModPhys.53.497>.

FIGUEIREDO, Djairo Guedes de. **Análise de Fourier e equações diferenciais parciais**. Rio de Janeiro: Instituto de Matemática Pura e Aplicada, 2000.

FOURIER, Joseph. **Théorie analytique de la chaleur**. France: Chez Firmin Didot, père et fils, 1822.

GOMES, Rodrigo M. Do fonógrafo ao MP3: algumas reflexões sobre música e tecnologia. **Revista Brasileira de Estudos da Canção–ISSN**, v. 2238, p. 1198, 2014.

GOUVEIA, Rosimar. **Desvio Padrão**. 2019. Disponível em: <https://www.todamateria.com.br/desvio-padrao/>.

HALLEY, John M. Ecology, evolution and 1f-noise. **Trends in ecology & evolution**, Elsevier, v. 11, n. 1, p. 33–37, 1996.

HUNTER, J. D. Matplotlib: A 2D graphics environment. **Computing in Science & Engineering**, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

JONES, Eric; OLIPHANT, Travis; PETERSON, Pearu et al. **SciPy**: **Open source scientific tools for Python**. [S.l.: s.n.], 2001. Disponível em: <http://www.scipy.org/>.

KLUYVER, Thomas et al. Jupyter Notebooks-a publishing format for reproducible computational workflows. In: ELPUB. [S.l.: s.n.], 2016. p. 87–90.

LYONS, Richard G. **Understanding Digital Signal Processing**. 3. ed. India: Pearson Education India, 2011.

MCFEE, Brian et al. librosa: Audio and music signal analysis in python. In: PROCEEDINGS of the 14th Python in Science Conference. [S.l.: s.n.], 2015. v. 8.

O'HAVER, Tom. A pragmatic introduction to signal processing. **University of Maryland at College Park**, 1997.

PETZOLD, Charles. **Code: The hidden language of computer hardware and software**. New York: Microsoft Press, 2000.

SAINBURG, Tim. **Noise reduction using spectral gating in Python**. 2018. Disponível em: <https://timsainburg.com/noise-reduction-python.html>. Acesso em: 7 mai. 2019.

SANTOS, Fabiano J. **Introdução às séries de Fourier**. Departamento de Matemática e Estatística, PUC Minas, 2004. Disponível em: <http://www.matematica.pucminas.br/profs/web_fabiano/calculo4/sf.pdf>.

SMITH, Julius O. **Spectral Audio Signal Processing**. [S.l.]: http://ccrma.stanford.edu/~jos/sasp/, 2011. Livro online, edição 2011.

VAN DER WALT, Stefan; COLBERT, S Chris; VAROQUAUX, Gael. The NumPy array: a structure for efficient numerical computation. **Computing in Science & Engineering**, IEEE Computer Society, v. 13, n. 2, p. 22, 2011.

VAN ROSSUM, Guido; DRAKE JR, Fred L. **Python tutorial**. [S.l.]: Centrum voor Wiskunde en Informatica Amsterdam, Netherlands, 1995.

VETTERLI, Martin; PRANDONI, Paolo. **Signal processing for communications**. Switzerland: EPFL Press, 2008.

ZANATO, FERNANDO DA SILVA. **Matemática e Música: Relações entre as Séries e Transformadas de Fourier e a Teoria Musical**. 2017. Diss. (Mestrado) – UNIVERSIDADE DO ESTADO DE MATO GROSSO.