Carlos Eduardo Leal de Castro

Análise dos métodos de reconstrução de imagens aplicados a sinais de áudio

Florianópolis

2017

Carlos Eduardo Leal de Castro

Análise dos métodos de reconstrução de imagens aplicados a sinais de áudio

Trabalho de Conclusão de Curso apresentado ao Curso de Matemática, do Departamento de Matemática - Centro de Ciências Físicas e Matemáticas da Universidade Federal de Santa Catarina, para obtenção de grau de Licenciado em Matemática.

Universidade Federal de Santa Catarina

Centro de Ciências Físicas e Matemática

Departamento de Matemática

Licenciatura em Matemática

Orientador: Dr. Leonardo Koller Sacht

Florianópolis

2017

Carlos Eduardo Leal de Castro

Análise dos métodos de reconstrução de imagens aplicados a sinais de áudio

Trabalho de Conclusão de Curso apresentado ao Curso de Matemática, do Departamento de Matemática - Centro de Ciências Físicas e Matemáticas da Universidade Federal de Santa Catarina, para obtenção de grau de Licenciado em Matemática.

Trabalho aprovado. Florianópolis, 2017.

Profa. Dra. Sonia Palomino Castro (UFSC)
Coordenadora do Curso

Banca Examinadora:

Dr. Leonardo Koller Sacht(Orientador)
Universidade Federal de Santa Catarina

Prof. Dr. Jáuber Cavalcante de Oliveira
Universidade Federal de Santa Catarina

Prof. Dr. Licio Hernanes Bezerra
Universidade Federal de Santa Catarina

Florianópolis
2017

Ao meu pai.

Agradecimentos

Gostaria de agradecer aos meus pais José Carlos e Tânia Leal pelo apoio, força, luz e paciência, à minha esposa e amiga Maíra Sevegnani pelo companheirismo, amor, carinho, colo e paciência pelas noites em claro e aos meus irmãos Marco Antônio, Ana Cristina (Tina) e João Vitor por manter a rédea da irmandade sempre firme.

Agradeço ao meu orientador, Leonardo Koller Sacht, por me abrir os caminhos, guiar os olhares e os pensamentos nesta pesquisa. Agradeço imensamente aos meus colegas que, junto comigo desde 2014, ficaram firmes e fortes nesta batalha matemática, Sabrina de Moraes Vigano e Ben-Hur Eidt, além dos colegas que mudaram o rumo ou chegaram por outros caminhos: Gabriel Schafaschek, Mateus Oliveira, Jean Gengnagel, Carlos Eduardo Caldeira, Helena Carolina Koch, Bruna da Silva Donadel, André Borges, Letícia Carvalho, Ana Altomani, Matheus Pimenta, Yuri Farias, Beatriz Khey, João Ruiz, entre tantos outros.

Agradeço aos meus queridos professores pelo apoio, dedicação e paciência, José Luiz Rosas Pinho, Carmem Suzane Comitre Gimenez, Silvia Martini de Holanda, Danilo Royer, David Antônio da Costa, Márcia Maria Bernal, Rafael Falcão, Melissa Weber Mendonça, Eliezer Batista, Celso Melchiades Doria, Maicon Marques Alves, Licio Hernanes Bezerra, Leonardo Silveira Borges, Alda Dayana Mattos Mortari, Fernando de Lacerda Mortari, Jáuber Cavalcante de Oliveira, João Artur de Souza e Rodrigo Garcez da Silva, que me mantem inserido e valorizado no mundo da música, além dos funcionários e servidores da UFSC, que nos acompanham em nossa trajetória.

"Eu passei muito tempo
Aprendendo a beijar outros homens
Como beijo o meu pai
Eu passei muito tempo
Pra saber que a mulher que eu amei
Que amo, que amarei
Será sempre a mulher
Como é minha mãe."
(Gilberto Gil)

Resumo

Dentro da área de recuperação de informações musicais, este trabalho irá investigar os resultados dos principais métodos de reconstrução de imagens em sua aplicação a sinais de áudio, além de buscar compreender os principais resultados por trás destes mecanismos. Desse modo, busca-se construir uma base teórica, com objetivo de estruturar os conceitos das transformadas de Fourier e convolução para o tratamento destes métodos, os quais se inserem nos métodos de amostragem e reconstrução de imagens. Além disso, serão feitas análises dos métodos utilizados para avaliar quais deles atuam de maneira mais eficiente no tratamento de um som. Dentre os resultados obtidos, percebeu-se que os melhores métodos de reconstrução de imagens aplicados a sons variam dependendo do tipo do som que se quer reconstruir.

Palavras-chave: Reconstrução de imagens; Sinais de áudio; Recuperação de informações musicais; Transformada de Fourier; Convolução.

Abstract

Within the area of musical information retrieval, this paper aims to investigate the results of the main image reconstruction methods in their applications to audio signals, in addition to seeking to understand the main results behind these mecanisms. Thus, we seek to build a theorical basis with the objective of structuring the concepts of Fourier transform and convolution for the treatment of these methods, which are inserted in the pipeline for images sampling and reconstruction. Besides that, analyses will be held on the methods used in order to evaluate which one acts more efficiently on sound treatment. Among the results obtained, it was noticed that the best image reconstruction methods applied to sound vary depending on the sound type we want to reconstruct.

Keywords: Image reconstruction; Audio Signals; Music Information Retrieval; Fourier Transform; Convolution.

Sumário

Introdução

Os sons estão presentes na vida de milhões de pessoas mundo afora. Seus conceitos e formas constituem um conjunto infinito de possibilidades de análises complexas, seja na forma de uma canção orquestrada, simples vibrações de cordas em um violão ou até análise de um discurso proferido por um orador.

A indústria da música tem se regulado, cada vez mais, nos avanços tecnológicos na área digital e da engenharia do som. A evolução de programas digitais, assim como evolução nos estudos da música teórica e as ramificações de suas práticas, proporcionam as possibilidades de manipulação que, antes, eram muitas vezes impossíveis de serem executadas.

De acordo com (GIANNAKOPOULOS; PIKRAKIS, 2014), a manipulação de Áudio digital é uma área que vem crescendo muito ao longo dos anos. Isso se dá por conta do desenvolvimento tecnológico na área e, principalmente, pelo crescimento no mercado digital e das pesquisas em torno desse assunto. Ainda segundo (GIANNAKOPOULOS; PIKRAKIS, 2014), no mundo estão trabalhando, principalmente, as seguintes áreas de pesquisa relacionadas a Áudio:

- **Reconhecimento de discurso**: O principal fundamento deste tópico é a tradução do sinal do discurso através de texto usando ferramentas computacionais. Este domínio é considerado a área de pesquisa mais antiga em processamento de sinais.
- **Identificação e visualização do orador**: Este tópico foca em métodos de diferenciação entre diferentes oradores. Está área de pesquisa é útil em desenvolvimento das áreas de sistema de segurança.
- **Recuperação de informações musicais**: Neste tópico, a pesquisa se baseia em extrair e automatizar informações do sinal de áudio, indexação inteligente, recuperação de sinais perdidos ou danificados, navegação de faixas de música e recomendações de novas músicas a partir de escolhas passadas, entre outros.
- **Detecção de eventos em áudio**: Neste tópico estuda-se detecção de acontecimentos a partir do sinal de áudio, por exemplo, detecção de violência e detecção de intrusos em ambientes.
- **Reconhecimento de emoção do discurso**: Este tópico analisa os sinais de áudio para detectar as emoções do orador, como raiva, alegria, depressão, etc. Essa linha de pesquisa, que ganhou força nos últimos anos, pode ter colaboração de estudos

de reconhecimento facial. Esse ramo de pesquisa poderá contribuir no futuro da inteligência artificial e estudos de interação humano-máquina.

- **Diversos métodos de análise de conteúdo em filmes**: Este tópico analisa e reconhece ações em filmes a partir das informações de áudio, vídeo e texto.

Dentro da área de recuperação e reconstrução de informações musicais, iremos derivar os caminhos que de fato tiveram resultados significativos na área de processamento e reconstrução de imagens, área essa que mantém relações próximas com o processamento de áudio, pela possibilidade de representarmos sons e imagens como funções.

Em sua Tese de Doutorado, (SACHT, 2014) desenvolveu um método de Quasi-Interpoladores Otimizados para reconstrução de imagens, com resultado superior aos apresentados anteriormente nesta área de pesquisa, apresentando critérios para se obter filtros digitais e funções geradoras para reconstrução. Além disso, trouxe um apanhado histórico dos principais métodos de reconstrução de imagens e suas principais características.

Visto isso e devido à proximidade entre as teorias bases de imagens e áudio, acreditamos ser válido o seguinte questionamento: O Método de Quasi-Interpoladores Otimizados para reconstrução de imagens é aplicável a sinais de áudio? Quais métodos de reconstrução de imagens dá o melhor resultado ao aplicarmos em sinais de áudio? Como podemos ajustá-los para reconstruir de maneira eficiente um sinal de áudio digital?

Como as aplicações neste trabalho são efetuadas no domínio do tempo, tal proximidade se dá na representação das imagens e sons discretizados, como matrizes (no caso das imagens), matrizes com duas colunas (no caso dos sons estéreo) e vetores (no caso dos sons mono), de modo que as aplicações das convoluções contínuas, discretas e mistas, operações base dos métodos estudados aqui, se apliquem em todos os casos. Além disso, o desenvolvimento da teoria aqui trabalhada se aplica a sinais quaisquer, desde que assumam as condições estabelecidas.

Neste presente trabalho, iremos construir uma base teórica para analisar a funcionalidade dos métodos de reconstrução de imagens para aplicar a sinais de áudio, construir uma base em manipulação de áudio no MATLAB e comparar os principais métodos de reconstrução de imagem em sua aplicabilidade a sons.

Este trabalho está organizado em 3 capítulos. O primeiro trará resultados preliminares para construção da teoria dos métodos de reconstrução de imagens, trabalhando as teorias introdutórias das séries de Fourier. O segundo capitulo mostra os principais conceitos e os teoremas fundamentais, para comprovar a admissibilidade das aplicações. O terceiro capítulo traz os resultados dos testes, obtidos através das aplicações dos principais métodos de reconstrução de imagens aplicados a áudio, conclusões e trabalhos futuros.

Além disso, este trabalho traz um apêndice com os scripts desenvolvidos para

aplicar os métodos de reconstrução de imagens em sinais de áudio, bem como o script do método de comparação entre cada processo de reconstrução e um script desenvolvido para imprimir os spectrogramas dos sons.

Pretende-se com este trabalho apresentá-lo como Trabalho de Conclusão de Curso do acadêmico Carlos Eduardo Leal de Castro, do curso de Licenciatura em Matemática da Universidade Federal de Santa Catarina, sob orientação do Professor Doutor Leonardo Koller Sacht, do Departamento de Matemática dessa mesma universidade.

1 Introdução às Séries de Fourier

Traremos aqui as noções básicas das Séries de Fourier, para auxiliar no desenvolvimento teórico deste trabalho. Aqui, falaremos de Funções Periódicas, convergência uniforme, diferenciação e integração de Séries de Funções, coeficientes das Séries de Fourier e sua forma complexa, além dos principais resultados de convergência da Série de Fourier.

Para um melhor entendimento deste capítulo, sugere-se ter uma boa base em Álgebra Linear (LIMA, 2000), Cálculo Diferencial e Integral (GUIDORIZZI, 2000) e Análise (LIMA, 2000), (LIMA, 2004), em especifico, conteúdos que abrangem sequências numéricas, convergência de séries, continuidade, derivação e integração.

1.1 Sequências e Séries de Funções

Para falarmos das séries de Fourier, precisamos falar de convergência uniforme de sequências e séries de funções. Não focaremos aqui nos critérios de convergência e em como verificar se uma sequência ou série converge ou não. Precisaremos aqui dos conceitos de convergência uniforme para podermos garantir a convergência das séries de Fourier. Ao utilizarmos a notação $(f_n : n \in \mathbb{N})$, estaremos tratando de sequências de funções $f_n : A \longrightarrow \mathbb{C}$, com $n \in \mathbb{N}$ e $A \subseteq \mathbb{R}$.

1.1.1 Sequências de Funções

Definição 1.1.1. *Seja $f_n : A \longrightarrow \mathbb{C}$, com $n \in \mathbb{N}$ e $A \subseteq \mathbb{R}$ uma sequência de funções. Se existir o limite $\lim_{n\to\infty} f_n(x)$, para cada $x \in A$, dizemos que a sequência $(f_n : n \in \mathbb{N})$ converge pontualmente (ou simplesmente) em A e podemos definir*

$$f(x) = \lim_{n\to\infty} f_n(x)$$

Definição 1.1.2. *Seja $(f_n : n \in \mathbb{N})$ uma sequência de funções definidas sobre um domínio $A \subseteq \mathbb{R}$. Dizemos que $(f_n : n \in \mathbb{N})$ converge uniformemente sobre A se, para todo $\varepsilon \in \mathbb{R}, \varepsilon > 0$, existir um $n_0 \in \mathbb{N}$ tal que, para todo $n \geq n_0$ $|f_n(x) - f(x)| < \varepsilon, \forall x \in A$.*

Proposição 1.1.1. *Seja $(f_n : n \in \mathbb{N})$ uma sequência de funções definidas sobre um domínio $A \subseteq \mathbb{R}$. $(f_n : n \in \mathbb{N})$ converge uniformemente para $f(x), \forall x \in A$ se, e somente se, $\lim_{n\to\infty} \sup_{x\in A} \mid f_n(x) - f(x) \mid = 0$*

Demonstração. ($\Rightarrow$) Suponhamos que f_n converge uniformemente para $f(x), \forall x \in A$. Então, $\forall \varepsilon \in \mathbb{R}, \varepsilon > 0, \exists n_0 \in \mathbb{N}$ tal que $\mid f_n(x) - f(x) \mid \leq \varepsilon, \forall n \leq n_0$ *e* $\forall x \in A$.

Dado $n \in \mathbb{N}$ quaisquer tal que $n \leq n_0$ e fixe $x \in A$ qualquer. Considere o conjunto $\rho(f_n, f) := \{\mid f_n(x) - f(x) \mid; n \in \mathbb{N}, n \leq n_0 \ e \ x \in A\}$. Suponha que $\lim_{n\to\infty} \rho(f_n, f) = \varepsilon_0$ com $\varepsilon_0 > 0$ qualquer.

Mas, para todo $n \leq n_0$, temos que $\mid f_n(x) - f(x) \mid \leq \frac{\varepsilon_0}{2} < \varepsilon_0$.

Segue, portanto que $\mid f_n(x) - f(x) \mid \nrightarrow \varepsilon_0$ e então

$$\lim_{n\to\infty} \sup_{x\in A} \mid f_n(x) - f(x) \mid = 0.$$

($\Leftarrow$) Suponha que $\lim_{n\to\infty} \sup_{x\in A} \mid f_n(x) - f(x) \mid = 0$. Então, para qualquer $\varepsilon > 0, \exists n_0 \in \mathbb{N}$ tal que $\mid f_n(x) - f(x) \mid \leq \varepsilon, \forall n \leq n_0 \ e \ \forall x \in A$. Portanto, temos que f_n converge uniformemente para $f(x), \forall x \in A$. ■

Exemplo 1.1.1. *1. A sequência de funções dadas por $f_n(x) = x^n$, para $0 \leq x \leq 1$, converge simplesmente mas não uniformemente para a função*

$$f(x) = \begin{cases} 0 & se \quad 0 \leq x < 1 \\ 1 & se \quad x = 1 \end{cases}$$

Para mostrarmos a convergência pontual da sequência de funções $f_n(x) = x^n$ para $0 \leq x \leq 1$, basta avaliarmos o $\lim_{n\to\infty} f_n(x)$, para cada $x \in [0, 1]$. Vamos observar os casos:

- *$x = 0$: $\lim_{n\to\infty} f_n(x) = \lim_{n\to\infty} x^n = \lim_{n\to\infty} 0^n = 0$.*
- *$x = 1$: $\lim_{n\to\infty} f_n(x) = \lim_{n\to\infty} x^n = \lim_{n\to\infty} 1^n = 1$.*
- *$x_0 \in (0, 1)$: $\lim_{n\to\infty} f_n(x) = \lim_{n\to\infty} x^n = \lim_{n\to\infty} x_0^n = 0$, pois $0 < x_0 < 1$.*

Segue, portanto, que $f_n(x) \longrightarrow f(x)$ pontualmente, para cada $x \in [0, 1]$.

Para mostrarmos que $f_n(x)$ não converge uniformemente para $f(x)$, usaremos a proposição 1.1.1. Observemos que

$$\lim_{n\to\infty} \sup_{x\in[0,1]} \mid f_n(x) - f(x) \mid = \lim_{n\to\infty} \sup_{x\in[0,1]} \mid x^n - f(x) \mid .$$

Se $x = 1$, vimos que $f_n(1) = 1, \forall n \in \mathbb{N}$. Logo $\mid x^n - f(x) \mid = \mid 1 - 1 \mid = 0$.

Se $x = 0$, vimos que $f_n(1) = 0, \forall n \in \mathbb{N}$. Logo $\mid x^n - f(x) \mid = \mid 0 - 0 \mid = 0$.

Considerando $x_0 \in (0, 1)$ qualquer, temos que $f(x_0) = 0$ e então $\mid x^n - f(x) \mid = \mid x_0^n - 0 \mid = x_0^n$ e

$$\lim_{n\to\infty} \sup_{x_0\in(0,1)} \mid x^n - f(x) \mid = \lim_{n\to\infty} \sup_{x_0\in(0,1)} x_0^n = 1.$$

Segue, portanto, que f_n não converge uniformemente para $f(x)$.

2. *A sequência de funções* $f_n(x) = \frac{1}{n}x$, *definidas para* $x \in [0,1]$, *converge uniformemente para a função* $f = 0$.

   Seja $\varepsilon \in \mathbb{R}$, *com* $\varepsilon > 0$ *e tome* $n_0 = \frac{1}{\varepsilon}$. *Então, para todo* $n \in \mathbb{N}, n \geq n_0$

   $$|f_n - f(x)| = \left|\frac{x}{n} - 0\right| = \left|\frac{x}{n}\right| = \frac{|\ x\ |}{n} \leq \frac{1}{n} \leq \frac{1}{\frac{1}{\varepsilon}} = \varepsilon,$$

   para todo $x \in [0,1]$.

   Segue, portanto, que $f_n(x) \longrightarrow f(x)$ *uniformemente,* $\forall x \in [0,1]$.

Definição 1.1.3 (Sequência de Cauchy)**.** *Seja* $(f_n : n \in \mathbb{N})$ *uma sequência de funções definidas sobre um domínio* $A \subseteq \mathbb{R}$. *Dizemos que* $(f_n : n \in \mathbb{N})$ *é de Cauchy se, para todo* $\varepsilon \in \mathbb{R}, \varepsilon > 0$, *existir um* $n_0 \in \mathbb{N}$ *tal que, para todos* $n, m \geq n_0$ $|f_m(x) - f_n(x)| < \varepsilon, \forall x \in A$.

Teorema 1.1.1 (Critério de Cauchy)**.** *Seja* $(f_n : n \in \mathbb{N})$ *uma sequência de funções definidas sobre um domínio* $A \subseteq \mathbb{R}$. *Então existe uma função* f *definida em* $A \subseteq \mathbb{R}$ *tal que* $f_n \to f$ *uniformemente em* A *se e somente se* $(f_n : n \in \mathbb{N})$ *for de Cauchy.*

Demonstração. ($\Rightarrow$) Seja $(f_n : n \in \mathbb{N})$ uma sequência de funções definidas em $A \subseteq \mathbb{R}$ tal que, por hipótese, converge uniformemente para uma f em A.

Pela definição de convergência uniforme, $\forall \varepsilon \in \mathbb{R}, \varepsilon > 0$, existe um $n_0 \in \mathbb{N}$ tal que, para todos $n, m \geq n_0, |f_n(x) - f(x)| < \frac{\varepsilon}{2}$ e $|f_m(x) - f(x)| < \frac{\varepsilon}{2}, \forall x \in A$.

Note que,

$$\begin{aligned} |f_m(x) - f_n(x)| &= |f_m(x) - f(x) + f(x) - f_n(x)| \\ &\leq |f_m(x) - f(x)| + |f(x) - f_n(x)|\ , \forall x \in A \text{ e } \forall n, m \geq n_0 \\ &\leq \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon \end{aligned}$$

($\Leftarrow$) Seja $(f_n : n \in \mathbb{N})$ uma sequência de Cauchy. Então, para cada $x_0 \in \mathbb{A}$, a sequência numérica $(f_n(x_0))_{n \in \mathbb{N}}$ será uma sequência numérica de Cauchy e, portanto, convergente.

Seja $f : A \to \mathbb{R}$ definida por

$$f(x) = \lim_{n \to \infty} f_n(x).$$

Mostraremos que $f_n \to f$ uniformemente.

Por hipótese, temos que $(f_n : n \in \mathbb{N})$ é de Cauchy. Então, para $\varepsilon > 0$ qualquer, $\exists n_0 \in \mathbb{N}$ tal que

$$|f_n(x) - f_m(x)| \leq \varepsilon, \forall x \in A \text{ e } \forall n, m \geq n_0.$$

Fixando $x \in A$ qualquer e $n \geq n_0$, como $f(x) = \lim_{n \to \infty} f_n(x)$, fazendo $\lim_{m \to \infty} |f_n(x) - f_m(x)| \leq \lim_{m \to \infty} \varepsilon$, obtemos

$$|f_n(x) - f(x)| \leq \varepsilon,\ \textit{com } n \geq n_0 \textit{ e } x \in A.$$

Como x foi fixado arbitrariamente, segue portanto que $f_n \to f$ uniformemente. ■

As sequências mais importantes que trabalharemos aqui são as sequências de funções contínuas. Assim, o próximo teorema é um resultado importante, que irá nos auxiliar na relação entre sequências de funções e a continuidade.

Teorema 1.1.2. *Seja $(f_n)_{n\in\mathbb{N}}$ uma sequência de funções contínuas que converge uniformemente, e seja f seu limite. Então f é uma função contínua.*

Demonstração. Vamos mostrar a continuidade de f. Como f_n converge uniformemente para f, $\forall \varepsilon \in \mathbb{R}, \varepsilon > 0$, $\exists n_0 \in \mathbb{N}$ tal que se $n \geq n_0$,

$$| f_n(x) - f(x) | < \frac{\varepsilon}{3} \tag{1.1.1}$$

Da continuidade de $(f_n)_{n\in\mathbb{N}}$, temos que, para x_0 no domínio de f_n, existe $\delta \in \mathbb{R}, \delta > 0$ tal que, para $n \geq n_0$, $| x - x_0 | < \delta$ e

$$| f_n(x) - f_n(x_0) | < \frac{\varepsilon}{3} \tag{1.1.2}$$

Então:

$$\begin{aligned} | f(x) - f(x_0) | &= | f(x) - f_n(x) + f_n(x) - f_n(x_0) + f_n(x_0) - f(x_0) | \\ &\leq | f(x) - f_n(x) | + | f_n(x) - f_n(x_0) | + | f_n(x_0) - f(x_0) | \qquad (1.1.3) \\ &< \frac{\varepsilon}{3} + \frac{\varepsilon}{3} + \frac{\varepsilon}{3} = \varepsilon \end{aligned}$$

Segue portanto que f é contínua em todo seu domínio. ■

O Teorema 1.1.2 nos dá uma forma de analisar a convergência de uma sequência de funções. Se, por exemplo, uma sequência qualquer tiver convergência pontual e conhecido seu limite, basta observarmos se o limite de convergência pontual é contínuo. Caso não seja, podemos afirmar, pelo teorema, que a função não converge uniformemente.

Definição 1.1.4. *Uma sequência de funções $(f_n : n \in \mathbb{N})$ é dita monótona se ela for crescente, decrescente, não crescente ou não decrescente.*

O próximo teorema é um resultado para determinarmos a convergência uniforme de uma sequência de funções. O teorema a seguir é conhecido como "Teste de Dini" e nos dá condições de afirmar a convergência uniforme de uma sequência de funções.

Teorema 1.1.3 (Teste de Dini)**.** *Seja $(f_n)_{n\in\mathbb{N}}$ uma sequência de funções contínuas definidas em um compacto $I \subset \mathbb{R}$. Se $(f_n)_{n\in\mathbb{N}}$ for monótona e convergir pontualmente para uma função contínua f, então a convergência é uniforme.*

Demonstração. Mostraremos que $(f_n)_{n\in\mathbb{N}}$ converge uniformemente num compacto $I \subset \mathbb{R}$. Sem perda de generalidade, vamos considerar que (f_n) é não-decrescente, isto é, $f_n \leq f_{n+1}$, pela hipótese de monotonicidade de f_n.

Fixe $\varepsilon > 0$ e $x_0 \in I$ de modo que, para $n_{x_0} \in \mathbb{N}$

$$| f(x_0) - f_{n_{x_0}}(x_0) | \leq \frac{\varepsilon}{3}.$$

Pela continuidade de f e f_n segue que, fixado $\delta_0 > 0$, temos que se $| x - x_0 | < \delta_0$, então

$$\begin{aligned} | f(x) - f_{n_{x_0}}(x_0) | &= | f(x) - f(x_0) + f(x_0) - f_{n_{x_0}}(x) + f_{n_{x_0}}(x) - f_{n_{x_0}}(x_0) | \\ &\leq | f(x) - f(x_0) | + | f(x_0) - f_{n_{x_0}}(x) | + | f_{n_{x_0}}(x) - f_{n_{x_0}}(x_0) | \\ &\leq \frac{\varepsilon}{3} + \frac{\varepsilon}{3} + \frac{\varepsilon}{3} = \varepsilon \end{aligned} \tag{1.1.4}$$

Considere uma cobertura para o compacto I pelos intervalos $I_x = \{x \in I; | x - x_0 | < \delta_0\}$. Como I é compacto, temos que existe uma cobertura finita de abertos, digamos $I_{x_1}, I_{x_2}, \ldots, I_{x_m}$, para algum $m \in \mathbb{N}$. Seja $n_0 = \max\{n(x_1), n(x_2), \ldots, n(x_m)\}$. Dado $y \in I$ qualquer, temos que $y \in I_{x_k}$, para algum $x_k, k = 1, 2, \ldots, m$. Pela monotonicidade de (f_n), temos que

$$| f(y) - f_{n_0}(y) | \leq | f(y) - f_{n_k}(y) | \leq \varepsilon$$

e, portanto, para todo $n \leq n_0$ e pela monotonicidade de (f_n), temos que

$$| f_n(x) - f(x) | \leq \varepsilon$$

para todo $x \in I$. ■

Para falarmos de limite, integração e derivação de sequências de funções, precisaremos de três teoremas que nos darão base para futuras definições.

Teorema 1.1.4. *Seja $(f_n : n \in \mathbb{N})$ uma sequência de funções Riemann integráveis sobre $I = [a, b] \subset \mathbb{R}$ com $a < b$. Vamos supor que $f_n \longrightarrow f$ uniformemente quando $n \longrightarrow \infty$ sobre I. Então, f é Riemann integrável e*

$$\lim_{n\to\infty} \int_I f_n(x)dx = \int_I \lim_{n\to\infty} f_n(x) = \int_I f(x)dx \tag{1.1.5}$$

Demonstração. Temos por hipótese que (f_n) converge uniformemente para f. Assim, podemos tomar $\varepsilon > 0$ e escolher $n \in \mathbb{N}$ de modo que

$$| f_n(x) - f(x) | < \frac{\varepsilon}{3(b-a)}, \forall x \in I. \tag{1.1.6}$$

Então,

$$-\frac{\varepsilon}{3(b-a)} + f(x) < f_n(x) < f(x) + \frac{\varepsilon}{3(b-a)}, \forall x \in I \tag{1.1.7}$$

Como (f_n) é Riemann Integrável, temos que existe uma partição p do intervalo I de modo que

$$S(f_n, p) - s(f_n, p) < \frac{\varepsilon}{3},$$

em que $S(f_n, p)$ e $s(f_n, p)$ são, respectivamente, as Somas de Riemann superior e inferior de f_n, na partição p.

Sejam $M_j := \sup\{f(x); x \in [x_{j-1}, x_j]\}$ e $m_j := \inf\{f(x); x \in [x_{j-1}, x_j]\}$. Da equação 1.1.7, temos que

$$-\frac{\varepsilon}{3(b-a)} + M_j(f) \leq M_j(f_n) \leq M_j(f) + \frac{\varepsilon}{3(b-a)}, \forall x \in I \tag{1.1.8}$$

$$-\frac{\varepsilon}{3(b-a)} + m_j(f) \leq m_j(f_n) \leq m_j(f) + \frac{\varepsilon}{3(b-a)}, \forall x \in I \tag{1.1.9}$$

Então, de (1.1.7), (1.1.8), (1.1.9)

$$\begin{aligned} S(f,p) - s(f,p) &= |S(f,p) - s(f,p)| \\ &\leq | S(f,p) - S(f_n,p) | + \\ &\quad + | S(f_n,p) - S(f_n,p) | + \\ &\quad + | s(f_n,p) - s(f_n,p) | \\ &< \frac{\varepsilon(b-a)}{3(b-a)} + \frac{\varepsilon}{3} + \frac{\varepsilon(b-a)}{3(b-a)} = \varepsilon \end{aligned} \tag{1.1.10}$$

Para provarmos 1.1.5, temos que

$$\begin{aligned} \left| \int_I f_n(x)dx - \int_I f(x)dx \right| &= \left| \int_I (f_n(x) - f(x)) \right| dx \\ &\leq \sup_{x \in I} | f_n(x) - f(x) | \cdot (b-a) \end{aligned} \tag{1.1.11}$$

Pela proposição 1.1.1, $\left| \int_I f_n(x)dx - \int_I f(x)dx \right| = 0$ e, portanto,

$$\lim_{n\to\infty} \int_I f_n(x)dx = \int_I \lim_{n\to\infty} f_n(x) = \int_I f(x)dx$$

■

Finalmente, para trabalharmos com as séries de Fourier a seguir, precisaremos de um importante resultado que relaciona sequências de funções e derivação. Existem duas versões deste teorema. Uma delas, que é considerada "forma fraca" do teorema, considera a hipótese de que, dada a sequência de funções $(f_n : n \in \mathbb{N})$, deriváveis em um compacto qualquer, as derivadas f_n', para cada n, são contínuas. Enunciaremos aqui a "forma forte" do teorema, cuja única diferença entre elas é a ausência da hipótese de continuidade das f_n'.

Teorema 1.1.5. *(Forma forte) Seja $(f_n : n \in \mathbb{N})$ uma sequência de funções deriváveis em um intervalo compacto qualquer $[a, b] = I \subset \mathbb{R}$. Suponha que a sequência $(f_n' : n \in \mathbb{N})$ convirja uniformemente em I e que exista $x_0 \in I$ de modo que a sequência $(f_n(x_0))_{n\in\mathbb{N}}$ convirja. Então $(f_n : n \in \mathbb{N})$ converge uniformemente e $\dfrac{d}{dx}\lim\limits_{n\to\infty} f_n(x) = \lim\limits_{n\to\infty}\dfrac{d}{dx} f_n(x)$.*

Demonstração. Primeiramente, demonstraremos que a sequência de funções $(f_n : n \in \mathbb{N})$ converge uniformemente. Em seguida, vamos mostrar que $f'(x) = \lim\limits_{n\to\infty} f_n'(x)$, para todo $x \in I$.

Dado $\varepsilon > 0$ qualquer, temos por hipótese que a sequência de funções $(f_n' : n \in \mathbb{N})$. Então, existe $n_1 \in \mathbb{N}$ de modo que, para todos $n, m \geq n_1$,

$$| f_n'(x) - f_m'(x) | < \frac{\varepsilon}{2 \cdot (b - a)} \tag{1.1.12}$$

para todo $x \in I$.

Temos também por hipótese que $\exists x_0 \in I$ tal que a sequência $(f_n(x_0))_{n\in\mathbb{N}}$ convirja. Logo, existe $n_2 \in \mathbb{N}$ tal que para todos $n, m \geq n_2$, $n, m \geq n_1$,

$$| f_n(x_0) - f_m(x_0) | < \frac{\varepsilon}{2}. \tag{1.1.13}$$

Sem perda de generalidade, consideremos um $x \in I$ qualquer de modo que $x < x_0$. Aplicando o Teorema do Ponto Médio na função $f_n(x) - f_m(x)$ no intervalo $[x, x_0]$, temos que existe $\tau \in (x, x_0)$ tal que

$$[f_n(x) - f_m(x)] - [f_n(x_0) - f_m(x_0)] \;=\; (x - x_0)[f_n'(\tau) - f_m'(\tau)] \tag{1.1.14}$$

Das equações 1.1.12, 1.1.13 e 1.1.14 temos:

$$
\begin{aligned}
| f_n(x) - f_m(x) | &= | [f_n(x_0) - f_m(x_0)] + (x - x_0)[f_n'(\tau) - f_m'(\tau)] | \\
| f_n(x) - f_m(x) | &\leq | f_n(x_0) - f_m(x_0) | + | (x - x_0)[f_n'(\tau) - f_m'(\tau)] | \\
| f_n(x) - f_m(x) | &\leq | f_n(x_0) - f_m(x_0) | + (b-a) \cdot | f_n'(\tau) - f_m'(\tau) | \\
| f_n(x) - f_m(x) | &\leq \frac{\varepsilon}{2} + (b-a) \cdot \frac{\varepsilon}{2 \cdot (b-a)} \\
| f_n(x) - f_m(x) | &\leq \varepsilon
\end{aligned}
$$

para todos $n, m \geq n_0 = \max\{n_1, n_2\}$ e para todo $x \in I$.

Pelo critério de Cauchy, e dado que foi tomado $x \in I$ qualquer, temos que a sequência $(f_n : n \in \mathbb{N})$ converge uniformemente.

Agora, mostraremos que $f'(x) = \lim_{n \to \infty} f_n'(x)$, para todo $x \in I$. Dado $\varepsilon > 0$, tome $n_0 \in \mathbb{N}$ tal que $\forall n, m \geq n_0$ e, usando as equações 1.1.12 e 1.1.14, temos que

$$
| [f_n(x) - f_m(x)] - [f_n(x_0) - f_m(x_0)] | \leq \varepsilon \, | \, x - x_0 \, | \tag{1.1.15}
$$

e fazendo $m \longrightarrow \infty$, temos

$$
\left| \frac{f_n(x) - f_n(x_0)}{x - x_0} - \frac{f(x) - f(x_0)}{x - x_0} \right| \leq \frac{\varepsilon}{3} \tag{1.1.16}
$$

para todo $n \geq n_0$.

Seja C o limite de convergência da série numérica $(f_n'(x_0))_{n \in \mathbb{N}}$. Então, $\forall n \geq n_0$

$$
|f_n'(x_0) - C| < \frac{\varepsilon}{3}. \tag{1.1.17}
$$

Como, por hipótese, f_n é diferenciável em x_0, então existe $\delta \in \mathbb{R}$, com $\delta > 0$, tal que, se $|x - x_0| < \delta$, então

$$
\left| \frac{f_n(x) - f_n(x_0)}{x - x_0} - f_n'(x_0) \right| < \frac{\varepsilon}{3}. \tag{1.1.18}
$$

Das equações 1.1.16, 1.1.17 e 1.1.18 temos que

$$\begin{aligned}\left|\frac{f(x)-f(x_0)}{x-x_0}-C\right| &= \left|\frac{f(x)-f(x_0)}{x-x_0}-\frac{f_n(x)-f_n(x_0)}{x-x_0}+\frac{f_n(x)-f_n(x_0)}{x-x_0}-\right.\\ &\quad \left.-f_n'(x_0)+f_n'(x_0)-C\right|\\ &\leq \left|\frac{f(x)-f(x_0)}{x-x_0}-\frac{f_n(x)-f_n(x_0)}{x-x_0}\right|+\\ &\quad +\left|\frac{f_n(x)-f_n(x_0)}{x-x_0}-f_n'(x_0)\right|+\\ &\quad +|f_n'(x_0)-C|\\ &\leq \frac{\varepsilon}{3}+\frac{\varepsilon}{3}+\frac{\varepsilon}{3}=\varepsilon\end{aligned} \tag{1.1.19}$$

para $0<|x-x_0|\leq\varepsilon$, provando assim que existe $f'(x_0)$ e que é o número C, que podemos chamar de $\lim_{n\to\infty} f_n(x_0)$. Como x_0 é qualquer, está provado o teorema, ou seja, $\frac{d}{dx}\lim_{n\to\infty} f_n(x)=\lim_{n\to\infty}\frac{d}{dx}f_n(x)$.

■

1.1.2 Séries de Funções

Com os resultados enunciados acima, podemos dar prosseguimento a um tópico fundamental para estudarmos as Séries de Fourier. O que mostraremos a seguir é o início do conteúdo de Séries de Funções. Nosso principal objetivo nesta seção é trabalharmos alguns dos resultados fundamentais para a construção das Séries de Fourier, principalmente, relacionados à convergência uniforme, continuidade, integração e derivação de séries de funções.

Definição 1.1.5. *Seja $\{f_n : n\in\mathbb{N}\}$ uma sequência de funções definidas sobre um domínio D. Fixado $m\in\mathbb{N}$, chamamos de Sequência das Somas Parciais a sequência*

$$S_m=\sum_{k=1}^{m} f_k(x)=f_1(x)+f_2(x)+\cdots+f_m(x),$$

para todo $x\in D$.

Se $\forall x\in D$, existe $\lim_{m\to\infty} S_m(x)$, dizemos que a série $\sum_{k=1}^{\infty} f_k(x)$ converge pontual-

mente em D para uma função f, definida por

$$f(x) = \sum_{k=1}^{\infty} f_k(x) \quad \left(= \lim_{m\to\infty} \sum_{k=1}^{m} f_k(x) \right).$$

Precisaremos de alguns resultados de séries de funções para responder às seguintes perguntas:

- Se f_k é contínuo em $D, \forall k \in \mathbb{N}$, podemos afirmar que $f(x) = \sum_{k=1}^{\infty} f_k(x)$ é uma função contínua sobre D?
- Se f_k é diferenciável em seu domínio, $\forall k \in \mathbb{N}$, podemos afirmar que
$$f'(x) = \sum_{k=1}^{\infty} f_k'(x)?$$
- Se f_k for integrável em seu domínio, $\forall k \in \mathbb{N}$, podemos afirmar que
$$\int_a^b f(x)dx = \sum_{k=1}^{\infty} \int_a^b f_k(x)dx?$$

Definição 1.1.6. *Seja $\{f_n : n \in \mathbb{N}\}$ uma sequência de funções definida sobre um domínio D. A série $\sum_{k=1}^{\infty} f_k(x)$ converge uniformemente se a sequência das somas parciais $(S_n : n \in \mathbb{N})$ convergir uniformemente.*

Proposição 1.1.2. *Seja $\{f_n : n \in \mathbb{N}\}$ uma sequência de funções definida sobre um domínio D. A série $\sum_{k=1}^{\infty} f_k(x)$ converge uniformemente se e somente se, dado $\varepsilon \in \mathbb{R}, \varepsilon > 0$, existe $n_0 \in \mathbb{N}$ tal que $\left|\sum_{k=n}^{m} f_k(x)\right| < \varepsilon$, $\forall x \in D$, para todos $m > n \geq n_o$.*

Demonstração. ($\Rightarrow$) Temos, por hipótese, que a série $\sum_{k=1}^{\infty} f_k(x)$ converge uniformemente. Então, pela definição, a sequência das somas parciais converge uniformemente, ou seja, $\forall \varepsilon \in \mathbb{R}, \varepsilon > 0$, existe $n_0 \in \mathbb{N}$ tal que

$$|S_m(x) - S_n(x)| < \varepsilon$$

para todo $x \in D$ e para todos $m > n \geq n_0$. Então:

$$\begin{aligned} \left|\sum_{k=n}^{m} f_k(x)\right| &= |f_n + f_{n+1} + \cdots + f_m| \\ &= |f_1 + f_2 + \cdots + f_m - f_1 - f_2 - \cdots - f_n| \\ &= |S_m(x) - S_n(x)| \\ &< \varepsilon \end{aligned} \tag{1.1.20}$$

para todos $x \in D$ e para todo $m > n \geq n_0$.

($\Leftarrow$) A hipótese nos diz que $\forall \varepsilon > 0, \exists n_0 \in \mathbb{N}$ tal que $\left|\sum_{k=n}^{m} f_k(x)\right| < \varepsilon$, para todos $m > n \geq n_0$. Então:

$$\begin{aligned} |S_m(x) - S_n(x)| &= |f_1 + f_2 + \cdots + f_m - f_1 - f_2 - \cdots - f_n| \\ &= |f_{n+1} + \cdots + f_m| \\ &= \left|\sum_{k=n+1}^{m} f_k(x)\right| \\ &< \varepsilon \end{aligned} \tag{1.1.21}$$

para todo $x \in D$, para todos $m > n \geq n_0$. ■

Proposição 1.1.3. *Seja* $\sum_{k=1}^{\infty} f_k(x)$ *uma série uniformemente convergente de funções contínuas, definidas em um intervalo* D*. Então, sua soma* $f(x) = \sum_{k=1}^{\infty} f_k(x)$ *é uma função contínua.*

Demonstração. Como a série $\sum_{k=1}^{\infty} f_k(x)$ converge uniformemente, temos que, pela definição de convergência uniforme de séries, a sequência das somas parciais converge uniformemente.

Logo, pelo Teorema 1.1.2, a sua soma $f(x) = \lim_{m\to\infty} \sum_{k=1}^{m} f_k(x) = \sum_{k=1}^{\infty} f_k(x)$ é uma função contínua, como queríamos demonstrar. ■

Definição 1.1.7. *A série* $\sum_{k=1}^{\infty} f_k(x)$ *converge absolutamente se* $\sum_{k=1}^{\infty} |f_k(x)|$ *converge. Se* $\sum_{k=1}^{\infty} |f_k(x)|$ *convergir uniformemente (ou simplesmente), dizemos que a série* $\sum_{k=1}^{\infty} f_k(x)$ *converge absoluta e uniformemente (ou simplesmente).*

Uma ferramenta importante para determinarmos convergência uniforme de séries de funções é dada pelo *Teste M de Weierstrass*, que enunciaremos a seguir:

Teorema 1.1.6 (Teste M de Weierstrass)**.** *Seja* $\sum_{k=1}^{\infty} f_k(x)$ *uma série de funções definidas em um conjunto* $D \subset \mathbb{R}$ *e seja* $\{M_n\}_{n\in\mathbb{N}}$ *uma sequência de constantes positivas. Se* $\sum_{k=0}^{\infty} M_k$ *converge e se*

$$|f_k(x)| \leq M_k$$

para todo $x \in D$*,* $k = 0, 1, 2 \cdots$*, então a série* $\sum_{k=1}^{\infty} f_k(x)$ *converge uniformemente em* D*.*

Demonstração. Temos por hipótese que $\sum_{k=0}^{\infty} M_k$ converge. Logo, $\forall \varepsilon > 0, \exists n_0 \in \mathbb{N}$ tal que

$$\left| \sum_{k=n}^{m} M_k \right| < \varepsilon,$$

$\forall m > n \geq n_0$. Como temos a hipótese que $\{M_n\}_{n\in\mathbb{N}}$ é uma sequência de constantes positivas, podemos afirmar que

$$\sum_{k=n}^{m} M_k < \varepsilon.$$

Logo:

$$\begin{aligned} \left| \sum_{k=n}^{m} f_k \right| &= |f_n + f_{n+1} + f_{n+2} + \cdots + f_m| \\ &\leq |f_n| + |f_{n+1}| + |f_{n+2}| + \cdots + |f_m| \\ &\leq \sum_{k=n}^{m} |f_k| \\ &\leq \sum_{k=n}^{m} M_n \\ &< \varepsilon. \end{aligned} \tag{1.1.22}$$

Portanto, pela Proposição 1.1.2, temos que a série $\sum_{k=1}^{\infty} f_k(x)$ converge uniformemente sobre D. ■

Proposição 1.1.4. *Seja $\sum_{k=1}^{\infty} f_k(x)$ uma série de funções integráveis, definidas em um conjunto $[a, b] \subset \mathbb{R}$, que converge uniformemente para uma função f em $[a, b]$. Então f é integrável e*

$$\int_a^b f(x)dx = \sum_{k=1}^{\infty} \int_a^b f_k(x)dx.$$

Demonstração. Como $\sum_{k=1}^{\infty} f_k(x)$ é uma série de funções contínuas, convergindo uniformemente em $[a, b]$, temos, pela Proposição 1.1.3, que $f(x) = \sum_{k=1}^{\infty} f_k(x)$ é uma função contínua em $[a, b]$. Segue que f é uma função Integrável em $[a, b]$ e

$$\int_a^b f(x)dx = \int_a^b \sum_{k=1}^{\infty} f_k(x)dx.$$

Agora, basta provarmos que $\int_a^b \sum_{k=1}^{\infty} f_k(x)dx = \sum_{k=1}^{\infty} \int_a^b f_k(x)dx$. Da convergência uniforme da série $\sum_{k=1}^{\infty} f_k(x)$, temos que a sequência das somas parciais $\{S_n(x) : n \in \mathbb{N}\}$ converge uniformemente, para todo $x \in [a, b]$. Logo, pelo Teorema 1.1.4 temos que

$$\begin{aligned} \int_a^b f(x)dx = \int_a^b \sum_{k=1}^{\infty} f_k(x)dx &= \int_a^b \lim_{n\to\infty} S_n(x)dx \\ &= \lim_{n\to\infty} \int_a^b S_n(x)dx \\ &= \sum_{k=1}^{\infty} \int_a^b f_k(x)dx. \end{aligned} \tag{1.1.23}$$

■

Proposição 1.1.5. *Seja* $(f_n(x) : n \in \mathbb{N})$ *uma sequência de funções deriváveis, cujas derivadas são contínuas no intervalo* $[a, b]$. *Suponha que a série* $\sum_{k=1}^{\infty} f_k(x)$ *convirja uniformemente para uma função* $f(x) = \sum_{k=1}^{\infty} f_k(x)$ *em* $[a, b], \forall x \in [a, b]$. *Se a série* $\sum_{k=1}^{\infty} f_k'(x)$ *convergir uniformemente em* $[a, b]$, *então*

$$f'(x) = \sum_{k=1}^{\infty} f_k'(x), \forall x \in [a, b].$$

Demonstração. Como a série $\sum_{k=1}^{\infty} f_k(x)$ converge uniformemente, temos que a sequência das somas parciais $(S_n : n \in \mathbb{N})$ converge uniformemente. Segue do Teorema 1.1.5 que

$$\frac{d}{dx} f(x) = \frac{d}{dx} \sum_{k=1}^{\infty} f_k(x) = \frac{d}{dx} \lim_{n\to\infty} S_n(x) = \lim_{n\to\infty} \frac{d}{dx} S_n(x) = \sum_{k=1}^{\infty} \frac{d}{dx} f_k(x).$$

■

Com estes resultados podemos iniciar a seção das Séries de Fourier. Tais séries em particular são importantes em nosso trabalho pois dão suporte à Transformada de Fourier, ferramenta fundamental para a admissibilidade das aplicações que faremos neste trabalho.

1.2 Série de Fourier

Neste capítulo, traremos alguns resultados para responder a seguinte questão: quais funções $f : \mathbb{R} \longrightarrow \mathbb{R}$ podem ser expressas por sua Série de Fourier, ou seja

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(\frac{n\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right). \tag{1.2.1}$$

1.2.1 Funções Periódicas

Definição 1.2.1. *Uma função $f : \mathbb{R} \to \mathbb{R}$ é periódica de período T se $f(x+T) = f(x), \forall x \in \mathbb{R}$.*

Se existe um $T \neq 0$ tal que T é o menor período positivo de uma função f, dizemos que T é o período fundamental de f. Podemos dizer assim, que todo múltiplo nT, com $n \in \mathbb{Z}$ também é período de f. De fato,

$$f(x+nT) = f(x+(n-1)T+T) = f(x+(n-1)T) = f(x), \quad n = 0, \pm 1, \pm 2, \ldots$$

Exemplo 1.2.1. *1. A função $\sin x$ é periódica e de período 2π.*

2. Considere a função $f : \mathbb{R} \to \mathbb{R}, f(x) = \sin \dfrac{n\pi x}{L}, \quad n \geq 1, \quad n \in \mathbb{N}$. Vamos descobrir qual seu período fundamental T. Observe que para T ser período fundamental de f, temos que ter

$$f(x+T) = \sin \frac{n\pi(x+T)}{L} = \sin \frac{n\pi x}{L} = f(x), \text{ para todo } x \in \mathbb{R},$$

ou seja,

$$\sin \frac{n\pi(x+T)}{L} = \sin \frac{n\pi x}{L} \cdot \cos \frac{n\pi T}{L} + \cos \frac{n\pi x}{L} \cdot \sin \frac{n\pi T}{L} = \sin \frac{n\pi x}{L}$$

Para $x = \dfrac{L}{2n}$, temos

$$\sin \frac{\pi}{2} \cdot \cos \frac{n\pi T}{L} = \sin \frac{\pi}{2},$$

o que implica em

$$\cos \frac{n\pi T}{L} = 1$$

e pela identidade trigonométrica $\sin^2 \theta + \cos^2 \theta = 1$, temos que

$$\sin \frac{n\pi T}{L} = 0.$$

Como estamos interessados no período fundamental de f, e pelas condições expostas anteriormente, segue que $\dfrac{n\pi T}{L} = 2\pi$, ou seja, o período fundamental de $f(x) = \sin \dfrac{n\pi x}{L}$ é $T = \dfrac{2L}{n}$.

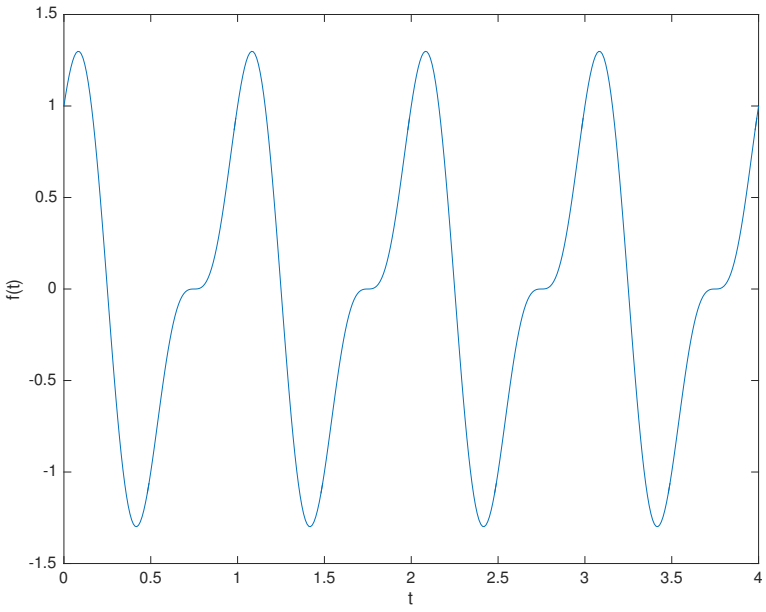


Figura 1.2.1 – Gráfico da função $f(t) = \cos 2\pi t + \frac{1}{2}\sin 4\pi t$ no intervalo $[0, 4] \subset \mathbb{R}$

3. *Consideremos a função* $f(t) = \cos 2\pi t + \frac{1}{2}\sin 4\pi t$*, cujo gráfico está representado na Figura 1.2.1.*

   Observe que o período fundamental da função $f(t) = \cos 2\pi t + \frac{1}{2}\sin 4\pi t$ *é* $T = 1$*. Com efeito,*

$$\begin{aligned} f(t+1) &= \cos(2\pi(t+1)) + \frac{1}{2}\sin(4\pi(t+1)) \\ &= \cos(2\pi t + 2\pi) + \frac{1}{2}\sin(4\pi t + 4\pi) \\ &= \cos 2\pi t + \frac{1}{2}\sin 4\pi t \\ &= f(t). \end{aligned}$$

1.2.2 Coeficientes de Fourier

Seja $f : \mathbb{R} \longrightarrow \mathbb{R}$ uma função periódica, de período $2L$. Mostraremos mais à frente, no teorema 1.2.2, a convergência uniforme das séries de Fourier. Então, para determinarmos os coeficientes de Fourier, vamos supor que a série

$$\sum_{n=1}^{\infty} a_n \cos\left(\frac{n\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right)$$

convirja uniformemente e que podemos expressar f como

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(\frac{n\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right), \tag{1.2.2}$$

com os coeficientes a_0, a_n *e* b_n a obter. Pela proposição 1.1.3, podemos afirmar que a função f é contínua, logo pode ser integrada. Podemos afirmar também, pela igualdade, que a função f tem período fundamental $2L$, pois este é o período fundamental de $\cos\left(\frac{n\pi x}{L}\right)$ e $\sin\left(\frac{n\pi x}{L}\right)$, pois, como a soma de finita de funções periódicas, de mesmo período, tem o mesmo período, a sequência de somas parciais $S_N = \sum_{n=1}^{N} \cos\left(\frac{n\pi x}{L}\right) + \sin\left(\frac{n\pi x}{L}\right)$ tem o mesmo período de $\cos\left(\frac{n\pi x}{L}\right)$ e $\sin\left(\frac{n\pi x}{L}\right)$.

Portanto, pela proposição 1.1.4, integraremos ambos os lados de 1.2.2 para obter

$$\int_{-L}^{L} f(x)dx = \frac{a_0}{2}\int_{-L}^{L} dx + \sum_{n=1}^{\infty} a_n \int_{-L}^{L} \cos\left(\frac{n\pi x}{L}\right)dx + b_n \int_{-L}^{L} \sin\left(\frac{n\pi x}{L}\right)dx.$$

Como

$$\int_{-L}^{L} dx = 2L,$$

$$\int_{-L}^{L} \cos\left(\frac{n\pi x}{L}\right)dx = \frac{L}{n\pi}\sin\left(\frac{n\pi L}{L}\right) - \frac{L}{n\pi}\sin\left(\frac{n\pi(-L)}{L}\right) = \frac{L}{n\pi}\left[\sin(n\pi) - \sin(-n\pi)\right] = 0$$

e

$$\int_{-L}^{L} \sin\left(\frac{n\pi x}{L}\right)dx = -\frac{L}{n\pi}\cos\left(\frac{n\pi L}{L}\right) + \frac{L}{n\pi}\cos\left(\frac{n\pi(-L)}{L}\right) = -\frac{L}{n\pi}\left[\cos(n\pi) - \cos(n\pi)\right] = 0,$$

obtemos que

$$a_0 = \frac{1}{L}\int_{-L}^{L} f(x)dx. \tag{1.2.3}$$

Para obtermos os termos a_n e b_n, vamos usar as *Relações de Ortogonalidade*:

Teorema 1.2.1 (Relações de Ortogonalidade)**.** *Se* $m, n \in \mathbb{Z}_+^*$*, então*

$$\int_{-L}^{L} \cos\frac{n\pi x}{L}\sin\frac{m\pi x}{L}dx = \ 0, \quad se \quad n, m \geq 1 \tag{1.2.4}$$

$$\int_{-L}^{L} \cos\frac{n\pi x}{L}\cos\frac{m\pi x}{L}dx = \begin{cases} L, & se \qquad n = m \geq 1 \\ 0, & se \quad n \neq m, n, m \geq 1 \end{cases} \tag{1.2.5}$$

$$\int_{-L}^{L} \sin\frac{n\pi x}{L}\sin\frac{m\pi x}{L}dx = \begin{cases} L, & se \qquad n = m \geq 1 \\ 0, & se \quad n \neq m, n, m \geq 1 \end{cases} \tag{1.2.6}$$

Antes de demonstrarmos o Teorema acima, vamos relembrar algumas relações trigonométricas:

$$\sin(\theta + \tau) = \sin(\theta)\cos(\tau) + \sin(\tau)\cos(\theta) \tag{1.2.7}$$

$$\sin(\theta - \tau) = \sin(\theta)\cos(\tau) - \sin(\tau)\cos(\theta) \tag{1.2.8}$$

$$\cos(\theta + \tau) = \cos(\theta)\cos(\tau) - \sin(\theta)\sin(\tau) \tag{1.2.9}$$

$$\cos(\theta - \tau) = \cos(\theta)\cos(\tau) + \sin(\theta)\sin(\tau) \tag{1.2.10}$$

Somando (1.2.9) com (1.2.10), subtraindo (1.2.10) por 1.2.9 e subtraindo (1.2.7) por (1.2.8), obtemos, respectivamente

$$2\cos(\theta)\cos(\tau) = \cos(\theta + \tau) + \cos(\theta - \tau). \tag{1.2.11}$$

$$2\sin(\theta)\sin(\tau) = \cos(\theta + \tau) - \cos(\theta - \tau). \tag{1.2.12}$$

$$2\cos(\theta)\sin(\tau) = \sin(\theta + \tau) - \sin(\theta - \tau) \tag{1.2.13}$$

Com essas relações trigonométricas, podemos demonstrar o Teorema 1.2.1.

Demonstração (1.2.1). Vamos mostrar, primeiro, a relação (1.2.4):

Sejam $n, m \in \mathbb{N}^*$ quaisquer. Pela equação (1.2.13), temos que:

$$\begin{aligned}
\int_{-L}^{L} \cos\frac{n\pi x}{L}\sin\frac{m\pi x}{L}dx &= \frac{1}{2}\int_{-L}^{L}\left[\sin\left(\frac{n\pi x}{L}+\frac{m\pi x}{L}\right)-\sin\left(\frac{n\pi x}{L}-\frac{m\pi x}{L}\right)\right]dx \\
&= \frac{1}{2}\int_{-L}^{L}\sin\left(\frac{(n+m)\pi x}{L}\right)dx- \\
&\quad -\frac{1}{2}\int_{-L}^{L}\sin\left(\frac{(n-m)\pi x}{L}\right)dx \\
&= \frac{-L}{2\pi(n+m)}\left[\cos[(n+m)\pi]-\cos[-(n+m)\pi]\right]- \\
&\quad -\frac{-L}{2\pi(n-m)}\left[\cos[(n-m)\pi]-\cos[-(n-m)\pi]\right] \\
&= 0,
\end{aligned}$$

pois $\cos(\theta)=\cos(-\theta), \forall\theta\in\mathbb{R}$.

Para mostrarmos a relação (1.2.5), vamos considerar $n, m \in \mathbb{Z}_+^*$, tais que:

- Se $n \neq m$, temos, usando a relação (1.2.11):

$$\begin{aligned}
\int_{-L}^{L} \cos\frac{n\pi x}{L}\cos\frac{m\pi x}{L}dx &= \frac{1}{2}\int_{-L}^{L}\cos\left(\frac{n\pi x}{L}+\frac{m\pi x}{L}\right)+\cos\left(\frac{n\pi x}{L}-\frac{m\pi x}{L}\right)dx \\
&= \frac{1}{2}\int_{-L}^{L}\cos\left(\frac{n\pi x}{L}+\frac{m\pi x}{L}\right)dx+ \\
&\quad +\frac{1}{2}\int_{-L}^{L}\cos\left(\frac{n\pi x}{L}-\frac{m\pi x}{L}\right)dx \\
&= \frac{L}{2\pi(n+m)}\left[\sin[(n+m)\pi]-\sin[(n+m)\pi]\right]+ \\
&\quad -\frac{L}{2\pi(n+m)}\sin[(n-m)\pi]-\sin[(n+m)\pi] \\
&= 0,
\end{aligned}$$

pois $\sin(k\pi)=0, \forall k\in\mathbb{Z}$.

- Se $n = m$, temos, usando a identidade trigonométrica $\cos^2(\theta)=\frac{1}{2}\cos(2\theta)+1$:

$$\begin{aligned}\int_{-L}^{L}\cos\frac{n\pi x}{L}\cos\frac{m\pi x}{L}dx &= \frac{1}{2}\int_{-L}^{L}\cos^2\left(\frac{2n\pi x}{L}\right)+1dx\\ &= \frac{1}{2}\int_{-L}^{L}\cos\left(\frac{2n\pi x}{L}\right)dx+\frac{1}{2}\int_{-L}^{L}dx\\ &= \frac{L}{4\pi n}[\sin(2n\pi)-\sin(-2n\pi)]+\frac{1}{2}[L-(-L)]\\ &= L,\end{aligned}$$

pois $\sin(k\pi)=0, \forall k\in\mathbb{Z}$.

Para mostrarmos a relação (1.2.6), vamos considerar $n, m \in \mathbb{N}^*$, tais que:

- Se $n \neq m$, temos, usando a relação (1.2.12):

$$\begin{aligned}\int_{-L}^{L}\sin\frac{n\pi x}{L}\sin\frac{m\pi x}{L}dx &= \frac{1}{2}\int_{-L}^{L}\cos\left(\frac{n\pi x}{L}+\frac{m\pi x}{L}\right)-\cos\left(\frac{n\pi x}{L}-\frac{m\pi x}{L}\right)dx\\ &= \frac{1}{2}\int_{-L}^{L}\cos\left(\frac{n\pi x}{L}+\frac{m\pi x}{L}\right)dx-\\ &\quad -\frac{1}{2}\int_{-L}^{L}\cos\left(\frac{n\pi x}{L}-\frac{m\pi x}{L}\right)dx\\ &= \frac{L}{2\pi(n+m)}\left[\sin[(n+m)\pi]-\sin[(n+m)\pi]\right]-\\ &\quad -\frac{L}{2\pi(n+m)}\sin[(n-m)\pi]-\sin[(n+m)\pi]\\ &= 0,\end{aligned}$$

pois $\sin(k\pi)=0, \forall k\in\mathbb{Z}$.

- Se $n = m$, temos, usando a identidade trigonométrica $\sin^2(\theta)=\frac{1}{2}[1-\cos(2\theta)]$:

$$\begin{aligned}\int_{-L}^{L} \sin\frac{n\pi x}{L}\sin\frac{m\pi x}{L}dx &= \frac{1}{2}\int_{-L}^{L} 1-\cos^2\left(\frac{2n\pi x}{L}\right)dx \\ &= \frac{1}{2}\int_{-L}^{L} dx - \frac{1}{2}\int_{-L}^{L}\cos\left(\frac{2n\pi x}{L}\right)dx \\ &= \frac{1}{2}[L-(-L)] - \frac{L}{4\pi n}[\sin(2n\pi)-\sin(-2n\pi)] \\ &= L,\end{aligned}$$

pois $\sin(k\pi) = 0, \forall k \in \mathbb{Z}$.

■

O teorema acima é chamado de *Relações de Ortogonalidade* por conta do produto interno no espaço de funções, ou seja, sejam f e g funções no espaço de funções com produto interno definido por

$$\langle f, g\rangle = \int_{-\infty}^{\infty} f(x)\cdot g(x)dx.$$

Dizemos que f e g são ortogonais se $\langle f, g\rangle = 0$.

Com as Relações de ortogonalidade bem definidas, podemos encontrar os coeficientes de Fourier. Para encontrarmos a_n, multiplicaremos a expressão

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n\cos\left(\frac{n\pi x}{L}\right) + b_n\sin\left(\frac{n\pi x}{L}\right)$$

por $\cos\left(\frac{m\pi x}{L}\right)$, em ambos os lados, para $m \geq 1$ fixado, obtendo:

$$f(x)\cos\left(\frac{m\pi x}{L}\right) = \frac{a_0}{2}\cos\left(\frac{m\pi x}{L}\right) + \sum_{n=1}^{\infty} a_n\cos\left(\frac{n\pi x}{L}\right)\cos\left(\frac{m\pi x}{L}\right) + b_n\sin\left(\frac{n\pi x}{L}\right)\cos\left(\frac{m\pi x}{L}\right).$$

Ao integrarmos em ambos os lados, temos

$$\begin{aligned}\int_{-L}^{L} f(x)\cos\left(\frac{m\pi x}{L}\right) &= \int_{-L}^{L}\frac{a_0}{2}\cos\left(\frac{m\pi x}{L}\right) + \\ &\quad + \int_{-L}^{L}\sum_{n=1}^{\infty} a_n\cos\left(\frac{n\pi x}{L}\right)\cos\left(\frac{m\pi x}{L}\right) + b_n\sin\left(\frac{n\pi x}{L}\right)\cos\left(\frac{m\pi x}{L}\right) \\ &= \int_{-L}^{L}\frac{a_0}{2}\cos\left(\frac{m\pi x}{L}\right) + \\ &\quad + \sum_{n=1}^{\infty}\int_{-L}^{L} a_n\cos\left(\frac{n\pi x}{L}\right)\cos\left(\frac{m\pi x}{L}\right) + \\ &\quad + \int_{-L}^{L} b_n\sin\left(\frac{n\pi x}{L}\right)\cos\left(\frac{m\pi x}{L}\right).\end{aligned}$$

Usando as relações de ortogonalidade, obteremos a seguinte expressão para a_n:

$$a_n = \frac{1}{L}\int_{-L}^{L} f(x)\cos\left(\frac{m\pi x}{L}\right)dx \tag{1.2.14}$$

De forma análoga, ao multiplicarmos a expressão (1.2.2) por $\sin\left(\frac{m\pi x}{L}\right)$, em ambos os lados, para $m \geq 1$ fixado, obtemos

$$f(x)\sin\left(\frac{m\pi x}{L}\right) = \frac{a_0}{2}\sin\left(\frac{m\pi x}{L}\right) + \sum_{n=1}^{\infty} a_n \cos\left(\frac{n\pi x}{L}\right)\sin\left(\frac{m\pi x}{L}\right) + b_n \sin\left(\frac{n\pi x}{L}\right)\sin\left(\frac{m\pi x}{L}\right).$$

Ao integrarmos ambos os lados, e usando as relações de ortogonalidade, obteremos a seguinte expressão para b_n:

$$b_n = \frac{1}{L}\int_{-L}^{L} f(x)\sin\left(\frac{m\pi x}{L}\right)dx \tag{1.2.15}$$

Definição 1.2.2. *Seja $f : \mathbb{R} \to \mathbb{C}$ uma função integrável. Dizemos que $f \in \mathscr{L}_1$ se*

$$\int_{-\infty}^{\infty} |f(x)|dx < \infty \tag{1.2.16}$$

Assim, podemos definir a série de Fourier de uma função f da seguinte forma:

Definição 1.2.3. *Seja $f : \mathbb{R} \longrightarrow \mathbb{R}$ uma função periódica, de período $2L$, $f \in \mathscr{L}_1$. Dizemos que a **Série de Fourier** de f é dada por*

$$f(x) \sim \frac{1}{2}a_0 + \sum_{n=1}^{\infty}\left(a_n \cos\frac{n\pi x}{L} + b_n \sin\frac{n\pi x}{L}\right). \tag{1.2.17}$$

em que

$$a_0 = \frac{1}{L}\int_{-L}^{L} f(x)dx,$$

$$a_n = \frac{1}{L}\int_{-L}^{L} f(x)\cos\left(\frac{m\pi x}{L}\right)dx \text{ e}$$

$$b_n = \frac{1}{L}\int_{-L}^{L} f(x)\sin\left(\frac{m\pi x}{L}\right)dx.$$

Podemos nos perguntar o seguinte: quando a série de Fourier de uma função está bem definida? Quais as condições necessárias e suficientes para que a série de Fourier convirja uniformemente para a sua função f. Neste sentido vamos trazer alguns resultados que irão nos ajudar a demonstrar a convergência uniforme das séries de Fourier.

1.2.2.1 Estimativas dos Coeficientes de Fourier

Nesta sessão iremos encontrar estimativas para os coeficientes de Fourier de uma função dada. Assim:

i. Supondo que f seja periódica, com período $2L$, $f \in \mathscr{L}_1$, podemos obter as seguintes estimativas:

$$|a_n| = \left| \frac{1}{L} \int_{-L}^{L} f(x) \cos \frac{n\pi x}{L} dx \right| \leq \frac{1}{L} \int_{-L}^{L} |f(x)| \, dx. \tag{1.2.18}$$

Além disso,

$$|b_n| = \left| \frac{1}{L} \int_{-L}^{L} f(x) \sin \frac{n\pi x}{L} dx \right| \leq \frac{1}{L} \int_{-L}^{L} |f(x)| \, dx. \tag{1.2.19}$$

Com isso, temos que se $f \in \mathscr{L}_1$, existe uma constante $M \in \mathbb{R}$ tal que

$$|a_n| \leq M \ e \ |b_n| \leq M, \forall n \in \mathbb{N}^*.$$

ii. Agora, suponhamos que f seja periódica, de período $2L$, derivável de modo que $f' \in \mathscr{L}_1$. Então, temos que:

$$a_n = \frac{1}{L} \int_{-L}^{L} f(x) \cos \frac{n\pi x}{L} dx.$$

Ao integrarmos por partes, obtemos

$$a_n = \frac{L}{Ln\pi} \, f(x) \sin \frac{n\pi x}{L} \Big|_{-L}^{L} - \frac{L}{Ln\pi} \int_{-L}^{L} f'(x) \sin \frac{n\pi x}{L} dx.$$

Como $f(x) \sin \frac{n\pi x}{L} \Big|_{-L}^{L} = 0$, obtemos

$$a_n = -\frac{1}{n\pi} \int_{-L}^{L} f'(x) \sin \frac{n\pi x}{L} dx$$

e, tomando os valores absolutos,

$$|a_n| \leq \frac{1}{n\pi} \int_{-L}^{L} |f'(x)| \, dx. \tag{1.2.20}$$

De maneira análoga, temos:

$$b_n = \frac{1}{L} \int_{-L}^{L} f(x) \sin \frac{n\pi x}{L} dx.$$

Ao integrarmos por partes, obtemos

$$b_n = \frac{-L}{Ln\pi}\, f(x)\cos\frac{n\pi x}{L}\Big|_{-L}^{L} + \frac{L}{Ln\pi}\int_{-L}^{L} f'(x)\cos\frac{n\pi x}{L}dx.$$

Como f é periódica de período $2L$, temos que

$$\frac{-L}{Ln\pi}\, f(x)\cos\frac{n\pi x}{L}\Big|_{-L}^{L} = \frac{-L}{Ln\pi}[f(L)\cos(n\pi) - f(-L)\cos(-n\pi)] = 0.$$

Portanto, segue que

$$b_n = \frac{1}{n\pi}\int_{-L}^{L} f'(x)\cos\frac{n\pi x}{L}dx$$

e tomando os valores absolutos, temos

$$|b_n| \leq \frac{1}{n\pi}\int_{-L}^{L} |f'(x)|\, dx. \tag{1.2.21}$$

Portanto, quando temos uma função f contínua e que sua derivada $f' \in \mathscr{L}_1$, conclui-se que existe uma constante M de modo que

$$|a_n| \leq \frac{M}{n}, |b_n| \leq \frac{M}{n}, \forall n \in \mathbb{N}^*.$$

iii. Agora, se supusermos que f seja periódica de período $2L$, com primeira derivada contínua e $f'' \in \mathscr{L}_1$, podemos melhorar as estimativas dos coeficientes de Fourier. Se integrarmos a equação

$$a_n = -\frac{1}{n\pi}\int_{-L}^{L} f'(x)\sin\frac{n\pi x}{L}dx \tag{1.2.22}$$

por partes, obtemos

$$a_n = -\frac{1}{n\pi}\left\{-f'(x)\frac{L}{n\pi}\cos\frac{n\pi x}{L}\Big|_{-L}^{L} + \frac{L}{n\pi}\int_{-L}^{L} f''(x)\cos\frac{n\pi x}{L}dx\right\}$$

e então, tirando o valor absoluto, temos

$$|a_n| \leq \frac{L}{n^2\pi^2}\int_{-L}^{L} |f''(x)|dx \tag{1.2.23}$$

Para estimarmos b_n, vamos integrar por partes a equação

$$b_n = \frac{1}{n\pi}\int_{-L}^{L} f'(x)\cos\frac{n\pi x}{L}dx \tag{1.2.24}$$

para obtermos

$$b_n = -\frac{1}{n\pi}\left\{f'(x)\frac{L}{n\pi}\sin\frac{n\pi x}{L}\Big|_{-L}^{L} - \frac{L}{n\pi}\int_{-L}^{L} f''(x)\sin\frac{n\pi x}{L}dx\right\},$$

e então

$$|b_n| \leq \frac{L}{n^2\pi^2} \int_{-L}^{L} |f''(x)|dx. \tag{1.2.25}$$

Portanto, existe uma constante M de modo que

$$|a_n| \leq \frac{M}{n^2}, |b_n| \leq \frac{M}{n^2}, \forall n \in \mathbb{N}^*.$$

Tais estimativas são melhores pois, à medida que $n \longrightarrow \infty$, a sequência $\left(\frac{1}{n^2}\right)_{n\in\mathbb{N}}$ converge mais rapidamente para 0 que a sequência $\left(\frac{1}{n}\right)_{n\in\mathbb{N}}$.

1.2.3 Convergência Uniforme da Série de Fourier

Dadas as estimativas dos coeficientes de Fourier, podemos caminhar para demonstrarmos o Teorema da Convergência Uniforme das Séries de Fourier. Para isso, vamos trazer duas desigualdades importantes para a demonstração: a Desigualdade de Bessel e a Desigualdade de Cauchy-Schwarz.

Desigualdade de Bessel e Desigualdade de Cauchy-Schwarz

Definição 1.2.4. *Seja $f : \mathbb{R} \to \mathbb{C}$ uma função. Dizemos que $f \in \mathscr{L}_2$ se*

$$\int_{-\infty}^{\infty} |f(x)|^2 dx < \infty \tag{1.2.26}$$

Neste caso, denotamos o valor dado em (1.2.26) de $||f||_{\mathscr{L}_2}$.

Para construirmos a Desigualdade de Bessel, precisaremos considerar as *Reduzidas das séries de Fourier*, ou seja, fixado $n \in \mathbb{N}^*$, a função

$$s_n = \frac{a_0}{2} + \sum_{k=1}^{n} a_k \cos\frac{k\pi x}{L} + b_k \sin\frac{k\pi x}{L},$$

em que a_0, a_k e b_k são os coeficientes de Fourier.

Definição 1.2.5. *Seja $(f_n)_{n\in\mathbb{N}}$ uma sequência de funções tais que $f_n \in \mathscr{L}_2, \forall n \in \mathbb{N}$, em um intervalo $[a, b]$. Dizemos que $(f_n)_{n\in\mathbb{N}}$ converge em média quadrática para uma função $f \in \mathscr{L}_2$ se*

$$\lim_{n\to\infty} \int_a^b |f_n(x) - f(x)|^2 dx.$$

A expressão

$$\int_a^b |f_n(x) - f(x)|^2 dx$$

*é chamada **o erro médio quadrático**, na aproximação de f por f_n.*

Nosso objetivo é mostrar que as reduzidas $s_n(x)$ da série de Fourier de uma função $f \in \mathscr{L}_2$ são os polinômios trigonométricos que melhor aproximam de f em média quadrática. Para isso, consideremos um polinômio trigonométrico de ordem n:

$$t_n = \frac{c_0}{2} + \sum_{k=1}^{n} c_k \cos\frac{k\pi x}{L} + d_k \sin\frac{k\pi x}{L}.$$

Sendo

$$\epsilon_n = \int_{-L}^{L} |s_n(x) - f(x)|^2 dx$$

e

$$\hat{\epsilon}_n = \int_{-L}^{L} |t_n(x) - f(x)|^2 dx,$$

vamos mostrar que $\epsilon_n \leq \hat{\epsilon}_n, \forall n \in \mathbb{N}$.

$$\begin{aligned}
\hat{\epsilon}_n &= \int_{-L}^{L} |t_n(x) - f(x)|^2 dx \\
&= \int_{-L}^{L} | \left(\frac{c_0}{2} + \sum_{k=1}^{n} c_k \cos\frac{k\pi x}{L} + d_k \sin\frac{k\pi x}{L} \right) - f(x)|^2 dx \\
&= \int_{-L}^{L} \left[\left(\frac{c_0}{2}\right)^2 + c_0 \sum_{k=1}^{n} c_k \cos\frac{k\pi x}{L} + d_k \sin\frac{k\pi x}{L} + \right. \\
&\quad + \left(\sum_{k=1}^{n} c_k \cos\frac{k\pi x}{L} + d_k \sin\frac{k\pi x}{L} \right)^2 - \\
&\quad \left. -2f(x) \left(\frac{c_0}{2} + \sum_{k=1}^{n} c_k \cos\frac{k\pi x}{L} + d_k \sin\frac{k\pi x}{L} \right) + |f(x)|^2 \right] dx.
\end{aligned}$$

Usando as relações de ortogonalidade e as expressões dos coeficientes de Fourier, podemos expandir a expressão dada em

$$\hat{\epsilon}_n = \frac{L}{2}c_0^2 - La_0c_0 + L\sum_{k=1}^{n}(c_k^2 + d_k^2) - 2L\sum_{k=1}^{n}(a_kc_k + b_kd_k) + \int_{-L}^{L} |f(x)|^2 dx.$$

Completando quadrados, obteremos a seguinte expressão:

$$\begin{aligned}
\hat{\epsilon}_n &= \frac{L}{2}c_0^2 - La_0c_0 + \frac{L}{2}a_0^2 - \frac{L}{2}a_0^2 + L\sum_{k=1}^{n}(c_k^2 + d_k^2) - 2L\sum_{k=1}^{n}(a_kc_k + b_kd_k) + \\
&\quad + L\sum_{k=1}^{n}(a_k^2 + b_k^2) - L\sum_{k=1}^{n}(a_k^2 + b_k^2) + \int_{-L}^{L} |f(x)|^2 dx.
\end{aligned}$$

E então, obtemos a seguinte expressão para $\hat{\epsilon}_n$:

$$\begin{aligned}\hat{\epsilon}_n &= \frac{L}{2}(c_0 - a_0)^2 - \frac{L}{2}a_o^2 + L\sum_{k=1}^{n}(c_k - a_k)^2 + L\sum_{k=1}^{n}(d_k - b_k)^2 - \\ &\quad -L\sum_{k=1}^{n}(a_k^2 + b_k^2) + \int_{-L}^{L}|f(x)|^2 dx.\end{aligned} \tag{1.2.27}$$

Observando a equação (1.2.27), temos que, se $a_0 = c_0$, $c_k = a_k$ e $d_k = b_k$, com $k = 1, 2, \ldots n$, é o caso em que obteremos o menor valor de $\hat{\epsilon}_n$. Esse caso ainda é o que obtemos a igualdade entre ϵ_n e $\hat{\epsilon}_n$, provando assim que, em geral, $\epsilon_n \leq \hat{\epsilon}_n, \forall n \in \mathbb{N}$.

Visto isso, para obtermos a *Desigualdade de Bessel*, podemos observar que $\hat{\epsilon}_n \geq 0$ quaisquer que sejam os coeficientes c_k e d_k. Logo, tomando um $n \in \mathbb{N}$ qualquer,

$$0 \leq \epsilon_n = \int_{-L}^{L}|f(x)|^2 dx - \frac{L}{2}a_0^2 - L\sum_{k=1}^{n}(a_k^2 + b_k^2)$$

e, então,

$$\frac{L}{2}a_0^2 + L\sum_{k=1}^{n}(a_k^2 + b_k^2) \leq \int_{-L}^{L}|f(x)|^2 dx.$$

Como a escolha do n foi arbitrária (logo, valendo para todo $n \in \mathbb{N}$), temos que a expressão para a **Desigualdade de Bessel** é

$$\frac{L}{2}a_0^2 + L\sum_{k=1}^{\infty}(a_k^2 + b_k^2) \leq \int_{-L}^{L}|f(x)|^2 dx. \tag{1.2.28}$$

Uma outra desigualdade importante para demonstrarmos o Teorema da Convergência Uniforme das Séries de Fourier é a *Desigualdade de Cauchy-Schwarz*.

Proposição 1.2.1. *Seja $n \in \mathbb{N}$ e sejam $a = (a_1, a_2, \ldots, a_n)$ e $b = (b_1, b_2, \ldots, b_n)$ vetores do $\mathbb{R}^n$. A **Desigualdade de Cauchy-Schwarz** para vetores do $\mathbb{R}^n$ tem a seguinte forma:*

$$\left|\sum_{j=1}^{n} a_j b_j\right| \leq \left(\sum_{j=1}^{n} a_j^2\right)^{\frac{1}{2}} \left(\sum_{j=1}^{n} b_j^2\right)^{\frac{1}{2}} \tag{1.2.29}$$

Demonstração. Dado $t \in \mathbb{R}$, considere o somatório

$$\sum_{j=1}^{n}(a_j + tb_j)^2.$$

Abrindo o produto notável temos

$$\sum_{j=1}^{n}(a_j + tb_j)^2 = \sum_{j=1}^{n} a_j^2 + 2t\sum_{j=1}^{n} a_j b_j + t^2\sum_{j=1}^{n} b_j^2.$$

Observando o lado esquerdo da equação acima, temos que a expressão é sempre ≥ 0, para todo $t \in \mathbb{R}$. Olhando o segundo membro, observemos que trata-se de um trinômio do segundo grau em t. Como o lado esquerdo é sempre maior ou igual que zero, temos que o discriminante do trinômio do segundo membro é sempre menor ou igual a zero. Ou seja,

$$\left(2\sum_{j=1}^{n} a_j b_j\right)^2 - 4\left(\sum_{j=1}^{n} a_j^2\right)\left(\sum_{j=1}^{n} b_j^2\right) \leq 0,$$

e portanto

$$\left|\sum_{j=1}^{n} a_j b_j\right| \leq 2\left(\sum_{j=1}^{n} a_j^2\right)^{\frac{1}{2}}\left(\sum_{j=1}^{n} b_j^2\right)^{\frac{1}{2}}.$$

■

Dadas as desigualdades, já temos munição suficiente para demonstrar o teorema sobre a Convergência Uniforme da Série de Fourier.

Teorema 1.2.2 (Teorema sobre a Convergência Uniforme da Série de Fourier)**.** *Seja f uma função periódica de período $2L$, contínua e com $f' \in \mathscr{L}_2$. Então a série de Fourier de f converge uniformemente para f.*

Demonstração. Considere a seguinte expressão

$$\frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos\frac{n\pi x}{L} + b_n \sin\frac{n\pi x}{L},$$

com

$$a_0 = \frac{1}{L}\int_{-L}^{L} f(x)dx,$$

$$a_n = \frac{1}{L}\int_{-L}^{L} f(x)\cos\left(\frac{m\pi x}{L}\right)dx \text{ e}$$

$$b_n = \frac{1}{L}\int_{-L}^{L} f(x)\sin\left(\frac{m\pi x}{L}\right)dx,$$

como sendo a série de Fourier da função f. Para mostrarmos a convergência uniforme da série de Fourier, vamos mostrar que a série

$$\sum_{n=1}^{\infty} a_n \cos\frac{n\pi x}{L} + b_n \sin\frac{n\pi x}{L}$$

converge uniformemente. Como

$$\left|a_n \cos\frac{n\pi x}{L}\right| \leq |a_n| \text{ e } \left|b_n \sin\frac{n\pi x}{L}\right| \leq |b_n|,$$

vamos verificar em quais condições se dá a convergência da série

$$\sum_{k=1}^{\infty}(|a_k| + |b_k|). \tag{1.2.30}$$

Como f é contínua, $f' \in \mathscr{L}_2$ e considerando

$$a'_n = \int_{-L}^{L} f'(x) \cos \frac{n\pi x}{L} \ e$$

$$b'_n = \int_{-L}^{L} f'(x) \sin \frac{n\pi x}{L}$$

os coeficientes de Fourier de f', temos, pelas equações dadas em (1.2.22) e (1.2.24), obtemos

$$a_n = \frac{-L}{\pi n} b'_n, \ b_n = \frac{L}{\pi n} a'_n.$$

Portanto, a reduzida de ordem n da série (1.2.30) é

$$\sum_{k=1}^{n} (|a_k| + |b_k|) = \frac{L}{\pi} \sum_{k=1}^{n} \frac{1}{k} (|a'_k| + |b'_k|)$$

Pela Desigualdade de Cauchy-Schwarz, temos que

$$\sum_{k=1}^{n} \frac{1}{k} (|a'_k| + |b'_k|) \leq \left(\sum_{k=1}^{n} \frac{1}{k^2} \right)^{\frac{1}{2}} \left(\sum_{k=1}^{n} (|a'_k| + |b'_k|)^2 \right)^{\frac{1}{2}}.$$

Assim, obtemos uma majoração para reduzida da série (1.2.30), dada por

$$\sum_{k=1}^{n} (|a_k| + |b_k|) \leq \frac{L}{\pi} \left(\sum_{k=1}^{n} \frac{1}{k^2} \right)^{\frac{1}{2}} \left(\sum_{k=1}^{n} (|a'_k| + |b'_k|)^2 \right)^{\frac{1}{2}} \tag{1.2.31}$$

Como $(|a| + |b|)^2 \leq 2(a^2 + b^2)$, obtemos da expressão acima a seguinte majoração para a reduzida da série (1.2.30):

$$\sum_{k=1}^{n} (|a_k| + |b_k|) \leq \frac{\sqrt{2}L}{\pi} \left(\sum_{k=1}^{n} \frac{1}{k^2} \right)^{\frac{1}{2}} \left(\sum_{k=1}^{n} (|a'_k|^2 + |b'_k|^2) \right)^{\frac{1}{2}}. \tag{1.2.32}$$

Logo, quando $n \longrightarrow \infty$, temos que a série numérica $\displaystyle\sum_{k=1}^{\infty} \frac{1}{k^2}$ converge (LIMA, 2004) e, pela Desigualdade de Bessel,

$$\sum_{k=1}^{\infty} (|a'_k|^2 + |b'_k|^2) \leq \int_{-L}^{L} |f'(x)| dx < \infty.$$

Portanto, pelo Teste M de Weierstrass, dado em 1.1.6, a Série de Fourier converge uniformemente para f. ■

1.2.4 Forma Complexa da Série de Fourier

As séries de Fourier podem ser expressas na forma complexa. Para construirmos, vamos considerar a fórmula de Euler (FIGUEIREDO, 2014),

$$e^{i\theta} = \cos\theta + i\sin\theta$$
$$e^{-i\theta} = \cos\theta - i\sin\theta.$$

Se fizermos $e^{i\theta} + e^{-i\theta}$ obtemos

$$e^{i\theta} + e^{-i\theta} = \cos\theta + i\sin\theta + \cos\theta - i\sin\theta = 2\cos\theta,$$

obtendo a seguinte expressão

$$\cos\theta = \frac{e^{-i\theta} + e^{-i\theta}}{2}. \tag{1.2.33}$$

De forma análoga, se fizermos $e^{i\theta} - e^{-i\theta}$ teremos

$$e^{i\theta} - e^{-i\theta} = \cos\theta + i\sin\theta - \cos\theta + i\sin\theta = 2i\sin\theta,$$

obtendo a seguinte expressão

$$\sin\theta = \frac{e^{-i\theta} - e^{-i\theta}}{2i}. \tag{1.2.34}$$

Com as expressões de seno e cosseno na sua forma complexa, podemos escrever os coeficientes de Fourier usando tais expressões, da seguinte forma:

$$a_n \cos\frac{n\pi x}{L} + b_n \sin\frac{n\pi x}{L} = \left(\frac{a_n}{2} + \frac{b_n}{2i}\right) e^{in\pi x/L} + \left(\frac{a_n}{2} - \frac{b_n}{2i}\right) e^{-in\pi x/L},$$

Logo, o coeficiente c_n para $e^{in\pi x/L}$ é dado por

$$c_n = \frac{a_n}{2} + \frac{b_n}{2i} = \frac{1}{2}(a_n - ib_n) = \frac{1}{2L}\int_{-L}^{L} f(x)\left(\cos\frac{n\pi x}{L} - i\sin\frac{n\pi x}{L}\right)dx,$$

ou seja

$$c_n = \frac{1}{2L}\int_{-L}^{L} f(x)e^{-in\pi x/L}dx. \tag{1.2.35}$$

Definimos também

$$c_0 = \frac{a_0}{2} = \frac{1}{2L}\int_{-L}^{L} f(x)dx. \tag{1.2.36}$$

Logo, podemos definir que, se $f : \mathbb{R} \longrightarrow \mathbb{R}$ for uma função periódica, com período 2L e $f \in \mathscr{L}_1$, podemos dizer que a série de Fourier de f poderá ser escrita da forma

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{in\pi x/L} \tag{1.2.37}$$

com

$$c_n = \frac{1}{2L} \int_{-L}^{L} f(x) e^{-in\pi x/L} dx, \ para \ n = 0, \pm 1, \pm 2 \ldots$$

2 Reconstrução: Conceitos principais e Admissibilidade

Nessa seção, introduziremos algumas notações, definições e alguns teoremas importantes para a fundamentação teórica da nossa aplicação. Estes são necessários para deixar claro que tudo que faremos em seguida tem validade. Nos dará também o suporte necessário para poder explicar sobre quais condições o programa que desenvolvemos no MATLAB está operando.

2.1 Transformada de Fourier

Para construirmos os conceitos da transformada de Fourier, vamos começar com um exemplo bem importante. Consideremos uma função "Retângulo", isto é, a função $\gamma : \mathbb{R} \longrightarrow \mathbb{R}$ definida por

$$\gamma(t) = \begin{cases} 1, & |t| < \frac{1}{2} \\ 0, & |t| \geq \frac{1}{2} \end{cases},$$

cujo gráfico é como mostramos a seguir:

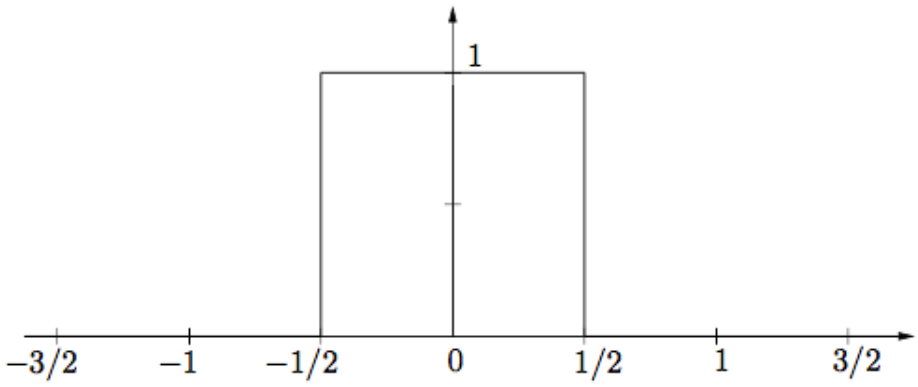


Figura 2.1.1 – Gráfico da função γ, retirado das notas de aula (OSGOOD, 2009).

Como podemos perceber, a função γ não é periódica, portanto, não tem sua série de Fourier bem definida. Vamos então criar uma versão periódica da função para nos auxiliar na construção da transformada de Fourier. Para torná-la periódica, vamos criar cópias da sua parte não nula e vamos separá-las de forma igualmente espaçada. Apresentaremos um exemplo da versão periódica de γ, com período 15, na figura 2.1:

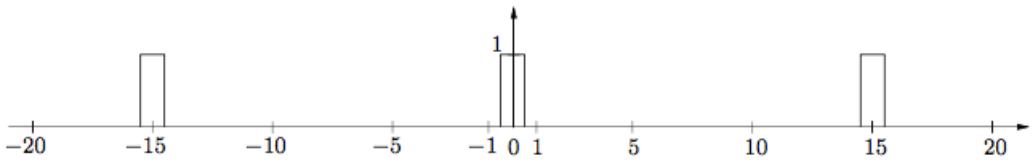


Figura 2.1.2 – Gráfico da versão periódica da função γ, retirado das notas de aula (OSGOOD, 2009).

Como γ é periódica de período, digamos T, podemos calcular sua série de Fourier na expressão

$$\gamma(t) = \frac{1}{T}\sum_{-\infty}^{\infty} c_n e^{2\pi i n t/T} dt$$

e calcularemos seu n-ésimo coeficiente c_n, que nos dá o seguinte resultado:

$$\begin{aligned} c_n &= \frac{1}{T}\int_{-T/2}^{T/2} \gamma(t) e^{-2\pi n i t/T} dt \\ &= \frac{1}{T}\int_{-1/2}^{1/2} 1 \cdot e^{-2\pi n i t/T} dt \\ &= \frac{1}{T}\left[\frac{T}{-2\pi i n} e^{-2\pi i n t/T}\right]_{t=-1/2}^{t=1/2} \\ &= \frac{1}{2\pi i n}\left(e^{\pi i n/T} - e^{-\pi i n/T}\right) \\ &= \frac{1}{\pi n}\sin\left(\frac{\pi n}{T}\right). \end{aligned}$$

Podemos chamar o valor de c_n da "transformada da versão periódica de γ", nos pontos $\frac{n}{T}$, com n variando de $\pm 1, \pm 2, \cdots$. Definiremos como Υ a "transformada da versão periódica de γ" e como Υ' a "Transformada da versão escalonada e periódica de γ". Então, escrevemos da seguinte forma:

$$\Upsilon(\frac{n}{T}) = \frac{1}{\pi n}\sin\left(\frac{\pi n}{T}\right).$$

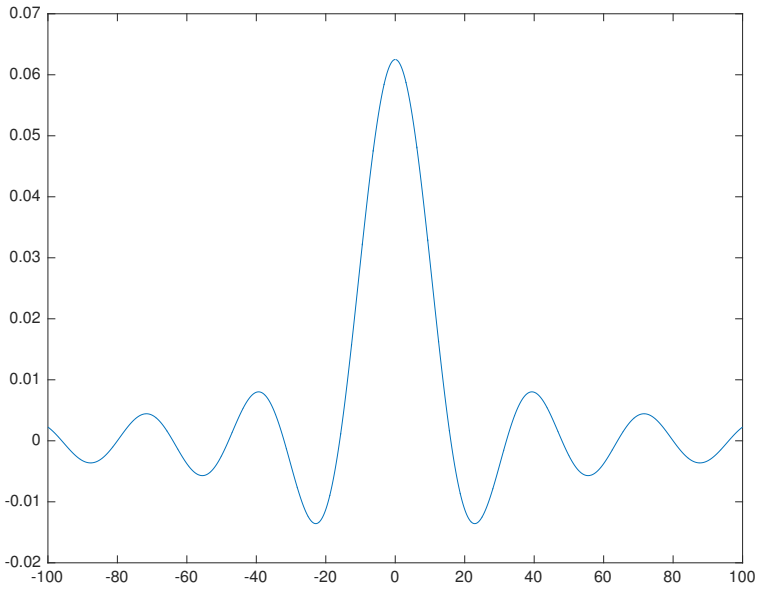
Como estamos trabalhando nos pontos $w = \frac{n}{T}$, podemos multiplicar a transformada mostrada acima para obtermos a versão escalonada em T da transformada, isto é,

$$\Upsilon'(\frac{n}{T}) = T\frac{1}{\pi n}\sin\left(\frac{\pi n}{T}\right).$$

E como $w = \frac{n}{T}$, escrevemos

$$\Upsilon'(w) = \frac{1}{\pi w}\sin(w).$$

Para T = 16, temos o seguinte gráfico para a Transformada de γ:



Recapitulando, construímos a transformada da versão periódica da função γ usando seu coeficiente de Fourier, obtendo:

$$\begin{aligned}\Upsilon(\frac{n}{T}) &= T \cdot c_n \\ &= T \cdot \frac{1}{T}\int_{-T/2}^{T/2} \gamma(t)e^{-2\pi int/T}dt \\ &= \int_{-T/2}^{T/2} \gamma(t)e^{-2\pi int/T}dt.\end{aligned}$$

Quanto $T \to \infty$, os limites de integração passam a ser $\pm\infty$ e substituindo a variável discreta $\frac{n}{T}$, pela variável contínua w ($w = \frac{n}{T}$), obtemos:

$$\hat{\gamma}(w) = \int_{-\infty}^{\infty} \gamma(t)e^{-2\pi iwt}dt$$

Definição 2.1.1. *O Espaço de Schwartz ou espaço das funções rapidamente decrescentes, que denotamos por* $\mathbb{S}$*, é o subespaço vetorial formado pelas funções* $\zeta \in C^\infty(\mathbb{R}^n)$ *tais que*

$$\lim_{|x|\to\infty} \| x \|^k \frac{\partial^\alpha}{\partial x}\zeta(x) = 0,$$

quaisquer que sejam $k \in \mathbb{N}$ *e* $\alpha \in \mathbb{N}^n$.

Estendendo o que construímos acima para qualquer função f periódica, podemos definir as Transformadas de Fourier da seguinte maneira:

Definição 2.1.2. *A **Transformada de Fourier** de uma função $f(t) \in \mathbb{S}$ é dada por;*

$$\hat{f}(w) = \int_{-\infty}^{+\infty} e^{-2\pi iwt} f(t)dt \tag{2.1.1}$$

O domínio da transformada de Fourier é um conjunto de números reais w. Dizemos que $\hat{f}$ está definido no Domínio das Frequências e o sinal original $f(t)$ está definido no Domínio do Tempo.

No caso do exemplo que construímos acima, podemos chamar a Transformada de Fourier da função γ de

$$\hat{\gamma}(w) \quad ou \quad \mathscr{F}\gamma(w).$$

Proposição 2.1.1. *A **Transformada Inversa de Fourier** de uma função $f \in \mathbb{S}$ é dada por*

$$f(x) = \int_{-\infty}^{+\infty} \hat{f}(w) e^{2\pi ixw} dw \tag{2.1.2}$$

Demonstração. Consideremos a expansão da versão periódica de f pela série de Fourier, dada por

$$f(x) = \sum_{n=-\infty}^{+\infty} c_n e^{2\pi iwx}$$

com $w = \dfrac{n}{T}$, T o período da função e

$$c_n = \frac{1}{T} \int_{-\infty}^{+\infty} e^{-2\pi iwt} f(t)dt$$

Pela definição da Transformada de Fourier,

$$\hat{f}(w) = \int_{-\infty}^{+\infty} e^{-2\pi iwt} f(t)dt$$

Logo, temos que

$$c_n = \frac{1}{T} \hat{f}(w)$$

Segue que

$$f(x) = \sum_{n=-\infty}^{+\infty} c_n e^{2\pi iwx} = \sum_{n=-\infty}^{+\infty} \frac{1}{T} \hat{f}(w) e^{2\pi iwx}$$

Seja $\dfrac{1}{T} = \triangle w$. Temos que:

$$\begin{aligned} f(x) &= \sum_{-\infty}^{+\infty} \frac{1}{T} \hat{f}(w) e^{2\pi iwx} \\ &= \sum_{-\infty}^{+\infty} \hat{f}(w) e^{2\pi iwx} \triangle w. \end{aligned}$$

Fazendo $\triangle w \longrightarrow \infty$, o somatório $\sum_{-\infty}^{+\infty} \hat{f}(w)e^{2\pi iwx}\triangle w$ converge para a integral, pela soma de Riemann:

$$\begin{aligned} f(x) &= \sum_{-\infty}^{+\infty} \hat{f}(w)e^{2\pi iwx}\triangle w \\ &\simeq \int_{-\infty}^{+\infty} \hat{f}(w)e^{2\pi ixw}dw. \end{aligned}$$

Segue, portanto, que a transformada inversa de Fourier é dada por

$$f(x) = \int_{-\infty}^{+\infty} \hat{f}(w)e^{2\pi ixw}dw.$$

■

Claro que, ao manipularmos as funções no MATLAB, precisaremos de conceitos para funções discretizadas. Além disso, sinais de áudio já vêm discretizados. Então, sempre que necessário, traremos aqui algumas definições análogas para funções discretas.

Definição 2.1.3. *A sequência $q : \mathbb{Z} \to \mathbb{C}$ pertence a ℓ_2 se*

$$\sum_{k\in\mathbb{Z}} |q_k|^2 < \infty \tag{2.1.3}$$

Definição 2.1.4. *A **Transformada de Fourier** em Discrete-Time (DTFT) da sequência q é definida por*

$$\hat{q}(w) = \sum_{k\in\mathbb{Z}} q_k e^{-2\pi iwt} \tag{2.1.4}$$

Precisaremos, em algum momento do processo de reconstrução, trabalhar com uma amostragem da função f, com um espaçamento específico T. Então, definiremos a *Amostragem em Grade* da função f como

$$[f]_T := [\ldots, f(-2T), f(-T), f(0), f(T), f(2T), \ldots]$$

Definição 2.1.5. *O flip de uma função f e de um filtro digital $q = [\ldots, q_{-1}, q_0, q_1, \ldots]$ é dado por*

$$f^\vee(x) = f(-x) \text{ e } (q^\vee)_{-i} = q_i, \ i \in \mathbb{Z}$$

Três propriedades das Transformadas de Fourier são fundamentais para o desenvolver da nossa construção. São elas:

Proposição 2.1.2 (Linearidade da Transformada de Fourier)**.** *Seja $f : \mathbb{R} \longrightarrow \mathbb{C}$ e $g : \mathbb{R} \longrightarrow \mathbb{C}$ funções, com $f, g \in \mathbb{S}$ e seja α um escalar real ou complexo qualquer. Então, valem as seguintes propriedades:*

$$\mathscr{F}(f+g)(w) = \mathscr{F}f(w) + \mathscr{F}g(w)$$

$$\mathscr{F}(\alpha f)(w) = \alpha \mathscr{F}f(w)$$

Demonstração.

$$\begin{aligned}
\mathscr{F}(f+g)(w) &= \int_{-\infty}^{\infty} (f(t)+g(t)))e^{-2\pi iwt}dt \\
&= \int_{-\infty}^{\infty} f(t)e^{-2\pi iwt}dt + \int_{-\infty}^{\infty} g(t)e^{-2\pi iwt}dt \\
&= \mathscr{F}f(w) + \mathscr{F}g(w).
\end{aligned}$$

$$\begin{aligned}
\mathscr{F}(\alpha f)(w) &= \int_{-\infty}^{\infty} \alpha f(t)e^{-2\pi iwt}dt \\
&= \alpha \int_{-\infty}^{\infty} f(t)e^{-2\pi iwt}dt \\
&= \alpha \mathscr{F}f(w).
\end{aligned}$$

■

Proposição 2.1.3. *Seja $f : \mathbb{R} \longrightarrow \mathbb{C}$ uma função, com $f \in \mathbb{S}$ e seja α um escalar real qualquer. Vale o seguinte:*

$$\mathscr{F}(f(t+\alpha))(w) = e^{2\pi iw\alpha}\mathscr{F}f(w)$$

Demonstração.

$$\begin{aligned}
\mathscr{F}(f(t+\alpha))(w) &= \int_{-\infty}^{\infty} f(t+\alpha)e^{-2\pi iwt}dt \\
&= \int_{-\infty}^{\infty} f(u)e^{-2\pi iw(u-\alpha)}du \\
&= \int_{-\infty}^{\infty} f(u)e^{-2\pi iwu}e^{2\pi iw\alpha}du \\
&= e^{2\pi iw\alpha}\int_{-\infty}^{\infty} f(u)e^{-2\pi iwu}du \\
&= e^{2\pi iw\alpha}\mathscr{F}f(w).
\end{aligned}$$

■

Proposição 2.1.4. *Seja $f : \mathbb{R} \longrightarrow \mathbb{C}$ uma função com $f \in \mathbb{S}$ e seja α um escalar real qualquer diferente de zero. Vale o seguinte:*

$$\mathscr{F}(f(\alpha t))(w) = \frac{1}{\alpha}\mathscr{F}f(\frac{w}{\alpha})$$

Demonstração.

$$\begin{aligned}\mathscr{F}(f(\alpha t))(w) &= \int_{-\infty}^{\infty} f(\alpha t)e^{-2\pi iwt}dt \\ &= \int_{-\infty}^{\infty} f(u)e^{-2\pi iw\frac{u}{\alpha}}\frac{1}{\alpha}du \\ &= \frac{1}{\alpha}\int_{-\infty}^{\infty} f(u)e^{-2\pi i\frac{w}{\alpha}u}du \\ &= \frac{1}{\alpha}\mathscr{F}f(\frac{w}{\alpha}).\end{aligned}$$

■

2.2 Convolução

Em nosso trabalho, usaremos filtros digitais para modificar um sinal de áudio. Mas, falando de maneira geral, como podemos usar um sinal para modificar outro? Já falamos da propriedade da Linearidade das Transformadas de Fourier que, de certa forma, são uma aplicação do que queremos. Mas, o que acontece se, dados dois sinais quaisquer $f(t)$ e $g(t)$, o que acontece se fizermos o produto das suas transformadas, isto é,

$$\mathscr{F}f(w)\mathscr{F}g(w)?$$

O produto das transformadas de Fourier de $f(t)$ e $g(t)$ nos fornece:

$$\mathscr{F}f(w)\mathscr{F}g(w) = \int_{-\infty}^{\infty} e^{-2\pi iwt}g(t)dt\int_{-\infty}^{\infty} e^{-2\pi iwx}f(x)dx$$

Usaremos diferentes variáveis t e x para combinarmos o produto em uma integral dupla.

$$\begin{aligned}\int_{-\infty}^{\infty} e^{-2\pi iwt}g(t)dt\int_{-\infty}^{\infty} e^{-2\pi iwx}f(x)dx &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-2\pi iwt}e^{-2\pi iwx}g(t)f(x)dtdx \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-2\pi iw(t+x)}g(t)f(x)dtdx \\ &= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} e^{-2\pi iw(t+x)}g(t)dt\right)f(x)dx\end{aligned}$$

Fazendo a mudança de variável $u = t + x$, teremos $t = u - x$, $du = dt$ e os limites permanecem os mesmos. Então,

$$\int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} e^{-2\pi iw(t+x)}g(t)dt\right)f(x)dx = \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} e^{-2\pi iwu}g(u-x)du\right)f(x)dx.$$

Trocando a ordem de integração (Teorema de Fubini), obtemos:

$$\begin{aligned}\int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty}e^{-2\pi iwu}g(u-x)du\right)f(x)dx &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}e^{-2\pi iwu}g(u-x)f(x)dudx\\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}e^{-2\pi iwu}g(u-x)f(x)dxdu\\ &= \int_{-\infty}^{\infty}e^{-2\pi iwu}\left(\int_{-\infty}^{\infty}g(u-x)f(x)dx\right)du\end{aligned}$$

Se denominarmos a integral interna determinada acima de $h(u)$, teremos

$$h(u)=\int_{-\infty}^{\infty}g(u-x)f(x)dx$$

produzindo

$$\int_{-\infty}^{\infty}e^{-2\pi iwu}\left(\int_{-\infty}^{\infty}g(u-x)f(x)dx\right)du=\int_{-\infty}^{\infty}e^{-2\pi iwu}h(u)du=\mathscr{F}h(w),$$

que é a transformada de Fourier da função h.

Com isso, podemos definir a Convolução entre os dois sinais $f(t)$ e $g(t)$. Em analogia com a convolução contínua, definiremos também convolução discreta e mista.

Definição 2.2.1. *Dadas funções $f,g\in\mathbb{S}$, sequências $c,q\in\ell_2$ e $T\in\mathbb{R}$. As convoluções contínua, discreta e mista são dadas por:*

Contínua

$$(f*g)(x)=\int_{-\infty}^{\infty}f(t)g(x-t)dt \tag{2.2.1}$$

Discreta

$$(c*q)_n=\sum_{k\in\mathbb{Z}}c_kq_{n-k} \tag{2.2.2}$$

Mista

$$(q*_T f)(x)=\sum_{k\in\mathbb{Z}}q_kf(x-kT) \tag{2.2.3}$$

Podemos também enunciar o *Teorema da Convolução*, cuja demonstração é a construção acima.

Teorema 2.2.1 (Teorema da Convolução)**.** *Sejam $f, g \in \mathbb{S}$ duas funções e $f * g$ sua convolução. Seja $\mathscr{F}$ o operador Transformada de Fourier, tal que $\mathscr{F}f$ e $\mathscr{F}g$ são as Transformadas de Fourier de f e g respectivamente. Então*

$$\mathscr{F}(f * g) = \mathscr{F}f \cdot \mathscr{F}g.$$

Demonstração. Vide construção no início desta seção. ■

Três propriedades da Convolução são importantes de serem citadas: a comutatividade da convolução e a distributividade.

Proposição 2.2.1 (Comutativa)**.** *Sejam $f, g \in \mathbb{S}$ duas funções e $f * g$ sua convolução. Então,*

$$f * g = g * f.$$

Demonstração.

$$\begin{aligned}(f * g)(t) &= \int_{-\infty}^{\infty} f(t)g(x-t)dt \\ &= \int_{-\infty}^{\infty} f(x-u)g(u)du \text{ (fazendo mudança de variável } x - t = u) \\ &= (g * f)(t).\end{aligned}$$

■

Proposição 2.2.2 (Associativa)**.** *Sejam f, g e h funções com $f, g, h \in \mathbb{S}$. Então,*

$$f * (g * h) = (f * g) * h.$$

Demonstração. Usaremos o Teorema da Convolução:

$$\begin{aligned}\mathscr{F}[f * (g * h)](w) &= \mathscr{F}f(w) \cdot [\mathscr{F}(g * h)(w)] \\ &= \mathscr{F}f \cdot (\mathscr{F}g \cdot \mathscr{F}h) \\ &= (\mathscr{F}f \cdot \mathscr{F}g) \cdot \mathscr{F}h \\ &= \mathscr{F}[(f * g) * h](w)\end{aligned}$$

Aplicando a Transformada Inversa em ambos os lados da igualdade e, como $\mathscr{F}^{-1}\mathscr{F}f = f$, temos que

$$f * (g * h)(t) = (f * g) * h(t).$$

■

Proposição 2.2.3 (Distributiva)**.** *Sejam f, g e h funções com $f, g, h \in \mathbb{S}$. Então,*

$$f * (g + h) = f * g + f * h.$$

Demonstração.

$$\begin{aligned}[f * (g+h)](t) &= \int_{-\infty}^{\infty} f(t)[g+h](x-t)dt \\ &= \int_{-\infty}^{\infty} f(t)g(x-t) + \int_{-\infty}^{\infty} f(t)h(x-t)dt \\ &= [f*g + f*h](t)\end{aligned}$$

■

2.3 Estágios de Reconstrução

Vistos os conceitos acima destacados, podemos construir a seguinte abordagem para reconstrução e amostragem de um sinal.

A figura 2.3.1 indica a abordagem de amostragem e reconstrução. De modo geral, uma aproximação $\tilde{f}_T$ de f é obtida seguindo alguns estágios: uma pré-filtragem usando um pré-filtro de análise, para eliminar, em f, frequências acima de $\frac{1}{2T}$, seguido de uma filtragem; aplicando o filtro digital q por convolução discreta; e então reconstruindo $\tilde{f}_T$ a partir de uma função geradora φ.

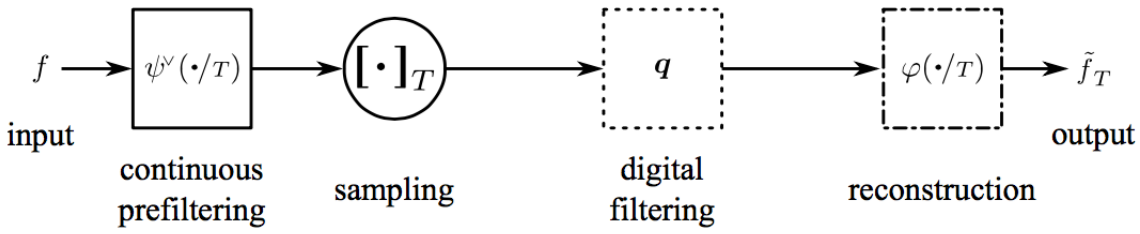
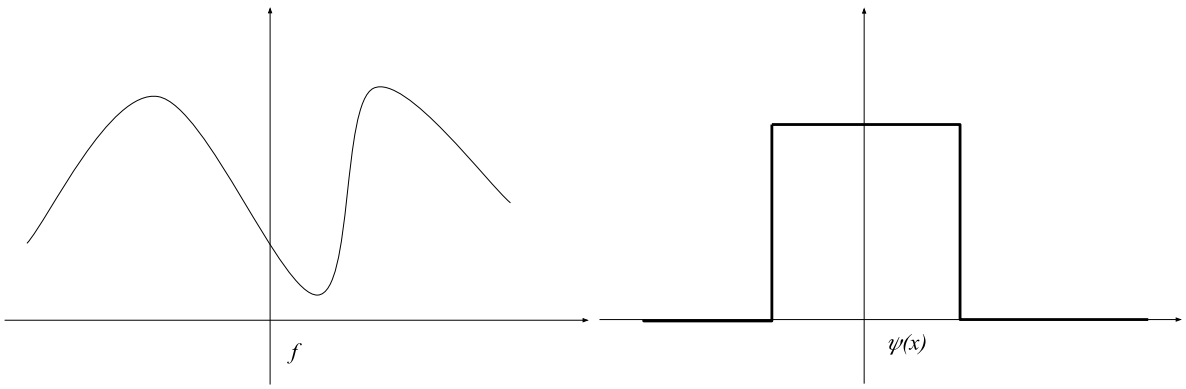


Figura 2.3.1 – Pipeline de reconstrução moderno, retirado de (SACHT, 2014)

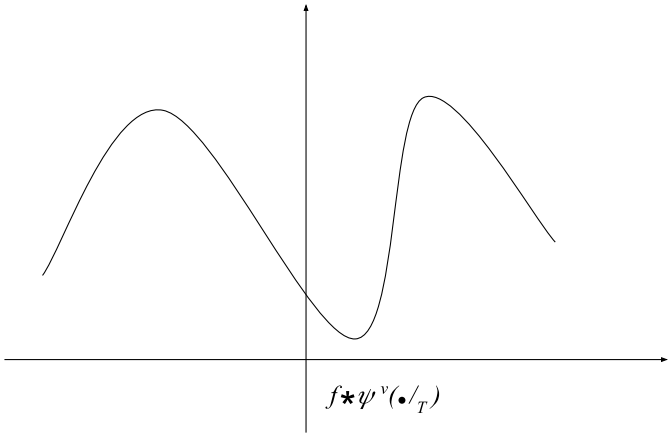
Considere uma função $f \in \mathscr{L}^2$ integrável qualquer e um filtro $\psi^\vee$ de análise (pré-filtro) qualquer.



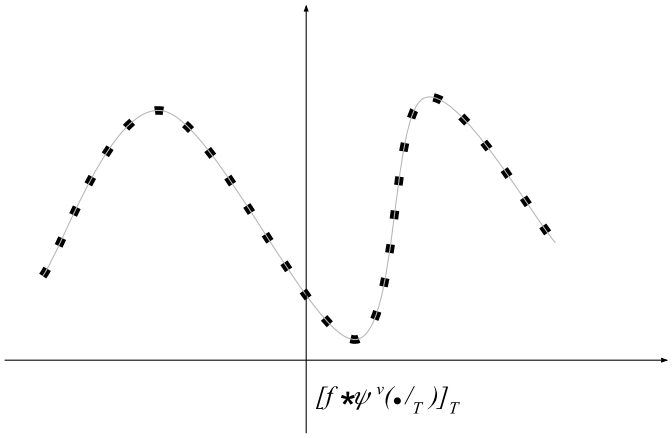
Operamos f e $\psi^{\vee}(\cdot/_T)$ usando uma convolução contínua

$$(f * \psi(\cdot/_T)) = \int_{-\infty}^{\infty} f(x)\psi(x - Tt)dt$$

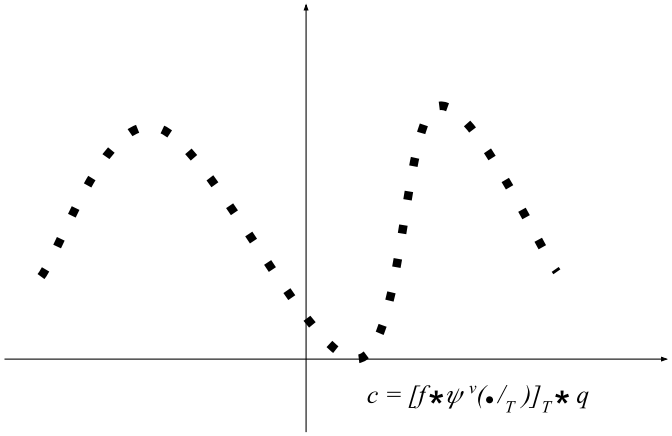
de modo a construirmos outra função com densidade menor de pontos no domínio (resolução menor).



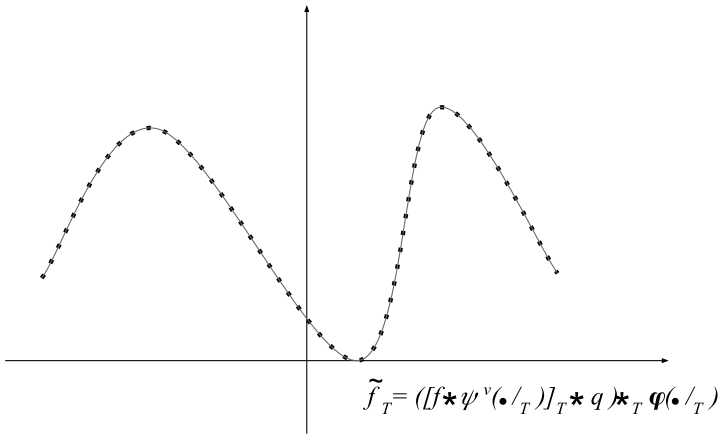
Em seguida, obteremos uma amostragem discreta com espaçamento T, obtendo



Aplicaremos então o filtro digital $q \in l_2$, aplicando convolução discreta com $[f * \psi(\cdot/_T)]$



e, em seguida, criando o sinal $\tilde{f}_T$, aplicando à sequência c uma convolução com uma função geradora $\varphi(\cdot/_T)$



Definição 2.3.1. *A inversa de q, quando existir, é outra sequência denotada* q^{-1} *tal que*

$$q * q^{-1} = \delta = [\ldots, 0, 1, 0, 0, \ldots]$$

Definição 2.3.2. *Um filtro digital q é dito* ***FIR*** *(Finite Impulse Response) se tem um número finito de elementos diferentes de zero. Um filtro digital q é dito* ***IFIR*** *(Inverse of Finite Impulse Response) quando é a inversa de um filtro FIR. Além disso, um filtro digital q é* ***FIR-IFIR*** *se pode ser escrito como* $q = q_1 * q_2$*, em que* q_1 *é FIR e* q_2 *é IFIR.*

O subespaço Vetorial das funções geradas por cópias de um gerador φ, com amostragem em grade T, é dado por

$$V_{\varphi,T} = \{\tilde{f} = \sum_{k\in\mathbb{Z}} c_k \cdot \varphi(x - kT) \ \mid \ \forall c \in \ell_2\},$$

pois, dado um gerador φ, a convolução iterada dele com ele mesmo gera uma família de geradores com mesma amostragem, como mostra o exemplo abaixo.

Exemplo 2.3.1. *O subespaço B-spline é a família de geradores mais comumente usada, que podem ser definidos como*

$$\beta^0 = \begin{cases} 1, & se \ \mid x \mid < 0.5 \\ 0.5, & se \ \mid x \mid = 0.5 \\ 0, & se \ \mid x \mid > 0.5 \end{cases} \quad e \ \beta^n = \beta^{n-1} * \beta^0 \tag{2.3.1}$$

A Transformada de Fourier correspondente é

$$\hat{\beta}^n = sinc^{n+1}(w) = \left(\frac{\sin \pi w}{\pi w}\right)^{n+1}, \ n \geq 0 \tag{2.3.2}$$

2.4 Admissibilidade

Com esses conceitos definidos e traçados, de forma geral, a sequência de passos para a reconstrução de um sinal, vamos discutir na próxima sessão as hipóteses da teoria, ou seja, quais resultados necessários para tornar nosso processo de reconstrução válido e para que nossa função gerada $\tilde{f}$ esteja bem definida.

Definição 2.4.1. *Dado um $r \in \mathbb{R}, r > 0$, definimos o espaço de Sobolev W_2^r como o conjunto das funções f que satisfazem*

$$\int_{-\infty}^{\infty} (1+w^2)^r \mid \hat{f}(w) \mid^2 dw < \infty.$$

Tomando a função f de entrada em nossa reconstrução, assumiremos que

$$f \in W_2^r \text{ para algum } r > \frac{1}{2}$$

Alguns exemplos de funções que pertentem ao Espaço de Sobolev são:

- Funções $f \in \mathscr{L}_2$ de banda limitada (ou seja, $\exists M > 0$ tal que $\hat{f}(w) = 0$, $\forall w \in \mathbb{R} \setminus [-M, M]$)
- Funções f tais que $\hat{f}(w)$ tem decaimento exponencial.

Teorema 2.4.1. *Seja $T > 0$. Seja $f \in W_2^r$, para algum $r > \frac{1}{2}$ e seja ψ um filtro digital tal que $\exists k \in \mathbb{R}$ em que $\| \hat{\psi} \|_\infty \leq k < \infty$. Então $[f * \hat{\psi}^\vee(\cdot/_T)]_T \in \ell_2$*

Demonstração. (BLU; UNSER, 1999) ■

Teorema 2.4.2 (Teorema de Plancherel-Parseval)**.** *Dadas $f, g \in \mathbb{S}$, tem-se*

$$\int_{-\infty}^{\infty} f(x)\overline{g(x)}dx = \int_{-\infty}^{\infty} \hat{f}(w)\overline{\hat{g}(w)}dw \tag{2.4.1}$$

Demonstração.

$$\begin{aligned}\int_{-\infty}^{\infty} \hat{f}(w)\hat{g}(w)dw &= \int_{-\infty}^{\infty} \hat{f}(w) \int_{-\infty}^{\infty} e^{2\pi i w x}\overline{g(x)}dxdw \\ &= \int_{-\infty}^{\infty} \overline{g(x)} \int_{-\infty}^{\infty} e^{2\pi i w x}\hat{f}(w)dwdx.\end{aligned}$$

Pela Inversa da Transformada de Fourier, temos que

$$\begin{aligned}\int_{-\infty}^{\infty} \hat{f}(w)\hat{g}(w)dw &= \int_{-\infty}^{\infty} \overline{g(x)} \int_{-\infty}^{\infty} e^{2\pi iwx} \hat{f}(w)dwdx \\ &= \int_{-\infty}^{\infty} \overline{g(x)} f(x)dx.\end{aligned}$$

■

No processo de reconstrução, usaremos o filtro digital **q** seja FIR, IFIR ou FIR-IFIR. Filtros $q = [\beta^n]^{-1}$ ou $q = [a_{\beta^n}]^{-1}$ são exemplos de IFIR filtros. A ideia do próximo resultado é mostrar que, após aplicarmos um filtro digital **q** IFIR, FIR ou FIR-IFIR, em um $c \in \ell_2$, a convolução $c * q$ ainda permanecerá em ℓ_2.

Teorema 2.4.3. *Seja $c \in \ell_2$ e seja q um filtro digital FIR, IFIR ou FIR-IFIR. Então $c * q \in \ell_2$.*

Demonstração. (SACHT, 2014) Primeiramente iremos demonstrar o caso em que q é FIR. Para mostrarmos que $c * q \in \ell_2$, basta mostrarmos que $\sum_{k\in\mathbb{Z}} \mid c * q \mid_k^2 < \infty$.

Pelo Teorema de Plancherel-Parseval e o Teorema da Convolução, temos que

$$\begin{aligned}\sum_{k\in\mathbb{Z}} \mid c * q \mid_k^2 &= \parallel \hat{c}(w) \cdot \hat{q}(w) \parallel_{L_2} \\ &= \int_{-\frac{1}{2}}^{\frac{1}{2}} \mid \hat{c}(w) \cdot \hat{q}(w) \mid^2 dw \\ &= \int_{-\frac{1}{2}}^{\frac{1}{2}} \mid \hat{c}(w) \mid^2 \sum_{k\in\mathbb{Z}} \mid q_k e^{-2k\pi iw} \mid^2 dw\end{aligned} \tag{2.4.2}$$

Como q é FIR, existem finitos pontos q_k que são diferentes de zero. Logo $\sum_{k\in\mathbb{Z}} q_k e^{-2k\pi iw}$ é uma soma finita de funções contínuas. Seja

$$M = \max_{w\in\mathbb{R}} \left| \sum_{k\in\mathbb{Z}} \mid q_k e^{-2k\pi iw} \mid^2 \right|^2 \tag{2.4.3}$$

Da equação 2.4.2, temos

$$\begin{aligned}\sum_{k\in\mathbb{Z}} \mid c * q \mid_k^2 &= \int_{-\frac{1}{2}}^{\frac{1}{2}} \mid \hat{c}(w) \mid^2 \sum_{k\in\mathbb{Z}} \mid q_k e^{-2k\pi iw} \mid^2 dw \\ &\leq M \int_{-\frac{1}{2}}^{\frac{1}{2}} \mid \hat{c}(w) \mid^2 dw \\ &= M \sum_{k\in\mathbb{Z}} \mid c_k \mid^2 < \infty\end{aligned} \tag{2.4.4}$$

em que usamos o Teorema de Plancherel-Parseval na última igualdade de 2.4.4, provando assim que $c * q \in \ell_2$ se q é FIR.

Se q é IFIR, sua Transformada de Fourier é dada por

$$\hat{q}(w) = \frac{1}{\sum q_k e^{-2\pi i w k}}. \tag{2.4.5}$$

Como q é IFIR, temos que o denominador de 2.4.5 não pode ser zero. Então, tomando

$$M = \max_{w \in \mathbb{R}} \left| \frac{1}{\sum q_k e^{-2\pi i w k}} \right|^2 \tag{2.4.6}$$

temos

$$\begin{aligned} \sum_{k \in \mathbb{Z}} \mid c * q \mid_k^2 &= \parallel \hat{c}(w) \cdot \hat{q}(w) \parallel_{L_2} \\ &= \int_{-\frac{1}{2}}^{\frac{1}{2}} \mid \hat{c}(w) \cdot \hat{q}(w) \mid^2 dw \\ &= \int_{-\frac{1}{2}}^{\frac{1}{2}} \mid \hat{c}(w) \mid^2 \left| \frac{1}{\sum q_k e^{-2\pi i w k}} \right|^2 dw \\ &\leq M \int_{-\frac{1}{2}}^{\frac{1}{2}} \mid \hat{c}(w) \mid^2 dw \\ &= M \sum_{k \in \mathbb{Z}} \mid c_k \mid^2 < \infty \end{aligned} \tag{2.4.7}$$

provando, assim, que $c * q \in \ell_2$ para q IFIR.

No caso em que q é FIR-IFIR, temos que q pode ser escrito como $q = q_1 * q_2$ em que q_1 é FIR e q_2 é IFIR. Então, temos que

$$c * q = c * (q_1 * q_2) = (c * q_1) * q_2$$

e como q_1 é FIR, pelo primeiro caso, temos que $c * q_1 \in \ell_2$ e, pelo segundo caso já demonstrado acima, segue que $c * q = (c * q_1) * q_2 \in \ell_2$. ■

Definição 2.4.2. *A auto-correlação de uma função φ pode ser denotada da forma*

$$a_\varphi(x) = (\varphi * \varphi^v)(x) = \int_{-\infty}^{\infty} \varphi(t)\varphi(t-x)dt. \tag{2.4.8}$$

Para prosseguirmos com nosso esquema de reconstrução, devemos tratar da função geradora φ. Para nosso trabalho, devemos assumir que existem constantes $A, B \in \mathbb{R}$ tais que

$$0 < A \leq \hat{a_\varphi}(w) \leq B < \infty, \tag{2.4.9}$$

quase em todo ponto.

Se considerarmos $\varphi = \beta^1$, temos que

$$a_\varphi = \beta^1 * \beta^{1^\vee} = \beta^1 * \beta^1 = \beta^3 \tag{2.4.10}$$

e

$$a_\varphi = [a_\varphi] = [\beta^3] = [\ldots, 0, 0, \frac{1}{6}, \frac{4}{6}, \frac{1}{6}, 0, 0, \ldots] \tag{2.4.11}$$

Então

$$\begin{aligned} \hat{a_\varphi} &= \frac{1}{6}e^{-2\pi iw(-1)} + \frac{4}{6}e^{-2\pi iw0} + \frac{1}{6}e^{-2\pi iw\cdot 1} \\ &= \frac{4}{6} + \frac{1}{6}\left(e^{2\pi iw} + e^{-2\pi iw}\right) \\ &= \frac{4}{6} + \frac{2}{6}\left(\frac{e^{2\pi iw} + e^{-2\pi iw}}{2}\right) \\ &= \frac{4}{6} + \frac{1}{3}\sin 2\pi w \end{aligned} \tag{2.4.12}$$

Como $-1 \leq \sin 2\pi w \leq 1$, temos que

$$\begin{aligned} \frac{4}{6} - \frac{1}{3} \leq \hat{a_\varphi} \leq \frac{4}{6} + \frac{1}{3} \quad \Leftrightarrow \\ \Leftrightarrow \quad A := \frac{1}{3} \leq \hat{a_\varphi} \leq 1 := B \end{aligned} \tag{2.4.13}$$

Ao assumirmos a hipótese 2.4.9, traremos um resultado importante na reconstrução.

Teorema 2.4.4. *Seja $T > 0$ e seja φ um gerador tal que*

$$0 < A \leq \hat{a_\varphi}(w) \leq B < \infty, \tag{2.4.14}$$

quase em todo lugar, para algum A, $B \in \mathbb{R}$. Então $V_{\varphi,T}$ é um subespaço fechado de L_2.

Demonstração. (SACHT, 2014) ■

O próximo resultado é um dos mais importantes com respeito a amostragem e reconstrução de sinais.

Teorema 2.4.5. *Seja f uma função para qual existe $W > 0$ tal que $\hat{f}(w) = 0, \forall w \in \mathbb{R}\backslash[-W, W]$. Então, a aproximação dada em nosso esquema de reconstrução produz $\tilde{f}_T = f$, se $T \leq \dfrac{1}{2W}, \psi = \delta, q = \delta$ e $\varphi(x) = sinc(x) = \dfrac{\sin \pi x}{\pi x}$*

Demonstração. (SHANNON, 1949) ■

Este resultado no diz que, se tivermos um sinal f tal que sua Transformada de Fourier tenha banda limitada, isto é, $\hat{f}(w) = 0$ para todo $w \in \mathbb{R}\backslash[-W, W]$, para algum $W > 0$, sua reconstrução $\tilde{f}$, tendo válidas as hipóteses do Teorema 2.4.5, reconstrói exatamente o sinal f. Porém, este resultado não é muito usado na prática, pois geralmente, os sinais de áudio e de imagem não são de banda limitada. Além disso, $\varphi(x) = \dfrac{\sin \pi x}{\pi x}$ não tem suporte compacto, o que tornaria a convolução $c *_T \varphi$ impossível de ser implementada. Isso ocorre pelo fato de que o custo computacional desta operação é determinado pelo suporte de φ.

Por conta disso, dado φ de suporte compacto, precisamos encontrar qual função do espaço $V_{\varphi,T}$ está mais próxima de uma dada função f. Na métrica do espaço $\mathscr{L}_2$, a função que queremos é a projeção ortogonal da função f no espaço $V_{\varphi,T}$, ou seja, a função $c *_T \varphi \in V_{\varphi,T}$ que é solução para o problema

$$arg \min_{c \in \ell_2} \| f - (c *_T \varphi) \|_{\mathscr{L}_2} := P_{\varphi,T}(f) \tag{2.4.15}$$

e, também, uma característica da projeção ortogonal $P_{\varphi,T}(f)$ é que seu resíduo com a f, ou seja, $[P_{\varphi,T}(f) - f] \perp V_{\varphi,T}$.

O próximo resultado nos trará uma expressão para a projeção ortogonal da função f no espaço $V_{\varphi,T}$.

Teorema 2.4.6. *Sejam φ uma função geradora que satisfaz* (2.4.14)*, $T > 0$ e $f \in \mathscr{L}_2$. Então a projeção ortogonal de f em $V_{\varphi,T}$ é dada por*

$$P_{\varphi,T}(f) = [f * \varphi^{\vee}(\cdot/_T)]_T * [a_\varphi]^{-1} *_T \varphi(\cdot/_T) \tag{2.4.16}$$

em que $\psi = \varphi$ e $q = [a_\varphi]^{-1}$.

Demonstração. (SACHT, 2014) Para simplificarmos a demonstração, assumiremos $T = 1$. Se $[f - P_{\varphi,T}(f)]$ é ortogonal a $V_{\varphi,T}$, então podemos dizer que ,para algum $\varphi \in V_\varphi$ qualquer,

$$\langle f - P_{\varphi,T}(f), \varphi \rangle = 0, \forall k \in \mathbb{Z} \tag{2.4.17}$$

Como o produto interno em $\mathscr{L}$ é dado por: $\langle f, h\rangle = \int_{-\infty}^{\infty} f(x)\overline{g(x)}dx$, temos que

$$\langle f - P_{\varphi,T}(f), \varphi\rangle = \int_{-\infty}^{\infty} (f - P_{\varphi,T}(f)) \cdot \overline{\varphi(x)}dx \tag{2.4.18}$$

e podemos reescrever o descrito em 2.4.18 como

$$[(f - P_{\varphi,T}(f)) * \varphi^{\vee}] = 0 \tag{2.4.19}$$

Se $P_{\varphi,T}(f)) = c * \varphi$, temos que

$$\begin{aligned} & [(f - P_{\varphi,T}(f)) * \varphi^{\vee}] = 0 \quad \Leftrightarrow \\ \Leftrightarrow \quad & [(f - (c * \varphi) * \varphi^{\vee}] = 0 \quad \Leftrightarrow \\ \Leftrightarrow \quad & [f * \varphi^{\vee}] = c * [\varphi * \varphi^{\vee}] \quad \Leftrightarrow \\ \Leftrightarrow \quad & c = [f * \varphi^{\vee}] * [\varphi * \varphi^{\vee}]^{-1} \end{aligned} \tag{2.4.20}$$

Obtendo assim, que $\psi = \varphi$ e $q = [\varphi * \varphi^{\vee}]^{-1} = [a_{\varphi}]$ ■

2.4.1 Ordem de aproximação e Quasi-Interpoladores

Ao encontrarmos a melhor aproximação para a função $f \in \mathscr{L}_2$ em $V_{\varphi,T}$, a ideia é pensar no quão próximo de f está essa aproximação. Um modo de tratar desse conceito é com a seguinte definição:

Definição 2.4.3. *Um gerador φ tem **ordem de aproximação L** se L for o maior inteiro positivo de modo que exista uma constante $C > 0$ tal que*

$$\| f - P_{\varphi,T}(f) \|_{\mathscr{L}_2} \leq C \cdot T^L \cdot \| f^{(L)} \|_{\mathscr{L}_2}, \forall f \in W_2^L. \tag{2.4.21}$$

A ordem de aproximação nos dá um modo de compararmos os diferentes métodos de reconstrução. Quanto maior a ordem de aproximação L, o erro tende a zero mais rapidamente, quando $T \longrightarrow 0$. No teorema anterior, dados dois métodos diferentes de mesma ordem, aquele que tiver menor constante C será o método cujo erro tende a zero mais rapidamente quando $T \longrightarrow 0$.

Um importante resultado nos dá a ideia de determinarmos se um gerador dado tem ordem de aproximação L.

Teorema 2.4.7. *Seja φ uma função geradora que satisfaz* (2.4.14) *e tem suporte compacto. São equivalentes as afirmações a seguir:*

i. $p \in V_{\varphi,T}$, para todos os polinômios p de grau $L-1$ e $\forall T > 0$.

ii. $\exists \varphi_{QI} \in V_{\varphi,T}$ *tal que*

$$\forall p \text{ polinômio de grau } L-1, \sum_{k\in\mathbb{Z}} p(k)\varphi_{QI}(x-k) = p(x) \tag{2.4.22}$$

iii. $\hat{\varphi}(0) \neq 0$ *e* $\hat{\varphi}^{(m)}(k) = 0, \forall k \in \mathbb{Z}\backslash\{0\}$ *e* $\forall m \in \{0,\ldots,L-1\}$

iv. φ *tem ordem de aproximação* L

Demonstração. (SACHT, 2014) ■

Definição 2.4.4 (Quasi-Interpoladores)**.** *Dado um gerador* φ *e* $\psi = \delta$, $\varphi_{QI} = q * \varphi$ *é dito um* ***quasi-interpolador*** *de ordem* L*, se ele reproduz exatamente todos os polinômios de grau* L*,*

$$([p] * q * \varphi)(x) = p(x), \forall p \text{ polinômios de grau } L-1 \tag{2.4.23}$$

Quando φ satisfaz a afirmação *ii* do teorema 2.4.7 para ordem de aproximação L, temos a existência um quasi-interpolador.

2.4.2 Erro de aproximação

Até o momento, apenas podemos comparar o erro apresentado pelo processo de reconstrução através da ordem de aproximação das projeções ortogonais. A seguir, apresentaremos um resultado que quantifica o erro dependendo da escolha do T e que pode ser aplicado para diferentes escolhas de ψ, q *e* φ (SACHT, 2014).

Teorema 2.4.8. *Seja* ψ, q *e* φ *funções que satisfazem as condições de admissibilidade dadas na seção 2.4. Para todo* $f \in W_2^r$ *com* $r > \frac{1}{2}$*, o erro de aproximação é dado por*

$$\| f - \tilde{f}_T \|_{\mathscr{L}_2} = \left(\int_{-\infty}^{\infty} | \hat{f}(w) |^2 \, E(Tw) dw \right)^{\frac{1}{2}} + e(f,T), \tag{2.4.24}$$

em que $e(f,T) = o(T^r)$ *e*

$$E(w) = \left| 1 - (\hat{q}(w)^* \hat{\psi}^*)\hat{\varphi}(w) \right|^2 + \left| \hat{q}(w)\hat{\psi}(w) \right|^2 \sum_{k\neq 0} | \hat{\varphi}(w+k) |^2 \tag{2.4.25}$$

Demonstração. (BLU; UNSER, 1999) ■

3 Resultados e Conclusão

3.1 Filtros e Geradores

Para escrevermos o script de reconstrução de um áudio qualquer no MATLAB, vamos precisar efetuar, basicamente, duas operações: uma convolução discreta entre o som original, discretizado, o qual queremos reconstruir, e uma filtragem digital. A reconstrução, como vimos anteriormente, se dá com a combinação entre um filtro digital e uma função de reconstrução. Logo, para concluirmos nossa reconstrução, vamos aplicar o resultado dessa primeira convolução a uma função geradora, contínua por partes, operando-os com uma convolução mista.

Os filtros e funções geradoras que temos disponíveis para efetuar os testes são:

Filtros

Filtros Finite Impulse Response		
Dalai 1	*Dalai 2*	*Dalai 3*

Filtros Infinite Impulse Response		
β-spline 1	*β-spline 2*	*β-spline 3*
Condat 1	*Condat 2*	*Condat 3*
Sacht-Nehab 1	*Sacht-Nehab 2*	*Sacht-Nehab 3*
Omoms 2	*Omoms 3*	

Funções Geradoras

Geradores		
β-spline 1	*β-spline 2*	*β-spline 3*
Sacht-Nehab 1	*Sacht-Nehab 2*	*Sacht-Nehab 3*
Omoms 2	*Omoms 3*	

Os quais, na reconstrução de imagens, são combinados da seguinte forma:

- *ifir_β-spline 1 e β-spline 1*: (UNSER; ALDROUBI; EDEN, 1991);
- *ifir_β-spline 2 e β-spline 2*: (UNSER; ALDROUBI; EDEN, 1991);
- *ifir_β-spline 3 e β-spline 3*: (UNSER; ALDROUBI; EDEN, 1991);

- *ifir_condat 1 e β-spline 1*: (CONDAT; BLU; UNSER, 2005);
- *ifir_condat 2 e β-spline 2*: (CONDAT; BLU; UNSER, 2005);
- *ifir_condat 3 e β-spline 3*: (CONDAT; BLU; UNSER, 2005);
- *ifir_sacht_nehab 1 e sacht_nehab 1*: (SACHT; NEHAB, 2015);
- *ifir_sacht_nehab 2 e sacht_nehab 2*: (SACHT; NEHAB, 2015);
- *ifir_sacht_nehab 3 e sacht_nehab 3*: (SACHT; NEHAB, 2015);
- *fir_dalai 1 e β-spline 1*: (DALAI; LEONARDI; MIGLIORATI, 2005);
- *fir_dalai 2 e β-spline 2*: (DALAI; LEONARDI; MIGLIORATI, 2005);
- *fir_dalai 3 e β-spline 3*: (DALAI; LEONARDI; MIGLIORATI, 2005);
- *ifir_omoms 2 e omoms 2*: (BLU; THCVENAZ; UNSER, 2001);
- *ifir_omoms 3 e omoms 3*: (BLU; THCVENAZ; UNSER, 2001).

3.2 Escrevendo os Scripts

Nosso intuito neste trabalho é reconstruir um som, cuja amostragem esteja abaixo da usual, tornando-a de melhor qualidade, independente do seu fim. Por exemplo, uma caixa preta de avião grava um áudio, em média, de $3000Hz$ a $6000Hz$. Tal frequência nos dá um som que pode ser muito difícil de escutar com clareza. Para podermos comparar, a frequência de amostragem em que geralmente se gravam as músicas nos CDs e DVDs são de $44.100Hz$ a $48.000Hz$.

Por isso, e para podermos comparar o resultado da reconstrução, tomamos sons gravados a $44.100Hz$ e subamostramos, tornando-os menos audíveis. Para fazer a subamostragem, agimos da seguinte forma:

Considere um som mono (podemos também utilizar sons estéreos, mas para facilitar, trataremos um som mono) gravado a $44.100Hz$, tal que sua representação no MATLAB nos leva a um vetor com $(44100 \cdot tempodosom(s)) \times 1$. Subamostramos esse vetor escolhendo somente os valores de entrada, igualmente espaçados, 10 a 10. Nesse caso, formaremos um novo vetor com o som a $4.410Hz$.

De posse desse novo vetor, aplicamos a ele uma convolução discreta com um filtro digital q, previamente determinado e, em seguida, uma função geradora φ, que irá reconstruir este som, obtendo uma representação contínua, que amostramos a $44.100Hz$.

Para aplicarmos o filtro digital no som original, prosseguimos da seguinte maneira:

Aplicando os Filtros digitais no MATLAB

Os filtros digitais são FIR (Finite Impulse Response) ou $IFIR$ (Inverse Finite Impulse Response). Suas informações são descritas na forma de um vetor

$$q = [0, 0, \ldots, q_{-n}, q_{-(n-1)}, \ldots, q_{-1}, q_0.q_1, \ldots, q_{n-1}, q_n, \ldots, 0, 0],$$

com $q_i \in \mathbb{R}$ e $q_{-n} = q_n, \forall n \in \mathbb{N}$.

Exemplo 3.2.1. ***Filtro IFIR - Sacht Nehab Linear****:*

$$q = [0, \ldots, 0, -0.00272602, 0.11566267, 0.77412669, 0.11566267, -0.00272602, 0, \ldots, 0].$$

Para filtros FIR sendo $f : \mathbb{R} \longrightarrow \mathbb{C}$ um sinal de áudio qualquer, aplicamos uma convolução discreta entre o sinal e o filtro $q : \mathbb{Z} \longrightarrow \mathbb{C}$, criando um sinal de áudio $c : \mathbb{Z} \longrightarrow \mathbb{C}$

$$c = q * [f]_T,$$

de modo a aplicar o filtro em toda a extensão do sinal.

Então, para aplicarmos a convolução nos filtros FIR, devemos analisar ponto a ponto do domínio, ou seja, para algum $T > 0$,

$$c_k = \sum_{i \in \mathbb{Z}} q_{i-k} \cdot f(T \cdot i).$$

Como estamos trabalhando com MATLAB, o som é transformado em um vetor $N \times 1$ (ou matriz $N \times 2$ se for estéreo), em que N é o número de linhas do som e os índices vão de 1 até N.

Seja $q : \mathbb{Z} \longrightarrow \mathbb{C}$ um filtro FIR qualquer definido por

$$q = [0, \ldots, 0, q_{-2}, q_{-1}, q_0, q_1, q_2, 0, \ldots, 0]$$

com $q_i \in \mathbb{R}$, $q_{-2} = q_2, q_i = q_{-i}, i = 1, 2$, e seja $f : \mathbb{R} \longrightarrow \mathbb{C}$ um sinal de áudio amostrado.

Aplicaremos uma convolução discreta entre o filtro q e a função $[f]_T$ amostrada com $T = 1$ e analisaremos ponto a ponto para criar uma sequência $c : \mathbb{Z} \longrightarrow \mathbb{C}$ tal que

$$c_k = \sum_{i \in \mathbb{Z}} q_{i-k} \cdot f(T \cdot i).$$

Como $q_j = 0, \forall j \in \mathbb{Z} \backslash [-2, 2]$, temos:

$$c_1 = \sum_{i \in \mathbb{Z}} q_{1-i} \cdot f(i) \qquad \begin{cases} 1 - i \leq 2 \Leftrightarrow i \geq -1 \\ 1 - i \geq -2 \Leftrightarrow i \leq 3 \end{cases}$$

$$c_1 = \sum_{i=-1}^{3} q_{1-i} \cdot f(i) = q_2 \cdot f(-1) + q_1 \cdot f(0) + q_0 \cdot f(1) + q_{-1} \cdot f(2) + q_{-2} \cdot f(3)$$

$$c_2 = \sum_{i \in \mathbb{Z}} q_{2-i} \cdot f(i) \qquad \begin{cases} 2-i & \leq & 2 & \Leftrightarrow & i & \geq & 0 \\ 2-i & \geq & -2 & \Leftrightarrow & i & \leq & 4 \end{cases}$$

$$c_2 = \sum_{i=0}^{4} q_{2-i} \cdot f(i) = q_2 \cdot f(0) + q_1 \cdot f(1) + q_0 \cdot f(2) + q_{-1} \cdot f(3) + q_{-2} \cdot f(4)$$

$$c_3 = \sum_{i \in \mathbb{Z}} q_{3-i} \cdot f(i) \qquad \begin{cases} 3-i & \leq & 2 & \Leftrightarrow & i & \geq & 1 \\ 3-i & \geq & -2 & \Leftrightarrow & i & \leq & 5 \end{cases}$$

$$c_3 = \sum_{i=1}^{5} q_{3-i} \cdot f(i) = q_2 \cdot f(1) + q_1 \cdot f(2) + q_0 \cdot f(3) + q_{-1} \cdot f(4) + q_{-2} \cdot f(5)$$

$$c_4 = \sum_{i \in \mathbb{Z}} q_{4-i} \cdot f(i) \qquad \begin{cases} 4-i & \leq & 2 & \Leftrightarrow & i & \geq & 2 \\ 4-i & \geq & -2 & \Leftrightarrow & i & \leq & 6 \end{cases}$$

$$c_4 = \sum_{i=2}^{6} q_{4-i} \cdot f(i) = q_2 \cdot f(2) + q_1 \cdot f(3) + q_0 \cdot f(4) + q_{-1} \cdot f(5) + q_{-2} \cdot f(6)$$

$$\vdots$$

Se N for o número de linhas do som, teremos:

$$c_{N-4} = \sum_{i \in \mathbb{Z}} q_{N-4-i} \cdot f(i) \qquad \begin{cases} N-4-i & \leq & 2 & \Leftrightarrow & i & \geq & N-6 \\ N-4-i & \geq & -2 & \Leftrightarrow & i & \leq & N-2 \end{cases}$$

$$c_{N-4} = \sum_{i=N-6}^{N-2} q_{N-4-i} \cdot f(i) = q_2 \cdot f(N-6) + q_1 \cdot f(N-5) + q_0 \cdot f(N-4) + q_{-1} \cdot f(N-3) + q_{-2} \cdot f(N-2)$$

$$c_{N-3} = \sum_{i \in \mathbb{Z}} q_{N-3-i} \cdot f(i) \qquad \begin{cases} N-3-i & \leq & 2 & \Leftrightarrow & i & \geq & N-5 \\ N-3-i & \geq & -2 & \Leftrightarrow & i & \leq & N-1 \end{cases}$$

$$c_{N-3} = \sum_{i=N-5}^{N-1} q_{N-3-i} \cdot f(i) = q_2 \cdot f(N-5) + q_1 \cdot f(N-4) + q_0 \cdot f(N-3) + q_{-1} \cdot f(N-2) + q_{-2} \cdot f(N-1)$$

$$c_{N-2} = \sum_{i \in \mathbb{Z}} q_{N-2-i} \cdot f(i) \qquad \begin{cases} N-2-i & \leq & 2 & \Leftrightarrow & i & \geq & N-4 \\ N-2-i & \geq & -2 & \Leftrightarrow & i & \leq & N \end{cases}$$

$$c_{N-2} = \sum_{i=N-4}^{N} q_{N-2-i} \cdot f(i) = q_2 \cdot f(N-4) + q_1 \cdot f(N-3) + q_0 \cdot f(N-2) + q_{-1} \cdot f(N-1) + q_{-2} \cdot f(N)$$

$$c_{N-1} = \sum_{i \in \mathbb{Z}} q_{N-1-i} \cdot f(i) \qquad \begin{cases} N-1-i \leq 2 \Leftrightarrow i \geq N-3 \\ N-1-i \geq -2 \Leftrightarrow i \leq N+1 \end{cases}$$

$$c_{N-1} = \sum_{i=N-3}^{N+1} q_{N-1-i} \cdot f(i) = q_2 \cdot f(N-3) + q_1 \cdot f(N-2) + q_0 \cdot f(N-1) + q_{-1} \cdot f(N) + q_{-2} \cdot f(N+1)$$

$$c_N = \sum_{i \in \mathbb{Z}} q_{N-i} \cdot f(i) \qquad \begin{cases} N-i \leq 2 \Leftrightarrow i \geq N-2 \\ N-i \geq -2 \Leftrightarrow i \leq N+2 \end{cases}$$

$$c_N = \sum_{i=N-2}^{N+2} q_{N-i} \cdot f(i) = q_2 \cdot f(N-2) + q_1 \cdot f(N-1) + q_0 \cdot f(N) + q_{-1} \cdot f(N+1) + q_{-2} \cdot f(N+2).$$

Como no MATLAB os índices negativos e o zero do sinal f, $f(0), f(-1), f(-2)$, assim como $f(N+1), f(N+2)$, não são representados, e como nos filtros os valores $q_i = q_{-i}, i = 1, 2, \ldots$, compensamos a não existência dos valores dos sinais com índice negativo dobrando os valores que estão sendo multiplicados pelo mesmo filtro.

Este processo, denominado "Extensão por Espelhamento", é utilizado quando a convolução gera valores, em seu resultado, que dependem de pontos inexistentes no sinal original. Com isso, utilizam-se os índices dos filtros digitais, que compreendem o mesmo valor, para corrigir o valor inexistente, resultado da convolução (CUNHA; TEIXEIRA; VELHO, 2001).

Portanto, como nas duas primeiras linhas e nas duas últimas temos

$$c(1) = q_{-2} \cdot f(-1) + q_{-1} \cdot f(0) + q_0 \cdot f(1) + q_1 \cdot f(2) + q_2 \cdot f(3)$$

$$c(2) = q_{-2} \cdot f(0) + q_{-1} \cdot f(1) + q_0 \cdot f(2) + q_1 \cdot f(3) + q_2 \cdot f(4)$$

$$c(N-1) = q_{-2} \cdot f(N-3) + q_{-1} \cdot f(N-2) + q_0 \cdot f(N-1) + q_1 \cdot f(N) + q_2 \cdot f(N+1)$$

$$c(N) = q_{-2} \cdot f(N-2) + q_{-1} \cdot f(N-1) + q_0 \cdot f(N) + q_1 \cdot f(N+1) + q_2 \cdot f(N+2),$$

corrigindo os índices $0, -1, -2, N+1, N+2$ com a extensão por espelhamento, teremos

$$c(1) = q_0 \cdot f(1) + 2q_1 \cdot f(2) + 2q_2 \cdot f(3)$$

$$c(2) = q_{-1} \cdot f(1) + q_0 \cdot f(2) + q_1 \cdot f(3) + 2q_2 \cdot f(4)$$

$$c(N-1) = 2q_{-2} \cdot f(N-3) + q_{-1} \cdot f(N-2) + q_0 \cdot f(N-1) + q_1 \cdot f(N)$$

$$c(N) = 2q_{-2} \cdot f(N-2) + 2q_{-1} \cdot f(N-1) + q_0 \cdot f(N).$$

Percebe-se então que, se o sinal f é representado como

$$[f] = \begin{bmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ \vdots \\ f(N-3) \\ f(N-2) \\ f(N-1) \\ f(N) \end{bmatrix}_{N\times 1},$$

podemos representar a filtragem digital $q * [f]$ por um produto matriz-vetor $Q \cdot [f]$, em que

$$Q = \begin{bmatrix} q_0 & 2q_1 & 2q_2 & 0 & 0 & 0 & \dots & 0 \\ q_{-1} & q_0 & q_1 & 2q_2 & 0 & 0 & \dots & 0 \\ q_{-2} & q_{-1} & q_0 & q_1 & q_2 & 0 & \dots & 0 \\ 0 & q_{-2} & q_{-1} & q_0 & q_1 & q_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & q_{-2} & q_{-1} & q_0 & q_1 & q_2 & 0 \\ 0 & \cdots & 0 & q_{-2} & q_{-1} & q_0 & q_1 & q_2 \\ 0 & \dots & 0 & 0 & 2q_{-2} & q_{-1} & q_0 & q_1 \\ 0 & \dots & 0 & 0 & 0 & 2q_{-2} & 2q_{-1} & q_0 \end{bmatrix}_{N\times N}.$$

Portanto, temos a convolução $c = q * f$ representada no produto matriz-vetor a seguir:

$$c = Q * [f] = \begin{bmatrix} q_0 & 2q_1 & 2q_2 & 0 & 0 & 0 & \dots & 0 \\ q_{-1} & q_0 & q_1 & 2q_2 & 0 & 0 & \dots & 0 \\ q_{-2} & q_{-1} & q_0 & q_1 & q_2 & 0 & \dots & 0 \\ 0 & q_{-2} & q_{-1} & q_0 & q_1 & q_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & q_{-2} & q_{-1} & q_0 & q_1 & q_2 & 0 \\ 0 & \cdots & 0 & q_{-2} & q_{-1} & q_0 & q_1 & q_2 \\ 0 & \dots & 0 & 0 & 2q_{-2} & q_{-1} & q_0 & q_1 \\ 0 & \dots & 0 & 0 & 0 & 2q_{-2} & 2q_{-1} & q_0 \end{bmatrix} \cdot \begin{bmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ \vdots \\ f(N-3) \\ f(N-2) \\ f(N-1) \\ f(N) \end{bmatrix}.$$

Para filtros IFIR sendo $f : \mathbb{R} \longrightarrow \mathbb{C}$ um sinal de áudio amostrado qualquer, aplicamos uma convolução discreta entre o sinal e a inversa do filtro $q : \mathbb{Z} \longrightarrow \mathbb{C}$, criando um sinal de áudio $c : \mathbb{Z} \longrightarrow \mathbb{C}$

$$c = q^{-1} *_T f,$$

de modo a aplicar o filtro em toda a extensão do sinal.

Como $q * q^{-1} = q^{-1} * q = \delta = [0, \ldots, 0, 1, 0, \ldots, 0]$, devemos calcular o filtro q de modo que $q * c = f$.

De forma análoga ao filtro FIR, encontramos as seguintes expressões para f:

$$
\begin{aligned}
f(1) &= q_{-2} \cdot c(-1) + q_{-1} \cdot c(0) + q_0 \cdot c(1) + q_1 \cdot c(2) + q_2 \cdot c(3) \\
f(2) &= q_{-2} \cdot c(0) + q_{-1} \cdot c(1) + q_0 \cdot c(2) + q_1 \cdot c(3) + q_2 \cdot c(4) \\
f(3) &= q_{-2} \cdot c(1) + q_{-1} \cdot c(2) + q_0 \cdot c(3) + q_1 \cdot c(4) + q_2 \cdot c(5) \\
f(4) &= q_{-2} \cdot c(2) + q_{-1} \cdot c(3) + q_0 \cdot c(4) + q_1 \cdot c(5) + q_2 \cdot c(6) \\
&\vdots \\
f(N-3) &= q_{-2} \cdot c(N-5) + q_{-1} \cdot c(N-4) + q_0 \cdot c(N-3) + q_1 \cdot c(N-2) + q_2 \cdot c(N-1) \\
f(N-2) &= q_{-2} \cdot c(N-4) + q_{-1} \cdot c(N-3) + q_0 \cdot c(N-2) + q_1 \cdot c(N-1) + q_2 \cdot c(N) \\
f(N-1) &= q_{-2} \cdot c(N-3) + q_{-1} \cdot c(N-2) + q_0 \cdot c(N-1) + q_1 \cdot c(N) + q_2 \cdot c(N+1) \\
f(N) &= q_{-2} \cdot c(N-2) + q_{-1} \cdot c(N-1) + q_0 \cdot c(N) + q_1 \cdot c(N+1) + q_2 \cdot c(N+2)
\end{aligned}
$$

Corrigindo os valores que não são representados no MATLAB, pela extensão por espelhamento, podemos transformar esta convolução num produto matriz-vetor

$$
[f] = Q * c = \begin{bmatrix}
q_0 & 2q_1 & 2q_2 & 0 & 0 & 0 & \ldots & 0 \\
q_{-1} & q_0 & q_1 & 2q_2 & 0 & 0 & \ldots & 0 \\
q_{-2} & q_{-1} & q_0 & q_1 & q_2 & 0 & \ldots & 0 \\
0 & q_{-2} & q_{-1} & q_0 & q_1 & q_2 & \ldots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & \ldots & q_{-2} & q_{-1} & q_0 & q_1 & q_2 & 0 \\
0 & \cdots & 0 & q_{-2} & q_{-1} & q_0 & q_1 & q_2 \\
0 & \ldots & 0 & 0 & 2q_{-2} & q_{-1} & q_0 & q_1 \\
0 & \ldots & 0 & 0 & 0 & 2q_{-2} & 2q_{-1} & q_0
\end{bmatrix} \cdot \begin{bmatrix}
c(1) \\ c(2) \\ c(3) \\ c(4) \\ \vdots \\ c(N-3) \\ c(N-2) \\ c(N-1) \\ c(N)
\end{bmatrix},
$$

transformando nosso problema no sistema linear

$$\begin{bmatrix} q_0 & 2q_1 & 2q_2 & 0 & 0 & 0 & \dots & 0 & 0 \\ q_{-1} & q_0 & q_1 & 2q_2 & 0 & 0 & \dots & 0 & 0 \\ q_{-2} & q_{-1} & q_0 & q_1 & q_2 & 0 & \dots & 0 & 0 \\ 0 & q_{-2} & q_{-1} & q_0 & q_1 & q_2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & q_{-2} & q_{-1} & q_0 & q_1 & q_2 & 0 \\ 0 & 0 & \cdots & 0 & q_{-2} & q_{-1} & q_0 & q_1 & q_2 \\ 0 & 0 & \dots & 0 & 0 & 2q_{-2} & q_{-1} & q_0 & q_1 \\ 0 & 0 & \dots & 0 & 0 & 0 & 2q_{-2} & 2q_{-1} & q_0 \end{bmatrix} \cdot \begin{bmatrix} c(1) \\ c(2) \\ c(3) \\ c(4) \\ \vdots \\ c(N-3) \\ c(N-2) \\ c(N-1) \\ c(N) \end{bmatrix} = \begin{bmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ \vdots \\ f(N-3) \\ f(N-2) \\ f(N-1) \\ f(N) \end{bmatrix}.$$

Para resolvermos este sistema, vamos fatorar a matriz Q usando fatoração LU, de modo que $Q = LU$, em que U é uma matriz triangular tri-diagonal inferior e L é uma matriz triangular tri-diagonal superior.

A *Fatoração LU (ou decomposição LU)* é um método de decomposição de matrizes que é fornecido diretamente do método de eliminação gaussiano e é usada pelo ganho computacional na solução de problemas $Ax = b$ muito grandes. A matriz U é resultado final da eliminação gaussiana aplicada à matriz original A, enquanto a matriz L é o produto das inversas das matrizes elementares, que originaram a matriz U. Para mais detalhes, consultar o capítulo 17 do livro *Álgebra Linear* (LIMA, 2000).

Então, para resolvermos este sistema, utilizamos a função *mldivide*, ou *Backslash* "\", do MATLAB, própria para resolução de sistemas lineares $Ax = B$, que pode ser observado no apêndice A. Além disso, como a matriz Q é uma matriz, geralmente, muito grande (a leitura de um som mono de $5s$ de duração, amostrado a $44.100Hz$, gera uma amostragem do sinal de 237.568×1 e o processo de filtragem digital gera, neste caso, uma matriz Q 237.568×237.568). O armazenamento de uma matriz 237.568×237.568 necessitaria de uma quantidade extremamente alta de memória. Por conta disso, e pelo fato da matriz Q ser uma matriz esparsa, ou seja, possui uma grande quantidade de elementos que valem zero, utilizamos o método de construção de matrizes esparsas, também disponível nos scripts do apêndice A.

Portanto, fazendo

$$Q \cdot c = [f]$$

$$L \cdot U \cdot c = [f]$$

$$U \cdot c = L \backslash [f]$$

$$c = U \backslash (L \backslash [f]),$$

encontrando assim o sinal filtrado c aplicando o filtro IFIR q^{-1} no sinal amostrado $[f]$.

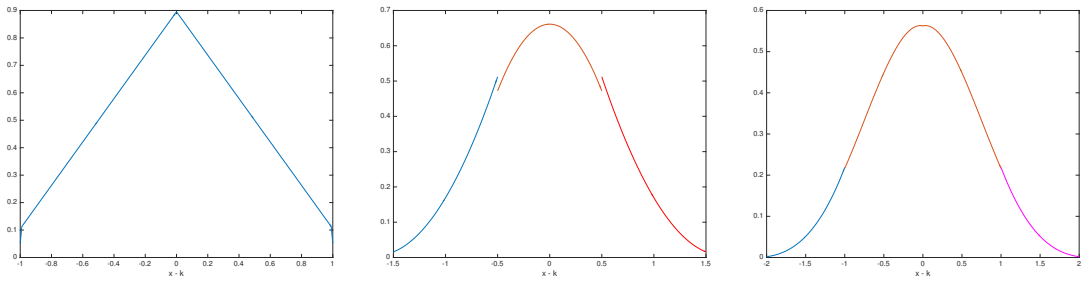
Reconstruindo o Sinal

Para reconstruirmos um sinal, vamos supor que queiramos reconstruir o sinal $f : \mathbb{R} \longrightarrow \mathbb{C}$ que está amostrado a $4.410Hz$. Sabendo que uma boa amostragem para som é por volta de $44.100Hz$, precisamos reconstruir este som para obtermos a amostragem que precisamos.

Consideremos $n = 10$ como sendo o número de vezes que iremos aumentar a resolução deste som. Após aplicarmos o filtro, seja ele FIR ou IFIR, encontramos o sinal de áudio $c : \mathbb{Z} \longrightarrow \mathbb{C}$ filtrado. Como o processo de filtragem do sinal não altera sua resolução, temos que o sinal c tem amostragem a $4410Hz$. Iremos construir o sinal $\tilde{f} : \mathbb{Z} \longrightarrow \mathbb{C}$ aplicando a convolução mista entre o sinal filtrado c e a função geradora $\varphi : \mathbb{R} \longrightarrow \mathbb{C}$, ou seja

$$\tilde{f}(x) = \sum_{k \in \mathbb{Z}} c(k)\varphi(x - k).$$

Cada função geradora φ, normalmente, é uma função Polinomial por Partes, ou seja, para determinados intervalos de seu domínio, a função assume uma forma polinomial distinta. Logo, no caso das funções geradoras, como descrito na convolução acima, para cada valor de $x - k$, a função φ poderá assumir formas diferentes. Por exemplo, as geradoras **Sacht Nehab 1**, **Sacht Nehab 2** e **Sacht Nehab 3** se diferenciam em sua aplicação na função original, e assumem formas diferentes, como vemos na figura 3.2.1.



(A) (B) (C)

Figura 3.2.1 – (A) Função Geradora Sacht Nehab 1; (B) Função Geradora Sacht Nehab 2; (C) Função Geradora Sacht Nehab 3.

Exemplo 3.2.2. ***Generate Function - Sacht Nehab 1****:* $r = x - k, x \in Dom(\tilde{f}), k \in \mathbb{Z}$*:*

$$\begin{array}{llllll} if & |r| < 1 & return & \varphi(r) & = & 0.89538176 - 0.79076352 \cdot |r| \\ else\ if & |r| = 1 & return & \varphi(r) & = & 0.05230912 \\ else & & return & \varphi(r) & = & 0. \end{array}$$

Para reconstruirmos um sinal usando uma função geradora $\varphi : \mathbb{R} \longrightarrow \mathbb{C}$ qualquer, devemos analisar as condições estabelecidas para r, para assim podermos calcular o som

reconstruído $\tilde{f}$, como observamos na figura 3.2.2. Supondo que a função geradora φ imprime condições para $|r| = 0, |r| \leq 1$ *e* $|r| \leq 2$, vamos analisar os casos

$$\tilde{f}(x) = \sum_{k\in\mathbb{Z}} c(k)\varphi(x-k).$$

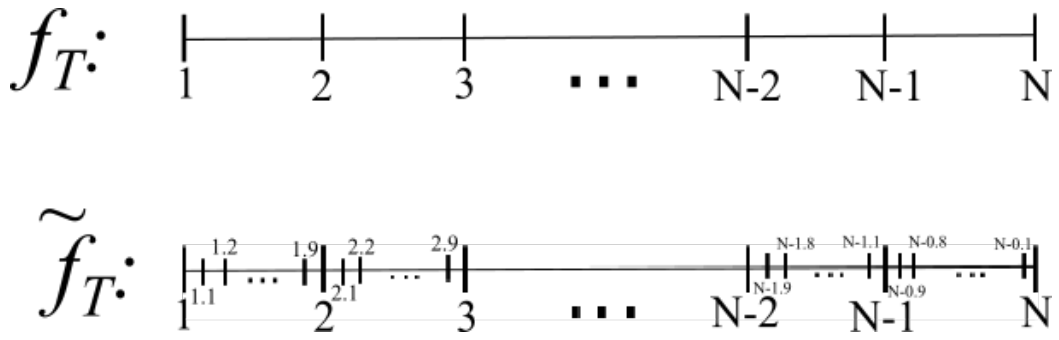


Figura 3.2.2 – A figura nos indica que a função geradora irá, a partir dos pontos da f, construir no sinal $\tilde{f}$ os pontos que fariam parte da amostragem original.

Iremos analisar ponto a ponto a convolução descrita acima para a reconstrução do sinal f:

$$\tilde{f}(1) = \sum_{k\in\mathbb{Z}} c(k)\cdot\varphi(1-k) \qquad \begin{cases} 1-k \leq 2 \Leftrightarrow k \geq -1 \\ 1-k \geq -2 \Leftrightarrow k \leq 3 \end{cases}$$

$$\tilde{f}(1) = \sum_{k=-1}^{3} c(k)\cdot\varphi(1-k) = c(-1)\cdot\varphi(2)+c(0)\cdot\varphi(1)+c(1)\cdot\varphi(0)+c(2)\cdot\varphi(-1)+c(3)\cdot\varphi(-2)$$

$$\tilde{f}(1.1) = \sum_{k\in\mathbb{Z}} c(k)\cdot\varphi(1.1-k) \qquad \begin{cases} 1.1-k \leq 2 \Leftrightarrow k \geq -0.9 \\ 1,1-k \geq -2 \Leftrightarrow k \leq 3.1 \end{cases}$$

$$\tilde{f}(1.1) = \sum_{k=0}^{3} c(k)\cdot\varphi(1.1-k) = c(0)\cdot\varphi(1.1)+c(1)\cdot\varphi(0.1)+c(2)\cdot\varphi(-0.9)+c(3)\cdot\varphi(-1.9)$$

$$\tilde{f}(1.2) = \sum_{k\in\mathbb{Z}} c(k)\cdot\varphi(1.2-k) \qquad \begin{cases} 1.2-k \leq 2 \Leftrightarrow k \geq -0.8 \\ 1.2-k \geq -2 \Leftrightarrow k \leq 3.2 \end{cases}$$

$$\tilde{f}(1.2) = \sum_{k=0}^{3} c(k)\cdot\varphi(1.2-k) = c(0)\cdot\varphi(1.2)+c(1)\cdot\varphi(0.2)+c(2)\cdot\varphi(-0.8)+c(3)\cdot\varphi(-1.8)$$

$$\vdots$$

$$\tilde{f}(2) = \sum_{k\in\mathbb{Z}} c(k)\cdot\varphi(2-k) \qquad \begin{cases} 2-k & \leq & 2 & \Leftrightarrow & k & \geq & 0 \\ 2-k & \geq & -2 & \Leftrightarrow & k & \leq & 4 \end{cases}$$

$$\tilde{f}(2) = \sum_{k=0}^{4} c(k)\cdot\varphi(2-k) = c(0)\cdot\varphi(2)+c(1)\cdot\varphi(1)+c(2)\cdot\varphi(0)+c(3)\cdot\varphi(-1)+c(4)\cdot\varphi(-2)$$

$$\tilde{f}(2.1) = \sum_{k\in\mathbb{Z}} c(k)\cdot\varphi(2.1-k) \qquad \begin{cases} 2.1-k & \leq & 2 & \Leftrightarrow & k & \geq & 0.1 \\ 2.1-k & \geq & -2 & \Leftrightarrow & k & \leq & 4.1 \end{cases}$$

$$\tilde{f}(2.1) = \sum_{k=1}^{4} c(k)\cdot\varphi(2.1-k) = c(1)\cdot\varphi(1.1)+c(2)\cdot\varphi(0.1)+c(3)\cdot\varphi(-0.9)+c(4)\cdot\varphi(-1.9)$$

$$\tilde{f}(2.2) = \sum_{k\in\mathbb{Z}} c(k)\cdot\varphi(2.2-k) \qquad \begin{cases} 2.2-k & \leq & 2 & \Leftrightarrow & k & \geq & 0.2 \\ 2.2-k & \geq & -2 & \Leftrightarrow & k & \leq & 4.2 \end{cases}$$

$$\tilde{f}(2.2) = \sum_{k=1}^{4} c(k)\cdot\varphi(1.2-k) = c(1)\cdot\varphi(1.2)+c(2)\cdot\varphi(0.2)+c(3)\cdot\varphi(-0.8)+c(4)\cdot\varphi(-1.8)$$

$$\vdots$$

Se N for o número de pontos no sinal de áudio amostrado $[f]$, devemos também analisar os últimos valores do sinal $\tilde{f}$:

$$\tilde{f}(N-1.1) = \sum_{k\in\mathbb{Z}} c(k)\cdot\varphi(N-1.1-k) \qquad \begin{cases} N-1.1-k & \leq & 2 & \Leftrightarrow & k & \geq & N-3.1 \\ N-1.1-k & \geq & -2 & \Leftrightarrow & k & \leq & N+0.9 \end{cases}$$

$$\begin{aligned} \tilde{f}(N-1.1) &= \sum_{k=N-3}^{N} c(k)\cdot\varphi(N-1.1-k) \\ &= c(N-3)\cdot\varphi(1.9)+c(N-2)\cdot\varphi(0.9) \\ &\quad +c(N-1)\cdot\varphi(-0.1)+c(N)\cdot\varphi(-1.1) \end{aligned}$$

$$\tilde{f}(N-1) = \sum_{k\in\mathbb{Z}} c(k)\cdot\varphi(N-1-k) \qquad \begin{cases} N-1-k & \leq & 2 & \Leftrightarrow & k & \geq & N-3 \\ N-1-k & \geq & -2 & \Leftrightarrow & k & \leq & N+1 \end{cases}$$

$$\begin{aligned} \tilde{f}(N-1) &= \sum_{k=N-3}^{N+1} c(k)\cdot\varphi(N-1-k) \\ &= c(N-3)\cdot\varphi(2)+c(N-2)\cdot\varphi(1) \\ &\quad +c(N-1)\cdot\varphi(0)+c(N)\cdot\varphi(-1) \\ &\quad +c(N+1)\cdot\varphi(-2) \end{aligned}$$

$$\tilde{f}(N-0.9)=\sum_{k\in\mathbb{Z}} c(k)\cdot\varphi(N-0.9-k) \qquad \begin{cases} N-0.9-k \leq 2 \Leftrightarrow k \geq N-2.9 \\ N-0.9-k \geq -2 \Leftrightarrow k \leq N+1.1 \end{cases}$$

$$\begin{aligned} \tilde{f}(N-0.9) &= \sum_{k=N-2}^{N+1} c(k)\cdot\varphi(N-0.9-k) \\ &= c(N-2)\cdot\varphi(1.1)+c(N-1)\cdot\varphi(0.1) \\ &\quad +c(N)\cdot\varphi(-0.9)+c(N+1)\cdot\varphi(-1.9) \end{aligned}$$

$$\vdots$$

$$\tilde{f}(N)=\sum_{k\in\mathbb{Z}} c(k)\cdot\varphi(N-k) \qquad \begin{cases} N-k \leq 2 \Leftrightarrow k \geq N-2 \\ N-k \geq -2 \Leftrightarrow k \leq N+2 \end{cases}$$

$$\begin{aligned} \tilde{f}(N) &= \sum_{k=N-2}^{N+2} c(k)\cdot\varphi(N-k) \\ &= c(N-2)\cdot\varphi(2)+c(N-1)\cdot\varphi(1) \\ &\quad +c(N)\cdot\varphi(0)+c(N+1)\cdot\varphi(-1) \\ &\quad +c(N+2)\cdot\varphi(-2). \end{aligned}$$

Da forma análoga à análise feita nos filtros digitais, há alguns pontos resultantes na convolução com a função geradora que não são representados no MATLAB. Assim, corrigimos essa falta, usando extensão por espelhamento, dobrando o valor que representará a mesma condição dada pela função geradora. Assim, teremos que

$$\tilde{f}(1)=c(1)\cdot\varphi(0)+2\cdot c(2)\cdot\varphi(-1)+2\cdot c(3)\cdot\varphi(-2)$$

$$\tilde{f}(1.1)=c(1)\cdot\varphi(0.1)+c(2)\cdot\varphi(-0.9)+2\cdot c(3)\cdot\varphi(-1.9)$$

$$\tilde{f}(1.2)=c(1)\cdot\varphi(0.2)+c(2)\cdot\varphi(-0.8)+2\cdot c(3)\cdot\varphi(-1.8),$$

$$\vdots$$

$$\tilde{f}(2)=c(1)\cdot\varphi(1)+c(2)\cdot\varphi(0)+c(3)\cdot\varphi(-1)+2\cdot c(4)\cdot\varphi(-2),$$

$$\tilde{f}(N-1) = 2 \cdot c(N-3) \cdot \varphi(2) + c(N-2) \cdot \varphi(1) + c(N-1) \cdot \varphi(0) + c(N) \cdot \varphi(-1),$$

$$\tilde{f}(N-0.9) = 2 \cdot c(N-2) \cdot \varphi(1.1) + c(N-1) \cdot \varphi(0.1) + c(N) \cdot \varphi(-0.9),$$

$$\vdots$$

$$\tilde{f}(N) = 2 \cdot c(N-2) \cdot \varphi(2) + 2 \cdot c(N-1) \cdot \varphi(1) + c(N) \cdot \varphi(0),$$

concluindo nossa análise para montarmos os scripts, disponíveis no apêndice A.

3.3 Resultados

Para obtermos um áudio com baixa resolução, tomamos um som em formato *Waveform Audio File Format (.wav)*, com resolução padrão ($44100Hz$) e tomamos uma subamostragem do sinal. Por exemplo, para reduzir a amostragem de um sinal de áudio, de $44100Hz$ para $4410Hz$, copiamos para um novo vetor (ou matriz, caso o som seja estéreo) apenas os valores do som original com espaçamento de 10 em 10, ou seja, se s é o som original e s' é o som subamostrado, teremos que $s'(1) = s(1)$, $s'(2) = s(11)$, $s'(3) = s(21)$, e assim por diante.

Ao testarmos os scripts nos sons mal amostrados, não tínhamos clareza de que estava havendo alguma reconstrução do som. Portanto, para nos auxiliar nos testes, buscamos uma forma visual de comparar o som original com o som reconstruído.

Para isso, obtemos os espectrogramas dos sinais com que estamos trabalhando. O espectrograma é uma representação visual do espectro das frequências de um som ou qualquer outro sinal que varia em função do tempo. Para calcular o espectrograma de um sinal, seja x um sinal, representado por um vetor $N \times 1, N \in \mathbb{N}$. Como o vetor x é dado por $x = [x(1), x(2), \ldots, x(N)]^t$, sua transformada de Fourier é dada por

$$\hat{x}(w) = \sum_{n=0}^{N} x(n) \cdot e^{\frac{-i2\pi w}{N} n}, \quad w = 1, \ldots N.$$

Podemos representar o cálculo da transformada de Fourier de x como o produto matriz-vetor

$$\hat{x} = \overline{F} \cdot x,$$

onde $\overline{F}$ é o conjugado de F e F é a matriz de Fourier $N \times N$:

$$F = \begin{bmatrix} e^{\frac{i2\pi}{N}} & e^{\frac{i4\pi}{N}} & \cdots & e^{\frac{i2\pi N}{N}} \\ e^{\frac{i4\pi}{N}} & e^{\frac{i8\pi}{N}} & \cdots & e^{\frac{i2\pi N}{N}} \\ \vdots & \vdots & \ddots & \vdots \\ e^{i2\pi\frac{N}{N}} & e^{i2\pi\frac{2N}{N}} & \cdots & e^{i2\pi\frac{N^2}{N}} \end{bmatrix}.$$

Para calcularmos o espectrograma de x, considere $m \in \mathbb{N}$, com $m < N$ e tome a matriz $X \in \mathbb{C}^{m\times(N-m)}$ como sendo a matriz cujo vetor $[x(1), x(1), \cdots, x(m)]^t$ é sua primeira coluna, $[x(2), x(3), \cdots, x(m+1)]^t$ é sua segunda coluna, e assim por diante. O **espectrograma** do sinal x é a matriz $\hat{X}$ dada por

$$\hat{X} = \overline{F} \cdot X.$$

Ao tirarmos os espectrogramas do som mal amostrado, do original e da sua reconstrução, e compararmos seus gráficos, dados nas figuras 3.3.1, 3.3.2 e 3.3.3, tivemos um indicativo de que os métodos de reconstrução de imagens funcionam na reconstrução de sons, auxiliando-nos, pois ouvir os áudios das reconstruções não nos davam a certeza da aproximação entre eles.

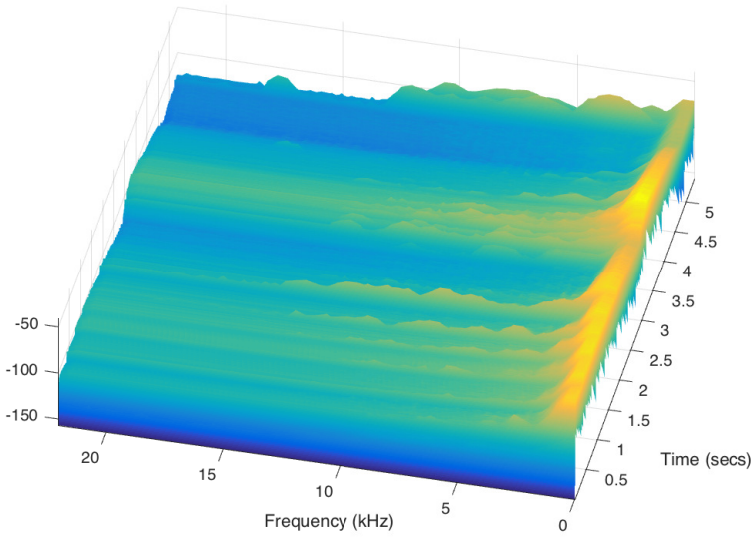


Figura 3.3.1 – Spectrograma do som de um violão acústico, tocando notas graves e agudas, a $44.100Hz$.

As imagens nos indicam que, ao aplicarmos os métodos de reconstrução de imagens a um som que esteja mal amostrado, há na sua reconstrução uma maior proximidade com o que deveria ser um som com amostragem adequada, tornando, assim, os métodos de reconstrução de imagens testados possíveis métodos de reconstrução de sons. Porém, as comparações visuais dos espectrogramas não nos dão clareza para concluir qual dos

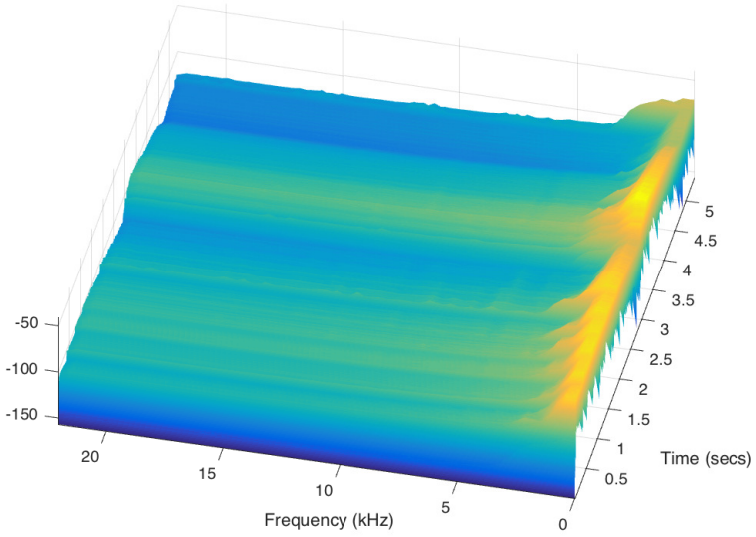


Figura 3.3.2 – Spectrograma do mesmo som de violão acústico, reconstruído usando Filtro e Gerador **Sacht Nehab 2** a $44100Hz$.

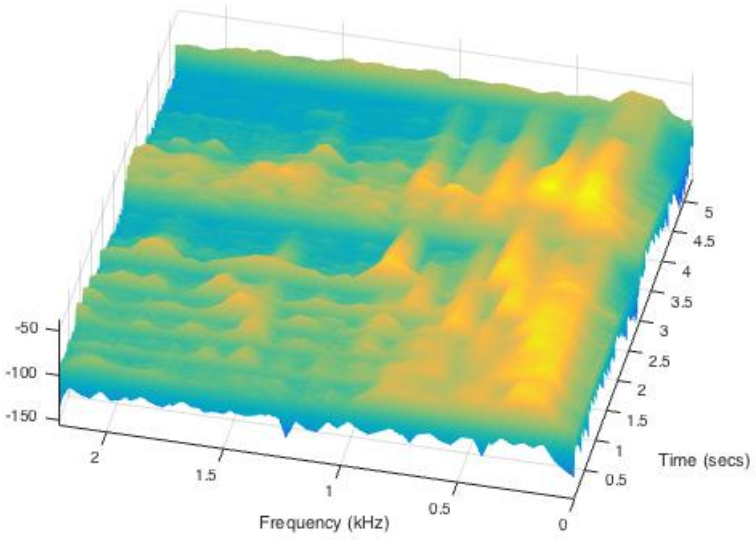


Figura 3.3.3 – Spectrograma do mesmo som de violão acústico, mal amostrado, a $4410Hz$.

métodos é o melhor para aplicar em áudio. Para isso, precisaremos usar alguma ferramenta para medir aproximação entre o sinal original e o sinal reconstruído.

Neste sentido, o método denominado **Peak signal-to-noise ratio (PSNR)** é o mais comumente é utilizado para medir qualidade de reconstrução de imagens.

Para utilizarmos este método, precisamos primeiro calcular o chamado Erro Médio Quadratico (**Mean Squared Error - MSE**), em que, se f é o sinal original, $\tilde{f}$ sua reconstrução e N o tamanho dos sinais, podemos definir o MSE entre f e $\tilde{f}$ da

seguinte forma:

$$MSE = \frac{1}{N}\sum_{i=1}^{N-1}\left[f(i) - \tilde{f}(i)\right]^2.$$

Então, podemos calcular o PSNR (em dB) da seguinte forma:

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX^2}{MSE}\right),$$

em que MAX é o valor máximo, em decibéis, que podemos encontrar em um som (no caso do MATLAB, a função *audioread*, que transforma o som em um vetor (ou matriz, no caso estéreo), gera sons em no máximo $1dB$. Logo, no caso de medirmos o PSNR da reconstrução no MATLAB, devemos usar $MAX = 1$). Neste método, quanto maior for o PSNR, mais próximo do original está o sinal reconstruído.

Para podermos concluir qual dos métodos de reconstrução de imagens é o que melhor aproxima a reconstrução de um som, criamos um banco de dados de sons diferentes para testarmos os métodos e fazermos a análise. Este banco de dados se constitui de sons criados sinteticamente, através de simuladores artificiais, para criar sons limpos e bem definidos e de sons de fala obtidos no site *Free Sound*[1], de sons livres. Tal banco contém um som de sino, um violão tocando notas de grave a agudas, uma sitar, uma nota solo de violoncelo, uma clarineta, um som de orquestra completa, uma fala masculina, uma fala feminina e um som de bateria em ritmo. Tais sons foram selecionados e gravados para apresentar a mais variada amplitude sonora, e com isso, obtermos uma conclusão mais precisa sobre os métodos de reconstrução.

Portanto, usando este banco de dados, criamos um script (disponível também no apêndice A) para comparar os métodos de reconstrução em diversos níveis de subamostragem do som. Neste script, geramos subamostragens de cada áudio em diferentes níveis, e a cada subamostragem, reconstruímos para resolução original, usando todos os métodos, e tomamos a distância entre o som reconstruído e o som original, via PSNR. Como resultado desta comparação, foram plotados gráficos mostrando que, para cada nível de subamostragem do som, este gera um erro, cujo valor é dado pelo cálculo do PSNR.

Esses gráficos foram feitos da seguinte forma: Primeiro subamostramos o som original, tomando apenas $\frac{1}{6}$ dos seus pontos e em seguida reconstruímos para seu tamanho original, usando cada método separadamente, e medimos seu PSNR. Logo depois, tomamos $\frac{1}{7}$ dos pontos do som original e em seguida reconstruímos para seu tamanho original e medimos seu PSNR. Este processo continua até fecharmos os gráficos, que são construídos em função do "Aumento de Resolução" e do seu respectivo PSNR.

[1] https://freesound.org/

Ao aplicarmos os métodos no som de uma bateria, podemos observar no gráfico 3.3.4 que o melhor método de reconstrução de sons, dentre os métodos de reconstrução de imagens, é o que utiliza o Filtro **IFIR Bspline 1** e a função geradora **Bspline 1**.

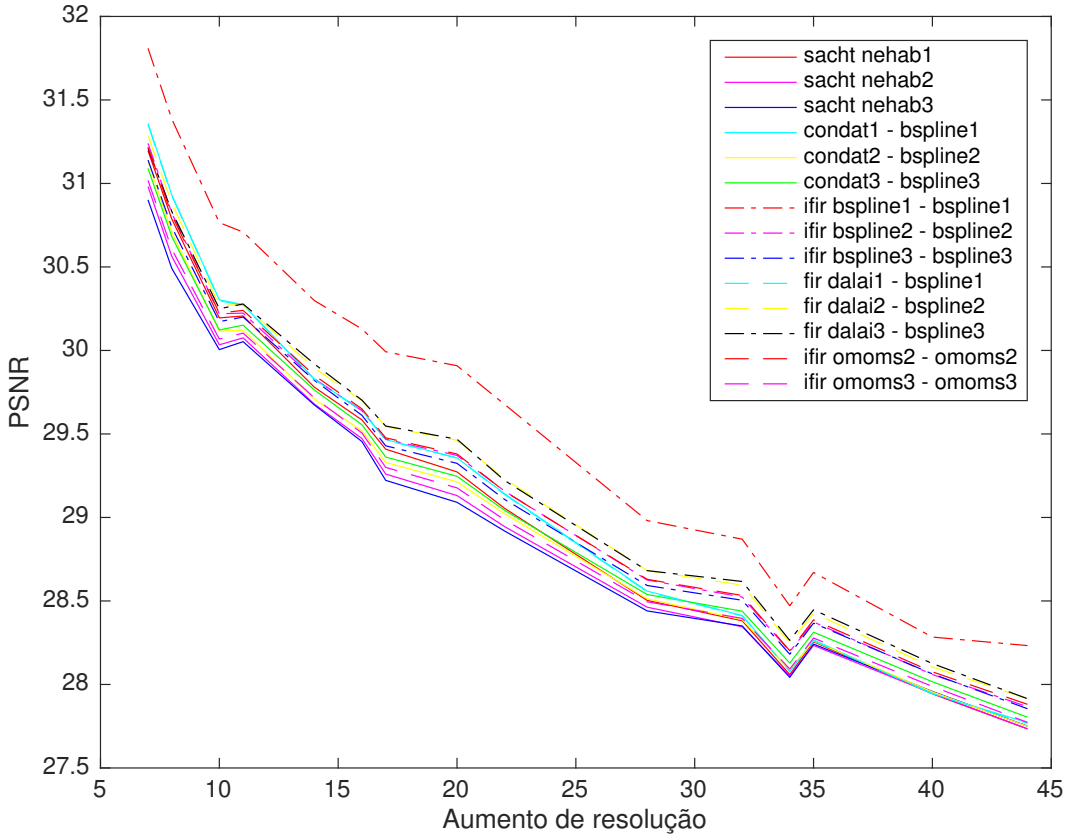


Figura 3.3.4 – Gráfico PSNR de reconstrução do som de uma bateria.

Podemos observar também que nos sons dos sinos (figura 3.3.5), voz de fala feminina (figura 3.3.6) e no som de orquestra completa sintetizada (figura 3.3.7), o método que utiliza o Filtro **IFIR Bspline 1** e a função geradora **Bspline 1** também se destaca na reconstrução do sinal.

O método **IFIR Bspline 1 – Bspline 1** é ainda a melhor alternativa para reconstruirmos os sons de guitarra e de fala masculina que estejam com amostragem a no mínimo por volta de $2205Hz$. A partir daí, o método passa a ser o que pior reconstrói um som, como podemos observar nas figuras 3.3.8 e 3.3.9.

No som de guitarra, o melhor método para sinais com amostragem entre $1102Hz$ e $2205Hz$ é o **FIR Dalai 3 – Bspline 3**, e para sinais com amostragem menor que $1102Hz$ é o **IFIR Sacht Nehab 3 – Sacht Nehab 3**. No som de fala masculina, o melhor método de reconstrução para sinais amostrados abaixo de $2205Hz$ é o **FIR Dalai 3 – Bspline 3**, como podemos observar na figura 3.3.9.

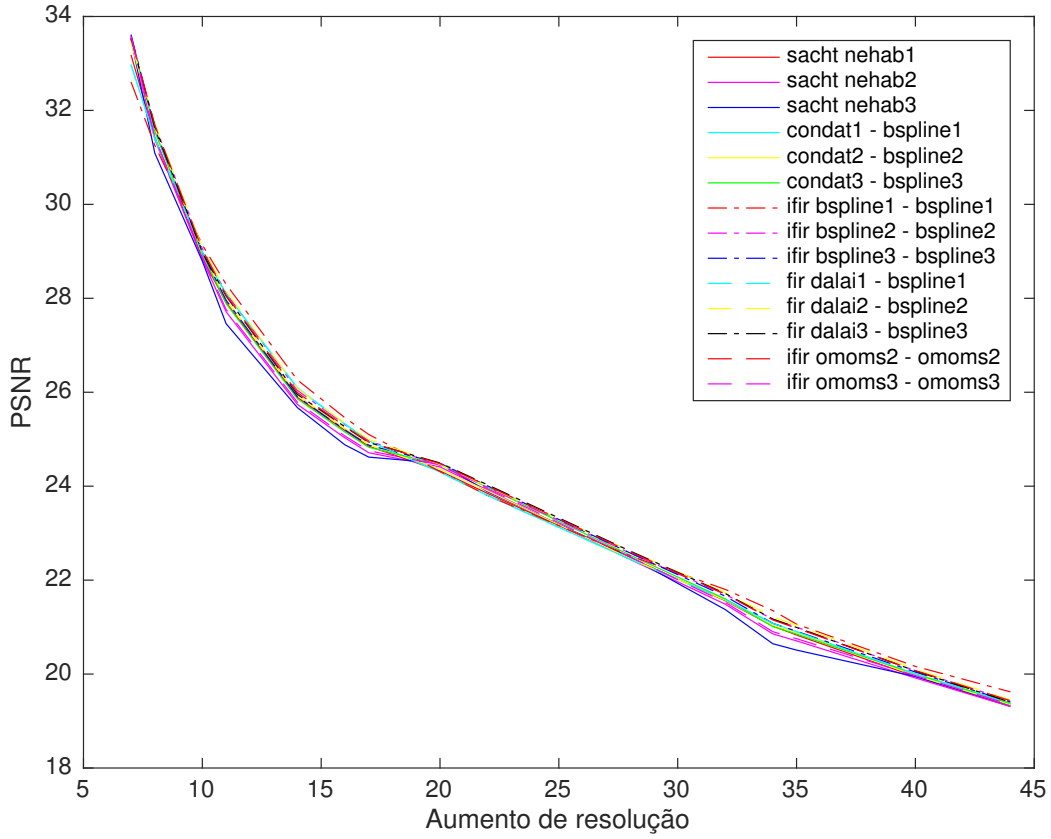


Figura 3.3.5 – Gráfico PSNR de reconstrução do som de sinos.

No som da Sitar Indiana, ao observar o gráfico da figura 3.3.10, podemos concluir que as reconstruções obtidas pelos métodos **FIR Dalai 3 – Bspline 3**, **FIR Dalai 2 – Bspline 2**, **IFIR Omoms 2 – Omoms 2** e **IFIR Bspline 3 – Bspline 3** estão muito próximas e seus gráficos PSNR estão praticamente sobrepostos. Segue, portanto, que estes são os melhores métodos para reconstruir um som de uma Sitar Indiana, dentre os métodos aqui testados.

No som de violão, os métodos **IFIR Sacht Nehab 3 – Sacht Nehab 3** e **IFIR Sacht Nehab 2 – Sacht Nehab 2** oscilam a partir da variação do aumento da resolução. Como observamos no gráfico da figura 3.3.11, apenas a partir de sons abaixo de $1470Hz$, o método **IFIR Sacht Nehab 3 – Sacht Nehab 3** passa a ser o que melhor reconstrói o sinal.

No som de violoncelo sintetizado, com exceção dos métodos **IFIR Sacht Nehab 1 – Sacht Nehab 1**, **Condat 1 – Bspline 1** e **IFIR Bspline 1 – Bspline 1**, cujos gráficos PSNR estão bem abaixo dos demais, as reconstruções aparentam ser muito próximas umas das outras, de modo que seus gráficos PSNR fiquem praticamente sobrepostos, como podemos observar na figura 3.3.12.

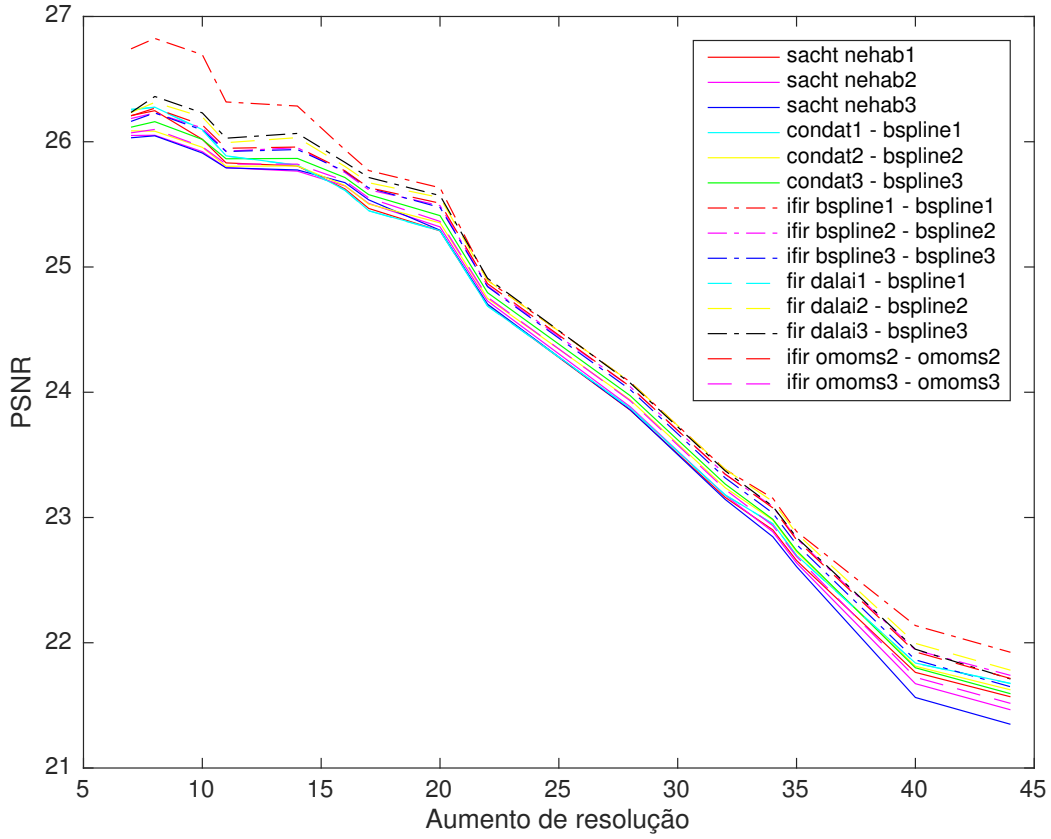


Figura 3.3.6 – Gráfico PSNR de reconstrução do som de fala feminina.

3.4 Conclusão

Com o intuito de analisar os principais métodos de reconstrução de imagens para reconstrução de sinais de áudio, analisamos os principais resultados que mostram a validade da aplicação. Para estes resultados, foi necessário avaliar as teorias que envolvem as Transformadas de Fourier e Convolução. Para transformada de Fourier, foi necessário avaliar em quais condições uma função pode ser representada pela sua série de Fourier. Com tais resultados, construímos a transformada de Fourier de uma função no espaço de Schwartz. Para analisarmos a Convolução, operação essa base de toda nossa aplicação, fez-se sua construção, além de suas principais propriedades e o Teorema da Convolução, resultado importantíssimo em diversas áreas da Matemática.

Com o domínio da Transformada de Fourier e da Convolução, analisamos os principais resultados que deram admissibilidade para a nossa aplicação, resultados esses que mostravam que podemos reconstruir um sinal a partir da convolução discreta entre um sinal amostrado e um filtro digital, e da convolução mista do resultado da convolução anterior com uma função geradora.

Em posse destes resultados, pudemos fazer uma análise das convoluções, ob-

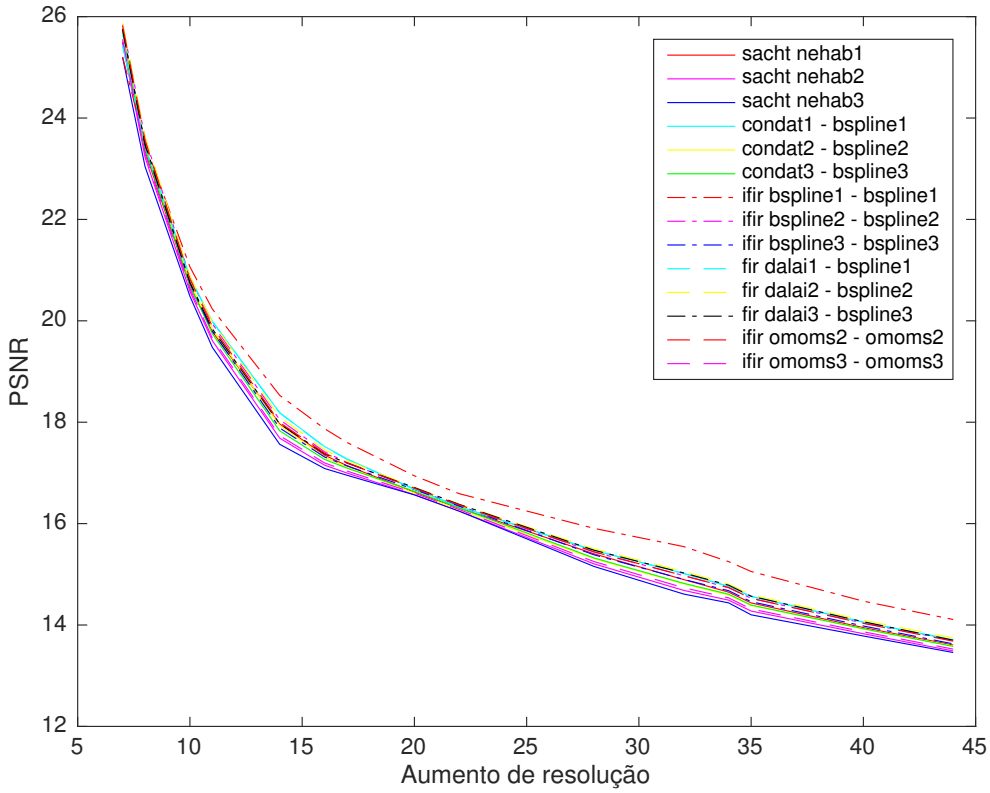


Figura 3.3.7 – Gráfico PSNR de reconstrução do som de uma orquestra completa sintetizada.

tendo um modelo para construção dos scripts, em MATLAB. Além disso, pudemos criar condições para montar um banco de sons e analisar, caso a caso, a reconstrução de sons, utilizando os principais métodos de reconstrução de imagens, como propusemos desde o início.

Portanto, de acordo com nossa análise feita a partir dessas amostras de som, o método de reconstrução de imagens que melhor reconstrói sinais de áudio é o **IFIR Bspline 1 – Bspline 1**. Apesar de ele não reconstruir bem alguns sons, como o som do violão e do violoncelo, ele mostrou ser o melhor método para sons de bateria, de sino, de orquestra e de sons de fala feminina, além de ser o melhor também para sons de fala masculina e guitarra que estão amostrados a frequências maiores que $2205Hz$. Vale lembrar que, de acordo com (SACHT, 2014), em suas análises, dentre os métodos que aqui estão sendo analisados, o método **IFIR Bspline 1 – Bspline 1** é o que apresenta o pior resultado na reconstrução de imagens.

Além disso, podemos concluir também que, de acordo com as imagens obtidas das amostras de som, o método **FIR Dalai 3 – Bspline 3** aparece, predominantemente, como um dos melhores métodos de reconstrução. Portanto, pela sua consistente eficácia em diferentes tipos de sons, em comparação com outros métodos, podemos concluir que,

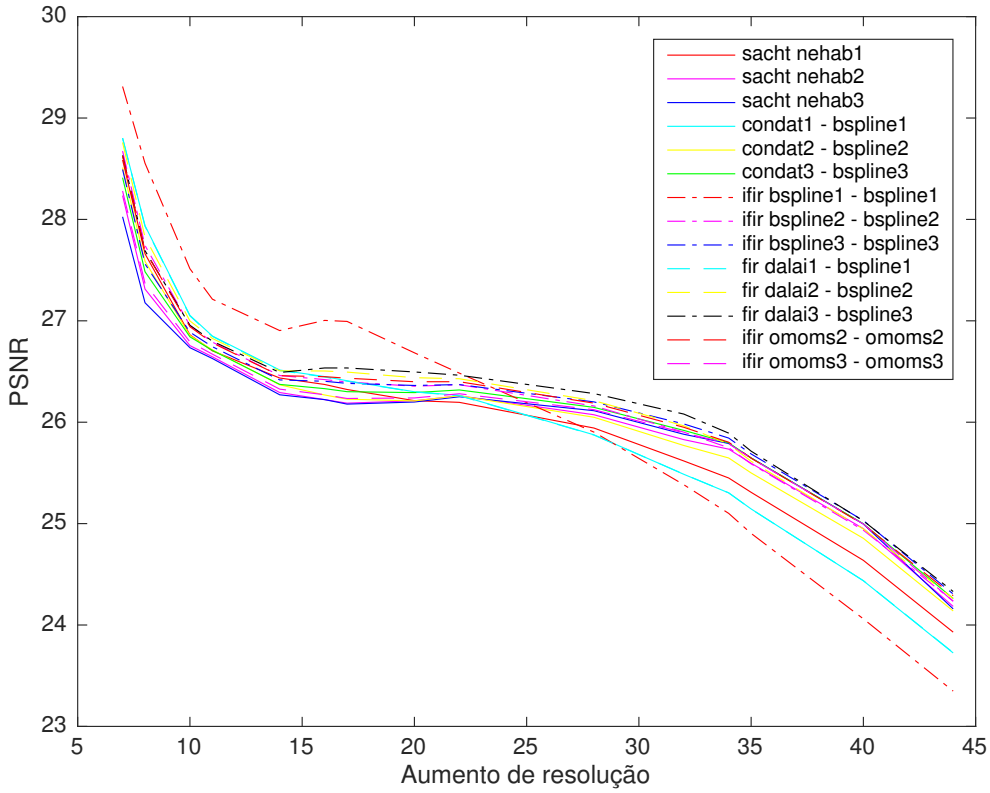


Figura 3.3.8 – Gráfico PSNR de reconstrução do som de uma fala masculina.

de modo geral, o método **FIR Dalai 3 – Bspline 3** é um bom esquema de reconstrução de áudio.

Visando trabalhos futuros, podemos pensar em quais modificações nos métodos podemos fazer para melhorar ainda mais a reconstrução de sinais de áudio. Podemos também analisar as diversas aplicações dessa reconstrução, para construir um programa ou aplicativo que se utilize desses métodos para de fato reconstruir sons no dia a dia de quem necessite. Além disso, um estudo mais aprofundado das métricas utilizadas para comparação das reconstruções de som poderá ser feito.

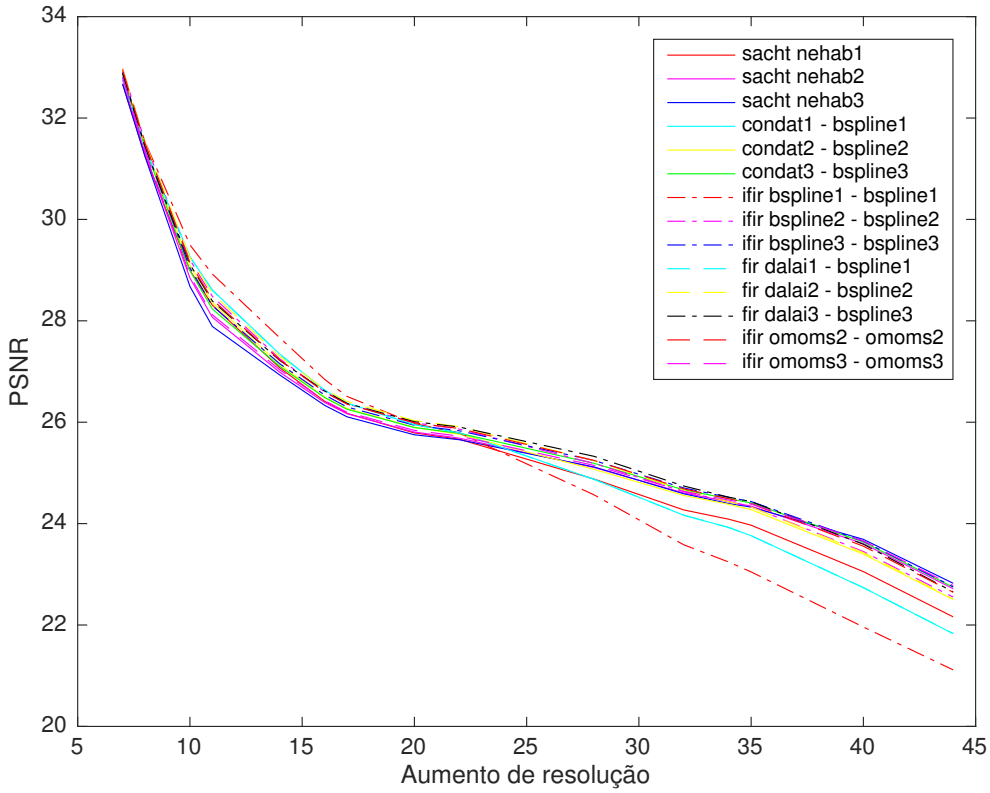


Figura 3.3.9 – Gráfico PSNR de reconstrução do som de uma guitarra.

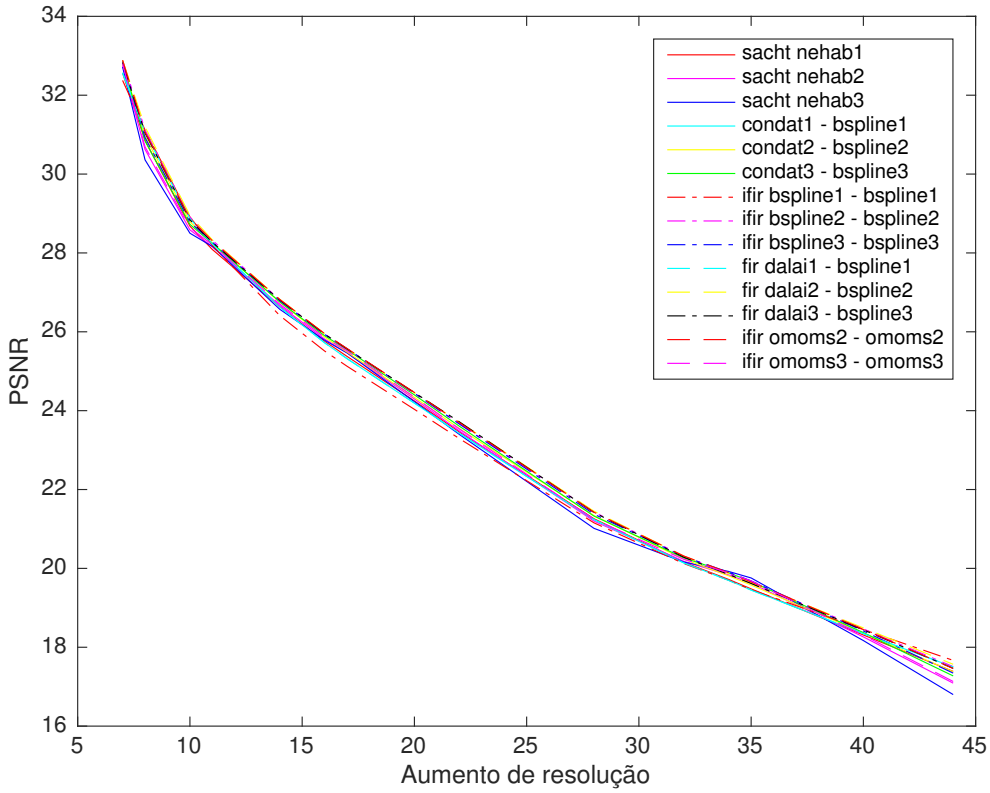


Figura 3.3.10 – Gráfico PSNR de reconstrução do som de uma sitar indiana.

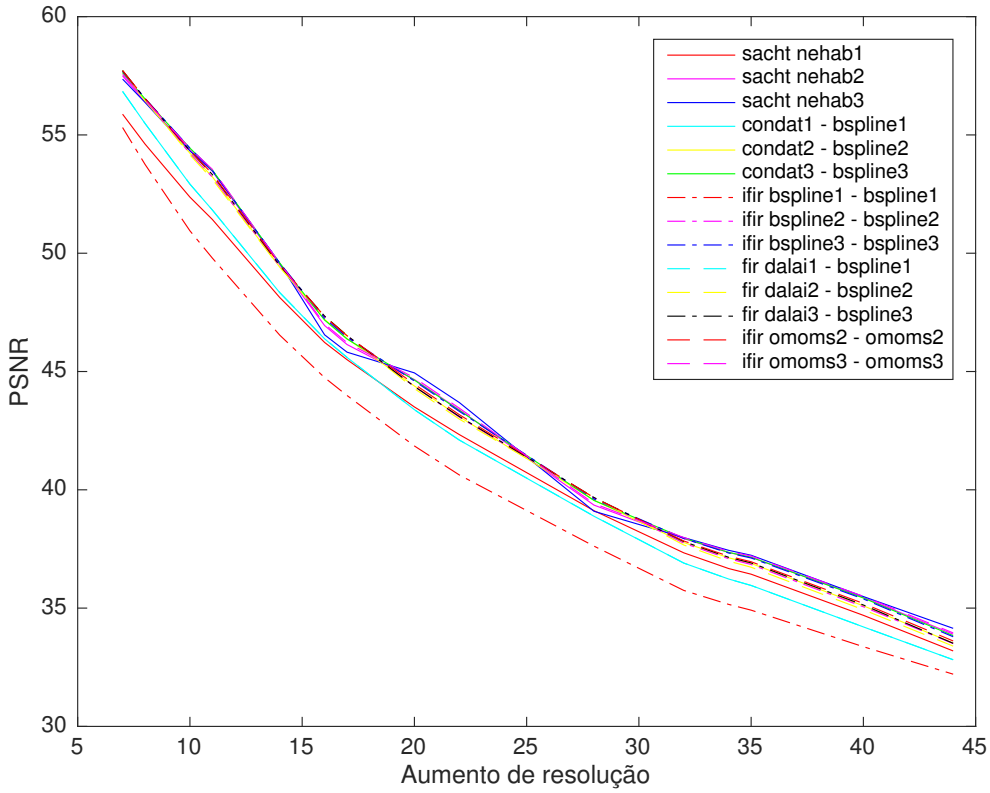


Figura 3.3.11 – Gráfico PSNR de reconstrução do som de um violão.

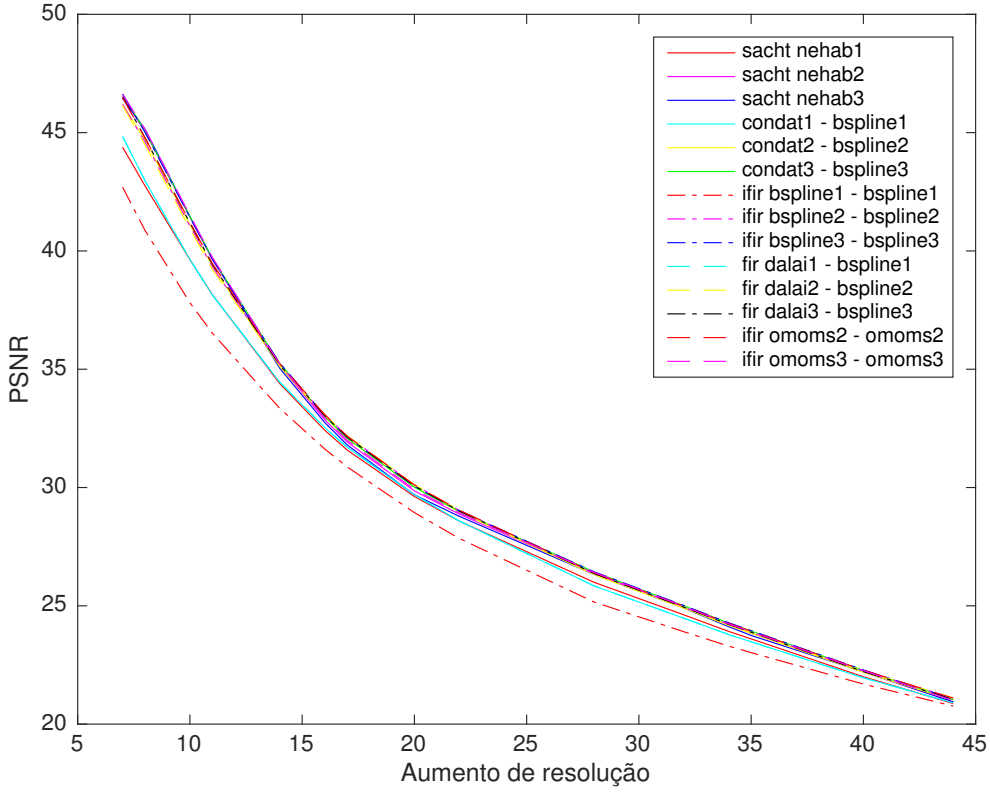


Figura 3.3.12 – Gráfico PSNR de reconstrução do som de um violoncelo sintético.

APÊNDICE A – Scripts

A.1 PSNRSOUND

```
clear;
[original, freqs] = audioread('violao44100.wav');
original = original(1:209441,:);
if size(original,2) == 2
    original = original(:,1);
end
k = 50;
MAX = 1;% MAX_I for signal-to-quantization- noise-ratio (SQNR) em dB
dixx = [];
PSNR1 = [];
PSNR2 = [];
PSNR3 = [];
PSNR4 = [];
PSNR5 = [];
PSNR6 = [];
PSNR7 = [];
PSNR8 = [];
PSNR9 = [];
PSNR10 = [];
PSNR11 = [];
PSNR12 = [];
PSNR13 = [];
PSNR14 = [];
for n = 6:k
    if mod(size(original,1)-1,n) == 0
        dixx = [dixx n];
        x = zeros(floor(length(original)/n),1);
        for i = 1:length(x)
            x(i) = original((i-1)*n + 1);
        end
        [X1,F,x,f] = upsignal(x,freqs/n,'ifir_sacht_nehab1',...
            'sacht_nehab1',n);
        [X2,F,x,f] = upsignal(x,freqs/n,'ifir_sacht_nehab2',...
            'sacht_nehab2',n);
        [X3,F,x,f] = upsignal(x,freqs/n,'ifir_sacht_nehab3',...
            'sacht_nehab3',n);
        [X4,F,x,f] = upsignal(x,freqs/n,'ifir_condat1','bspline1',n);
        [X5,F,x,f] = upsignal(x,freqs/n,'ifir_condat2','bspline2',n);
```

```
[X6,F,x,f] = upsignal(x,freqs/n,'ifir_condat3','bspline3',n);
[X7,F,x,f] = upsignal(x,freqs/n,'ifir_bspline1','bspline1',n);
[X8,F,x,f] = upsignal(x,freqs/n,'ifir_bspline2','bspline2',n);
[X9,F,x,f] = upsignal(x,freqs/n,'ifir_bspline3','bspline3',n);
[X10,F,x,f] = upsignal(x,freqs/n,'fir_dalai1','bspline1',n);
[X11,F,x,f] = upsignal(x,freqs/n,'fir_dalai2','bspline2',n);
[X12,F,x,f] = upsignal(x,freqs/n,'fir_dalai3','bspline3',n);
[X13,F,x,f] = upsignal(x,freqs/n,'ifir_omoms2','omoms2',n);
[X14,F,x,f] = upsignal(x,freqs/n,'ifir_omoms3','omoms3',n);
original1 = original(1:size(X1,1));
vMSE1 = zeros(size(X1,1),1);
for i = 1:size(X1)
vMSE1(i) = (1/size(X1,1))*((original1(i) - X1(i))^2);
end
MSE1 = sum(vMSE1);
PSNR1 = [ PSNR1 10*log10((MAX^2)/MSE1)];
%%%%%%%%%%%%%
%%%%%%%%%%%%%
original2 = original(1:size(X2,1));
vMSE2 = zeros(size(X2,1),1);
for i = 1:size(X2)
vMSE2(i) = (1/size(X2,1))*((original2(i) - X2(i))^2);
end
MSE2 = sum(vMSE2);
PSNR2 = [ PSNR2 10*log10((MAX^2)/MSE2)];
%%%%%%%%%%%%%
%%%%%%%%%%%%%
original3 = original(1:size(X3,1));
vMSE3 = zeros(size(X3,1),1);
for i = 1:size(X3)
vMSE3(i) = (1/size(X3,1))*((original3(i) - X3(i))^2);
end
MSE3 = sum(vMSE3);
PSNR3 = [ PSNR3 10*log10((MAX^2)/MSE3)];
%%%%%%%%%%%%%
%%%%%%%%%%%%%
original4 = original(1:size(X4,1));
vMSE4 = zeros(size(X4,1),1);
for i = 1:size(X4)
vMSE4(i) = (1/size(X4,1))*((original4(i) - X4(i))^2);
end
MSE4 = sum(vMSE4);
PSNR4 = [ PSNR4 10*log10((MAX^2)/MSE4)];
%%%%%%%%%%%%%
%%%%%%%%%%%%%
original5 = original(1:size(X5,1));
vMSE5 = zeros(size(X5,1),1);
```

```
for i = 1:size(X5)
vMSE5(i) = (1/size(X5,1))*((original5(i) - X5(i))^2);
end
MSE5 = sum(vMSE5);
PSNR5 = [ PSNR5 10*log10((MAX^2)/MSE5)];
%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
original6 = original(1:size(X6,1));
vMSE6 = zeros(size(X6,1),1);
for i = 1:size(X6)
vMSE6(i) = (1/size(X6,1))*((original6(i) - X6(i))^2);
end
MSE6 = sum(vMSE6);
PSNR6 = [ PSNR6 10*log10((MAX^2)/MSE6)];
%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
original7 = original(1:size(X7,1));
vMSE7 = zeros(size(X7,1),1);
for i = 1:size(X7)
vMSE7(i) = (1/size(X7,1))*((original7(i) - X7(i))^2);
end
MSE7 = sum(vMSE7);
PSNR7 = [ PSNR7 10*log10((MAX^2)/MSE7)];
%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
original8 = original(1:size(X8,1));
vMSE8 = zeros(size(X8,1),1);
for i = 1:size(X8)
vMSE8(i) = (1/size(X8,1))*((original8(i) - X8(i))^2);
end
MSE8 = sum(vMSE8);
PSNR8 = [ PSNR8 10*log10((MAX^2)/MSE8)];
%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
original9 = original(1:size(X9,1));
vMSE9 = zeros(size(X9,1),1);
for i = 1:size(X9)
    vMSE9(i) = (1/size(X9,1))*((original9(i) - X9(i))^2);
end
MSE9 = sum(vMSE9);
PSNR9 = [ PSNR9 10*log10((MAX^2)/MSE9)];
%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
original10 = original(1:size(X10,1));
vMSE10 = zeros(size(X10,1),1);
for i = 1:size(X10)
vMSE10(i) = (1/size(X10,1))*((original10(i) - X10(i))^2);
```

```
        end
        MSE10 = sum(vMSE10);
        PSNR10 = [ PSNR10 10*log10((MAX^2)/MSE10)];
        %%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%
        original11 = original(1:size(X11,1));
        vMSE11 = zeros(size(X11,1),1);
        for i = 1:size(X11)
        vMSE11(i) = (1/size(X11,1))*((original8(i) - X11(i))^2);
        end
        MSE11 = sum(vMSE11);
        PSNR11 = [ PSNR11 10*log10((MAX^2)/MSE11)];
        %%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%
        original12 = original(1:size(X12,1));
        vMSE12 = zeros(size(X12,1),1);
        for i = 1:size(X12)
            vMSE12(i) = (1/size(X12,1))*((original12(i) - X12(i))^2);
        end
        MSE12 = sum(vMSE12);
        PSNR12 = [ PSNR12 10*log10((MAX^2)/MSE12)];
        %%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%
        original13 = original(1:size(X13,1));
        vMSE13 = zeros(size(X13,1),1);
        for i = 1:size(X8)
        vMSE13(i) = (1/size(X13,1))*((original8(i) - X13(i))^2);
        end
        MSE13 = sum(vMSE13);
        PSNR13 = [ PSNR13 10*log10((MAX^2)/MSE13)];
        %%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%
        original14 = original(1:size(X14,1));
        vMSE14 = zeros(size(X14,1),1);
        for i = 1:size(X14)
            vMSE14(i) = (1/size(X14,1))*((original14(i) - X14(i))^2);
        end
        MSE14 = sum(vMSE14);
        PSNR14 = [ PSNR14 10*log10((MAX^2)/MSE14)];
    end
end
    plot(dixx,PSNR1,'r');
    hold on
    plot(dixx,PSNR2,'m');
    hold on
    plot(dixx,PSNR3,'b');
    hold on
```

```
    plot(dixx,PSNR4,'c');
    hold on
    plot(dixx,PSNR5,'y');
    hold on
    plot(dixx,PSNR6,'g');
    hold on
    plot(dixx,PSNR7,'-.r');
    hold on
    plot(dixx,PSNR8,'-.m');
    hold on
    plot(dixx,PSNR9,'-.b');
    hold on
    plot(dixx,PSNR10,'--c');
    hold on
    plot(dixx,PSNR11,'--y');
    hold on
    plot(dixx,PSNR12,'-.k');
    hold on
    plot(dixx,PSNR13,'--r');
    hold on
    plot(dixx,PSNR14,'--m');
    hold on


    ylabel('PSNR'); xlabel('Aumento de resolucao');
    legend('sacht nehab1', 'sacht nehab2', 'sacht nehab3', ...
        'condat1 - bspline1','condat2 - bspline2','condat3 - bspline3',...
        'ifir bspline1 - bspline1','ifir bspline2 - bspline2',...
        'ifir bspline3 - bspline3','fir dalai1 - bspline1',...
        'fir dalai2 - bspline2','fir dalai3 - bspline3',...
        'ifir omoms2 - omoms2','ifir omoms3 - omoms3');
    hold off
```

A.2 SPEC3D

```
function [ ] = spec3D( y, Fs )
%spec3D is a function to generate y signal spectrogram of a audio signal
%and Fs Frequency
if size(y,2) == 1
spectrogram(y,100,80,100,Fs,'yaxis')
view(-77,72); shading interp; colorbar off
else
    y = sum(y,2);
    spectrogram(y,100,80,100,Fs,'yaxis')
view(-77,72); shading interp; colorbar off
end
```

```
end
```

A.3 UPSIGNAL

```
function [nSIGNAL,nFreqs,xSIGNAL,xFreqs] = upsignal(signal,...
                                                    frequency,...
                                                    filter,...
                                                    generator,...
                                                    n)

if ischar(signal) == 1
[xSIGNAL,xFreqs] = audioread(signal);
nFreqs = n*frequency;
else
    xSIGNAL = signal;
    xFreqs = frequency;
    nFreqs = n*frequency;
end
 %Criando o valor de frequencia n-vezes aumentada.

t = 1:1/n:size(xSIGNAL);

fSIGNAL = zeros(size(xSIGNAL));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%FILTROS Finite Impulse Response%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if strcmp(filter,'fir_dalai1')==1
    q0 = 49/40;
    q1 = -11/90;
    q2 = 7/720;
    if size(xSIGNAL,2) == 2
        fSIGNAL(1,1) = q0*xSIGNAL(1,1) + 2*q1*xSIGNAL(2,1) ...
            + 2*q2*xSIGNAL(3,1);
        fSIGNAL(1,2) = q0*xSIGNAL(1,2) + 2*q1*xSIGNAL(2,2) ...
            + 2*q2*xSIGNAL(3,2);
        fSIGNAL(2,1) = q1*xSIGNAL(1,1) + q0*xSIGNAL(2,1) ...
            + q1*xSIGNAL(3,1) + 2*q2*xSIGNAL(4,1);
        fSIGNAL(2,2) = q1*xSIGNAL(1,2) + q0*xSIGNAL(2,2) ...
            + q1*xSIGNAL(3,2) + 2*q2*xSIGNAL(4,2);
        fSIGNAL(size(xSIGNAL,1)-1,1) = 2*q2*xSIGNAL(size(xSIGNAL,1)-3,1)...
            + q1*xSIGNAL(size(xSIGNAL,1)-2,1) ...
            + q0*xSIGNAL(size(xSIGNAL,1)-1,1) ...
            + q1*xSIGNAL(size(xSIGNAL,1),1);
        fSIGNAL(size(xSIGNAL,1)-1,2) = 2*q2*xSIGNAL(size(xSIGNAL,1)-3,2)...
```

```
            + q1*xSIGNAL(size(xSIGNAL,1)-2,2) ...
            + q0*xSIGNAL(size(xSIGNAL,1)-1,2) ...
            + q1*xSIGNAL(size(xSIGNAL,1),2);
        fSIGNAL(size(xSIGNAL,1),1) = 2*q2*xSIGNAL(size(xSIGNAL,1)-2,1)...
            + 2*q1*xSIGNAL(size(xSIGNAL,1)-1,1)...
            + q0*xSIGNAL(size(xSIGNAL,1),1);
        fSIGNAL(size(xSIGNAL,1),2) = 2*q2*xSIGNAL(size(xSIGNAL,1)-2,2)...
            + 2*q1*xSIGNAL(size(xSIGNAL,1)-1,2)...
            + q0*xSIGNAL(size(xSIGNAL,1),1);
        for i = 3:(size(fSIGNAL)-2)
            fSIGNAL(i,1) = q2*xSIGNAL(i-2,1) + q1*xSIGNAL(i-1,1)...
                + q0*xSIGNAL(i,1) + q1*xSIGNAL(i+1,1) + q2*xSIGNAL(i+2,1);
            fSIGNAL(i,2) = q2*xSIGNAL(i-2,2) + q1*xSIGNAL(i-1,2)...
                + q0*xSIGNAL(i,2) + q1*xSIGNAL(i+1,2) + q2*xSIGNAL(i+2,2);
        end
    else
        fSIGNAL(1) = q0*xSIGNAL(1) + 2*q1*xSIGNAL(2) + 2*q2*xSIGNAL(3);
        fSIGNAL(2) = q1*xSIGNAL(1) + q0*xSIGNAL(2) + q1*xSIGNAL(3)...
            + 2*q2*xSIGNAL(4);
        fSIGNAL(size(xSIGNAL,1)-1) = 2*q2*xSIGNAL(size(xSIGNAL,1)-3)...
            + q1*xSIGNAL(size(xSIGNAL,1)-2)...
            + q0*xSIGNAL(size(xSIGNAL,1)-1)...
            + q1*xSIGNAL(size(xSIGNAL,1));
        fSIGNAL(size(xSIGNAL,1)) = 2*q2*xSIGNAL(size(xSIGNAL,1)-2)...
            + 2*q1*xSIGNAL(size(xSIGNAL,1)-1)...
            + q0*xSIGNAL(size(xSIGNAL,1));
        for i = 3:(size(fSIGNAL)-2)
            fSIGNAL(i,1) = q2*xSIGNAL(i-2,1)...
                + q1*xSIGNAL(i-1) + q0*xSIGNAL(i)...
                + q1*xSIGNAL(i+1) + q2*xSIGNAL(i+2);
        end
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(filter,'fir_dalai2')==1
    q0 = 437/320;
    q1 = -97/480;
    q2 = 37/1920;
    if size(xSIGNAL,2) == 2
        fSIGNAL(1,1) = q0*xSIGNAL(1,1) + 2*q1*xSIGNAL(2,1)...
            + 2*q2*xSIGNAL(3,1);
        fSIGNAL(1,2) = q0*xSIGNAL(1,2) + 2*q1*xSIGNAL(2,2)...
            + 2*q2*xSIGNAL(3,2);
        fSIGNAL(2,1) = q1*xSIGNAL(1,1) + q0*xSIGNAL(2,1)...
            + q1*xSIGNAL(3,1) + 2*q2*xSIGNAL(4,1);
        fSIGNAL(2,2) = q1*xSIGNAL(1,2) + q0*xSIGNAL(2,2)...
            + q1*xSIGNAL(3,2) + 2*q2*xSIGNAL(4,2);
        fSIGNAL(size(xSIGNAL,1)-1,1) = 2*q2*xSIGNAL(size(xSIGNAL,1)-3,1)...
```

```
            + q1*xSIGNAL(size(xSIGNAL,1)-2,1)...
            + q0*xSIGNAL(size(xSIGNAL,1)-1,1)...
            + q1*xSIGNAL(size(xSIGNAL,1),1);
        fSIGNAL(size(xSIGNAL,1)-1,2) = 2*q2*xSIGNAL(size(xSIGNAL,1)-3,2)...
            + q1*xSIGNAL(size(xSIGNAL,1)-2,2)...
            + q0*xSIGNAL(size(xSIGNAL,1)-1,2)...
            + q1*xSIGNAL(size(xSIGNAL,1),2);
        fSIGNAL(size(xSIGNAL,1),1) = 2*q2*xSIGNAL(size(xSIGNAL,1)-2,1)...
            + 2*q1*xSIGNAL(size(xSIGNAL,1)-1,1)...
            + q0*xSIGNAL(size(xSIGNAL,1),1);
        fSIGNAL(size(xSIGNAL,1),2) = 2*q2*xSIGNAL(size(xSIGNAL,1)-2,2)...
            + 2*q1*xSIGNAL(size(xSIGNAL,1)-1,2)...
            + q0*xSIGNAL(size(xSIGNAL,1),1);
        for i = 3:(size(fSIGNAL)-2)
            fSIGNAL(i,1) = q2*xSIGNAL(i-2,1) + q1*xSIGNAL(i-1,1)...
                + q0*xSIGNAL(i,1) + q1*xSIGNAL(i+1,1) + q2*xSIGNAL(i+2,1);
            fSIGNAL(i,2) = q2*xSIGNAL(i-2,2) + q1*xSIGNAL(i-1,2)...
                + q0*xSIGNAL(i,2) + q1*xSIGNAL(i+1,2) + q2*xSIGNAL(i+2,2);
        end
    else
        fSIGNAL(1) = q0*xSIGNAL(1) + 2*q1*xSIGNAL(2) + 2*q2*xSIGNAL(3);
        fSIGNAL(2) = q1*xSIGNAL(1) + q0*xSIGNAL(2) + q1*xSIGNAL(3)...
            + 2*q2*xSIGNAL(4);
        fSIGNAL(size(xSIGNAL,1)-1) = 2*q2*xSIGNAL(size(xSIGNAL,1)-3)...
            + q1*xSIGNAL(size(xSIGNAL,1)-2)...
            + q0*xSIGNAL(size(xSIGNAL,1)-1) + q1*xSIGNAL(size(xSIGNAL,1));
        fSIGNAL(size(xSIGNAL,1)) = 2*q2*xSIGNAL(size(xSIGNAL,1)-2)...
            + 2*q1*xSIGNAL(size(xSIGNAL,1)-1)...
            + q0*xSIGNAL(size(xSIGNAL,1));
        for i = 3:(size(fSIGNAL)-2)
            fSIGNAL(i) = q2*xSIGNAL(i-2) + q1*xSIGNAL(i-1)...
                + q0*xSIGNAL(i) + q1*xSIGNAL(i+1) + q2*xSIGNAL(i+2);
        end
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(filter,'fir_dalai3')==1
    q0 = 12223/7560;
    q1 = -919/2520;
    q2 = 311/5040;
    q3 = -41/7560;
    if size(xSIGNAL,2) == 2
        fSIGNAL(1,1) = q0*xSIGNAL(1,1) + 2*q1*xSIGNAL(2,1)...
            + 2*q2*xSIGNAL(3,1) + 2*q3*xSIGNAL(4,1);
        fSIGNAL(1,2) = q0*xSIGNAL(1,2) + 2*q1*xSIGNAL(2,2)...
            + 2*q2*xSIGNAL(3,2) + 2*q3*xSIGNAL(4,2);
        fSIGNAL(2,1) = q1*xSIGNAL(1,1) + q0*xSIGNAL(2,1)...
            + q1*xSIGNAL(3,1) + 2*q2*xSIGNAL(4,1) + 2*q3*xSIGNAL(5,1);
```

```
fSIGNAL(2,2) = q1*xSIGNAL(1,2) + q0*xSIGNAL(2,2)...
    + q1*xSIGNAL(3,2) + 2*q2*xSIGNAL(4,2) + 2*q3*xSIGNAL(5,2);
fSIGNAL(3,1) = q2*xSIGNAL(1,1) + q1*xSIGNAL(2,1)...
    + q0*xSIGNAL(3,1) + q1*xSIGNAL(4,1)...
    + q2*xSIGNAL(5,1) + 2*q3*xSIGNAL(6,1);
fSIGNAL(3,2) = q2*xSIGNAL(1,2) + q1*xSIGNAL(2,2)...
    + q0*xSIGNAL(3,2) + q1*xSIGNAL(4,2)...
    + q2*xSIGNAL(5,2) + 2*q3*xSIGNAL(6,2);
fSIGNAL(size(xSIGNAL,1)-2,1) = 2*q3*xSIGNAL(size(xSIGNAL,1)-5,1)...
    + q2*xSIGNAL(size(xSIGNAL,1)-4,1)...
    + q1*xSIGNAL(size(xSIGNAL,1)-3,1)...
    + q0*xSIGNAL(size(xSIGNAL,1)-2,1)...
    + q1*xSIGNAL(size(xSIGNAL,1)-1,1)...
    + q2*xSIGNAL(size(xSIGNAL,1),1);
fSIGNAL(size(xSIGNAL,1)-2,2) = 2*q3*xSIGNAL(size(xSIGNAL,1)-5,2)...
    + q2*xSIGNAL(size(xSIGNAL,1)-4,2)...
    + q1*xSIGNAL(size(xSIGNAL,1)-3,2)...
    + q0*xSIGNAL(size(xSIGNAL,1)-2,2)...
    + q1*xSIGNAL(size(xSIGNAL,1)-1,2)...
    + q2*xSIGNAL(size(xSIGNAL,1),2);
fSIGNAL(size(xSIGNAL,1)-1,1) = 2*q3*xSIGNAL(size(xSIGNAL,1)-4,1)...
    + 2*q2*xSIGNAL(size(xSIGNAL,1)-3,1)...
    + q1*xSIGNAL(size(xSIGNAL,1)-2,1)...
    + q0*xSIGNAL(size(xSIGNAL,1)-1,1)...
    + q1*xSIGNAL(size(xSIGNAL,1),1);
fSIGNAL(size(xSIGNAL,1)-1,2) = 2*q3*xSIGNAL(size(xSIGNAL,1)-4,2)...
    + 2*q2*xSIGNAL(size(xSIGNAL,1)-3,2)...
    + q1*xSIGNAL(size(xSIGNAL,1)-2,2)...
    + q0*xSIGNAL(size(xSIGNAL,1)-1,2)...
    + q1*xSIGNAL(size(xSIGNAL,1),2);
fSIGNAL(size(xSIGNAL,1),1) = 2*q3*xSIGNAL(size(xSIGNAL,1)-3,1)...
    + 2*q2*xSIGNAL(size(xSIGNAL,1)-2,1)...
    + 2*q1*xSIGNAL(size(xSIGNAL,1)-1,1)...
    + q0*xSIGNAL(size(xSIGNAL,1),1);
fSIGNAL(size(xSIGNAL,1),2) = 2*q3*xSIGNAL(size(xSIGNAL,1)-3,2)...
    + 2*q2*xSIGNAL(size(xSIGNAL,1)-2,2)...
    + 2*q1*xSIGNAL(size(xSIGNAL,1)-1,2)...
    + q0*xSIGNAL(size(xSIGNAL,1),2);
for i = 4:(size(fSIGNAL)-3)
    fSIGNAL(i,1) = q3*xSIGNAL(i-3,1) + q2*xSIGNAL(i-2,1)...
        + q1*xSIGNAL(i-1,1) + q0*xSIGNAL(i,1)...
        + q1*xSIGNAL(i+1,1) + q2*xSIGNAL(i+2,1)...
        + q3*xSIGNAL(i+3,1);
    fSIGNAL(i,2) = q3*xSIGNAL(i-3,2) + q2*xSIGNAL(i-2,2)...
        + q1*xSIGNAL(i-1,2) + q0*xSIGNAL(i,2)...
        + q1*xSIGNAL(i+1,2) + q2*xSIGNAL(i+2,2)...
        + q3*xSIGNAL(i+3,2);
```

```
        end
    else
        fSIGNAL(1) = q0*xSIGNAL(1) + 2*q1*xSIGNAL(2) + 2*q2*xSIGNAL(3)...
            + 2*q3*xSIGNAL(4);
        fSIGNAL(2) = q1*xSIGNAL(1) + q0*xSIGNAL(2) + q1*xSIGNAL(3)...
            + 2*q2*xSIGNAL(4) + 2*q3*xSIGNAL(5);
        fSIGNAL(3) = q2*xSIGNAL(1) + q1*xSIGNAL(2) + q0*xSIGNAL(3)...
            + q1*xSIGNAL(4) + q2*xSIGNAL(5) + 2*q3*xSIGNAL(6);
        fSIGNAL(size(xSIGNAL,1)-2) = 2*q3*xSIGNAL(size(xSIGNAL,1)-5)...
            + q2*xSIGNAL(size(xSIGNAL,1)-4)...
            + q1*xSIGNAL(size(xSIGNAL,1)-3)...
            + q0*xSIGNAL(size(xSIGNAL,1)-2)...
            + q1*xSIGNAL(size(xSIGNAL,1)-1)...
            + q2*xSIGNAL(size(xSIGNAL,1));
        fSIGNAL(size(xSIGNAL,1)-1) = 2*q3*xSIGNAL(size(xSIGNAL,1)-4)...
            + 2*q2*xSIGNAL(size(xSIGNAL,1)-3)...
            + q1*xSIGNAL(size(xSIGNAL,1)-2)...
            + q0*xSIGNAL(size(xSIGNAL,1)-1)...
            + q1*xSIGNAL(size(xSIGNAL,1));
        fSIGNAL(size(xSIGNAL,1)) = 2*q3*xSIGNAL(size(xSIGNAL,1)-3)...
            + 2*q2*xSIGNAL(size(xSIGNAL,1)-2)...
            + 2*q1*xSIGNAL(size(xSIGNAL,1)-1)...
            + q0*xSIGNAL(size(xSIGNAL,1));
        for i = 4:(size(fSIGNAL)-3)
            fSIGNAL(i) = q3*xSIGNAL(i-3) + q2*xSIGNAL(i-2)...
                + q1*xSIGNAL(i-1) + q0*xSIGNAL(i) + q1*xSIGNAL(i+1)...
                + q2*xSIGNAL(i+2) + q3*xSIGNAL(i+3);
        end
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(filter,'fir_blu35')==1
    q0 = 8.76565;
    q1 = 4.65992;
    q2 = 0.621773;
    q3 = 0.012285;
    q4 = 1.875*10^(-6);
    if size(xSIGNAL,2) == 2
        fSIGNAL(1,1) = q0*xSIGNAL(1,1) + 2*q1*xSIGNAL(2,1)...
            + 2*q2*xSIGNAL(3,1) + 2*q3*xSIGNAL(4,1) + 2*q4*xSIGNAL(5,1);
        fSIGNAL(1,2) = q0*xSIGNAL(1,2) + 2*q1*xSIGNAL(2,2)...
            + 2*q2*xSIGNAL(3,2) + 2*q3*xSIGNAL(4,2) + 2*q4*xSIGNAL(5,2);
        fSIGNAL(2,1) = q1*xSIGNAL(1,1) + q0*xSIGNAL(2,1)...
            + q1*xSIGNAL(3,1) + 2*q2*xSIGNAL(4,1) + 2*q3*xSIGNAL(5,1)...
            + 2*q4*xSIGNAL(6,1);
        fSIGNAL(2,2) = q1*xSIGNAL(1,2) + q0*xSIGNAL(2,2)...
            + q1*xSIGNAL(3,2) + 2*q2*xSIGNAL(4,2)...
            + 2*q3*xSIGNAL(5,2) + 2*q4*xSIGNAL(6,2);
```

```
fSIGNAL(3,1) = q2*xSIGNAL(1,1) + q1*xSIGNAL(2,1)...
    + q0*xSIGNAL(3,1) + q1*xSIGNAL(4,1) + q2*xSIGNAL(5,1)...
    + 2*q3*xSIGNAL(6,1) + 2*q4*xSIGNAL(7,1);
fSIGNAL(3,2) = q2*xSIGNAL(1,2) + q1*xSIGNAL(2,2)...
    + q0*xSIGNAL(3,2) + q1*xSIGNAL(4,2) + q2*xSIGNAL(5,2)...
    + 2*q3*xSIGNAL(6,2) + 2*q4*xSIGNAL(7,2);
fSIGNAL(4,1) = q3*xSIGNAL(1,1) + q2*xSIGNAL(2,1)...
    + q1*xSIGNAL(3,1) + q0*xSIGNAL(4,1) + q1*xSIGNAL(5,1)...
    + q2*xSIGNAL(6,1) + q3*xSIGNAL(7,1) + 2*q4*xSIGNAL(8,1);
fSIGNAL(4,2) = q3*xSIGNAL(1,2) + q2*xSIGNAL(2,2)...
    + q1*xSIGNAL(3,2) + q0*xSIGNAL(4,2) + q1*xSIGNAL(5,2)...
    + q2*xSIGNAL(6,2) + q3*xSIGNAL(7,2) + 2*q4*xSIGNAL(8,2);
fSIGNAL(size(xSIGNAL,1)-3,1) = 2*q4*xSIGNAL(size(xSIGNAL,1)-7,1)...
    + q3*xSIGNAL(size(xSIGNAL,1)-6,1)...
    + q2*xSIGNAL(size(xSIGNAL,1)-5,1)...
    + q1*xSIGNAL(size(xSIGNAL,1)-4,1)...
    + q0*xSIGNAL(size(xSIGNAL,1)-3,1)...
    + q1*xSIGNAL(size(xSIGNAL,1)-2,1)...
    + q2*xSIGNAL(size(xSIGNAL,1)-1,1)...
    + q3*xSIGNAL(size(xSIGNAL,1),1);
fSIGNAL(size(xSIGNAL,1)-3,2) = 2*q4*xSIGNAL(size(xSIGNAL,1)-7,2)...
    + q3*xSIGNAL(size(xSIGNAL,1)-6,2)...
    + q2*xSIGNAL(size(xSIGNAL,1)-5,2)...
    + q1*xSIGNAL(size(xSIGNAL,1)-4,2)...
    + q0*xSIGNAL(size(xSIGNAL,1)-3,2)...
    + q1*xSIGNAL(size(xSIGNAL,1)-2,2)...
    + q2*xSIGNAL(size(xSIGNAL,1)-1,2)...
    + q3*xSIGNAL(size(xSIGNAL,1),2);
fSIGNAL(size(xSIGNAL,1)-2,1) = 2*q4*xSIGNAL(size(xSIGNAL,1)-6,1)...
    + 2*q3*xSIGNAL(size(xSIGNAL,1)-5,1)...
    + q2*xSIGNAL(size(xSIGNAL,1)-4,1)...
    + q1*xSIGNAL(size(xSIGNAL,1)-3,1)...
    + q0*xSIGNAL(size(xSIGNAL,1)-2,1)...
    + q1*xSIGNAL(size(xSIGNAL,1)-1,1)...
    + q2*xSIGNAL(size(xSIGNAL,1),1);
fSIGNAL(size(xSIGNAL,1)-2,2) = 2*q4*xSIGNAL(size(xSIGNAL,1)-6,2)...
    + 2*q3*xSIGNAL(size(xSIGNAL,1)-5,2)...
    + q2*xSIGNAL(size(xSIGNAL,1)-4,2)...
    + q1*xSIGNAL(size(xSIGNAL,1)-3,2)...
    + q0*xSIGNAL(size(xSIGNAL,1)-2,2)...
    + q1*xSIGNAL(size(xSIGNAL,1)-1,2)...
    + q2*xSIGNAL(size(xSIGNAL,1),2);
fSIGNAL(size(xSIGNAL,1)-1,1) = 2*q4*xSIGNAL(size(xSIGNAL,1)-5,1)...
    + 2*q3*xSIGNAL(size(xSIGNAL,1)-4,1)...
    + 2*q2*xSIGNAL(size(xSIGNAL,1)-3,1)...
    + q1*xSIGNAL(size(xSIGNAL,1)-2,1)...
    + q0*xSIGNAL(size(xSIGNAL,1)-1,1)...
```

```
            + q1*xSIGNAL(size(xSIGNAL,1),1);
        fSIGNAL(size(xSIGNAL,1)-1,2) = 2*q4*xSIGNAL(size(xSIGNAL,1)-5,2)...
            + 2*q3*xSIGNAL(size(xSIGNAL,1)-4,2)...
            + 2*q2*xSIGNAL(size(xSIGNAL,1)-3,2)...
            + q1*xSIGNAL(size(xSIGNAL,1)-2,2)...
            + q0*xSIGNAL(size(xSIGNAL,1)-1,2)...
            + q1*xSIGNAL(size(xSIGNAL,1),2);
        fSIGNAL(size(xSIGNAL,1),1) = 2*q4*xSIGNAL(size(xSIGNAL,1)-4,1)...
            + 2*q3*xSIGNAL(size(xSIGNAL,1)-3,1)...
            + 2*q2*xSIGNAL(size(xSIGNAL,1)-2,1)...
            + 2*q1*xSIGNAL(size(xSIGNAL,1)-1,1)...
            + q0*xSIGNAL(size(xSIGNAL,1),1);
        fSIGNAL(size(xSIGNAL,1),2) = 2*q4*xSIGNAL(size(xSIGNAL,1)-4,2)...
            + 2*q3*xSIGNAL(size(xSIGNAL,1)-3,2)...
            + 2*q2*xSIGNAL(size(xSIGNAL,1)-2,2)...
            + 2*q1*xSIGNAL(size(xSIGNAL,1)-1,2)...
            + q0*xSIGNAL(size(xSIGNAL,1),2);
        for i = 5:(size(xSIGNAL)-4)
            fSIGNAL(i,1) = q4*xSIGNAL(i-4,1) + q3*xSIGNAL(i-3,1)...
                + q2*xSIGNAL(i-2,1) + q1*xSIGNAL(i-1,1)...
                + q0*xSIGNAL(i,1) + q1*xSIGNAL(i+1,1)...
                + q2*xSIGNAL(i+2,1) + q3*xSIGNAL(i+3,1)...
                + q4*xSIGNAL(i+4,1);
            fSIGNAL(i,2) = q4*xSIGNAL(i-4,2) + q3*xSIGNAL(i-3,2)...
                + q2*xSIGNAL(i-2,2) + q1*xSIGNAL(i-1,2)...
                + q0*xSIGNAL(i,2) + q1*xSIGNAL(i+1,2)...
                + q2*xSIGNAL(i+2,2) + q3*xSIGNAL(i+3,2)...
                + q4*xSIGNAL(i+4,2);
        end
    else
        fSIGNAL(1) = q0*xSIGNAL(1) + 2*q1*xSIGNAL(2) + 2*q2*xSIGNAL(3)...
            + 2*q3*xSIGNAL(4) + 2*q4*xSIGNAL(5);
        fSIGNAL(2) = q1*xSIGNAL(1) + q0*xSIGNAL(2) + q1*xSIGNAL(3)...
            + 2*q2*xSIGNAL(4) + 2*q3*xSIGNAL(5) + 2*q4*xSIGNAL(6);
        fSIGNAL(3) = q2*xSIGNAL(1) + q1*xSIGNAL(2) + q0*xSIGNAL(3)...
            + q1*xSIGNAL(4) + q2*xSIGNAL(5)...
            + 2*q3*xSIGNAL(6) + 2*q4*xSIGNAL(7);
        fSIGNAL(4) = q3*xSIGNAL(1) + q2*xSIGNAL(2) + q1*xSIGNAL(3)...
            + q0*xSIGNAL(4) + q1*xSIGNAL(5) + q2*xSIGNAL(6)...
            + q3*xSIGNAL(7) + 2*q4*xSIGNAL(8);
        fSIGNAL(size(xSIGNAL,1)-3) = 2*q4*xSIGNAL(size(xSIGNAL,1)-7)...
            + q3*xSIGNAL(size(xSIGNAL,1)-6)...
            + q2*xSIGNAL(size(xSIGNAL,1)-5)...
            + q1*xSIGNAL(size(xSIGNAL,1)-4)...
            + q0*xSIGNAL(size(xSIGNAL,1)-3)...
            + q1*xSIGNAL(size(xSIGNAL,1)-2)...
            + q2*xSIGNAL(size(xSIGNAL,1)-1)...
```

```
            + q3*xSIGNAL(size(xSIGNAL,1));
        fSIGNAL(size(xSIGNAL,1)-2) = 2*q4*xSIGNAL(size(xSIGNAL,1)-6)...
            + 2*q3*xSIGNAL(size(xSIGNAL,1)-5)...
            + q2*xSIGNAL(size(xSIGNAL,1)-4)...
            + q1*xSIGNAL(size(xSIGNAL,1)-3)...
            + q0*xSIGNAL(size(xSIGNAL,1)-2)...
            + q1*xSIGNAL(size(xSIGNAL,1)-1)...
            + q2*xSIGNAL(size(xSIGNAL,1));
        fSIGNAL(size(xSIGNAL,1)-1) = 2*q4*xSIGNAL(size(xSIGNAL,1)-5)...
            + 2*q3*xSIGNAL(size(xSIGNAL,1)-4)...
            + 2*q2*xSIGNAL(size(xSIGNAL,1)-3)...
            + q1*xSIGNAL(size(xSIGNAL,1)-2)...
            + q0*xSIGNAL(size(xSIGNAL,1)-1)...
            + q1*xSIGNAL(size(xSIGNAL,1));
        fSIGNAL(size(xSIGNAL,1)) = 2*q4*xSIGNAL(size(xSIGNAL,1)-4)...
            + 2*q3*xSIGNAL(size(xSIGNAL,1)-3)...
            + 2*q2*xSIGNAL(size(xSIGNAL,1)-2)...
            + 2*q1*xSIGNAL(size(xSIGNAL,1)-1)...
            + q0*xSIGNAL(size(xSIGNAL,1));
        for i = 5:(size(xSIGNAL)-4)
            fSIGNAL(i) = q4*xSIGNAL(i-4) + q3*xSIGNAL(i-3)...
                + q2*xSIGNAL(i-2) + q1*xSIGNAL(i-1)...
                + q0*xSIGNAL(i) + q1*xSIGNAL(i+1)...
                + q2*xSIGNAL(i+2) + q3*xSIGNAL(i+3) + q4*xSIGNAL(i+4);
        end
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(filter,'mcool')==1
    q0 = 1/2;
    q1 = 2/9;
    q2 = 1/36;
    if size(xSIGNAL,2) == 2
        fSIGNAL(1,1) = q0*xSIGNAL(1,1) + 2*q1*xSIGNAL(2,1)...
            + 2*q2*xSIGNAL(3,1);
        fSIGNAL(1,2) = q0*xSIGNAL(1,2) + 2*q1*xSIGNAL(2,2)...
            + 2*q2*xSIGNAL(3,2);
        fSIGNAL(2,1) = q1*xSIGNAL(1,1) + q0*xSIGNAL(2,1)...
            + q1*xSIGNAL(3,1) + 2*q2*xSIGNAL(4,1);
        fSIGNAL(2,2) = q1*xSIGNAL(1,2) + q0*xSIGNAL(2,2)...
            + q1*xSIGNAL(3,2) + 2*q2*xSIGNAL(4,2);
        fSIGNAL(size(xSIGNAL,1)-1,1) = 2*q2*xSIGNAL(size(xSIGNAL,1)-3,1)...
            + q1*xSIGNAL(size(xSIGNAL,1)-2,1)...
            + q0*xSIGNAL(size(xSIGNAL,1)-1,1)...
            + q1*xSIGNAL(size(xSIGNAL,1),1);
        fSIGNAL(size(xSIGNAL,1)-1,2) = 2*q2*xSIGNAL(size(xSIGNAL,1)-3,2)...
            + q1*xSIGNAL(size(xSIGNAL,1)-2,2)...
            + q0*xSIGNAL(size(xSIGNAL,1)-1,2)...
```

```
            + q1*xSIGNAL(size(xSIGNAL,1),2);
        fSIGNAL(size(xSIGNAL,1),1) = 2*q2*xSIGNAL(size(xSIGNAL,1)-2,1)...
            + 2*q1*xSIGNAL(size(xSIGNAL,1)-1,1)...
            + q0*xSIGNAL(size(xSIGNAL,1),1);
        fSIGNAL(size(xSIGNAL,1),2) = 2*q2*xSIGNAL(size(xSIGNAL,1)-2,2)...
            + 2*q1*xSIGNAL(size(xSIGNAL,1)-1,2)...
            + q0*xSIGNAL(size(xSIGNAL,1),1);
        for i = 3:(size(fSIGNAL)-2)
            fSIGNAL(i,1) = q2*xSIGNAL(i-2,1) + q1*xSIGNAL(i-1,1)...
                + q0*xSIGNAL(i,1) + q1*xSIGNAL(i+1,1) + q2*xSIGNAL(i+2,1);
            fSIGNAL(i,2) = q2*xSIGNAL(i-2,2) + q1*xSIGNAL(i-1,2)...
                + q0*xSIGNAL(i,2) + q1*xSIGNAL(i+1,2) + q2*xSIGNAL(i+2,2);
        end
    else
        fSIGNAL(1) = q0*xSIGNAL(1) + 2*q1*xSIGNAL(2) + 2*q2*xSIGNAL(3);
        fSIGNAL(2) = q1*xSIGNAL(1) + q0*xSIGNAL(2) + q1*xSIGNAL(3)...
            + 2*q2*xSIGNAL(4);
        fSIGNAL(size(xSIGNAL,1)-1) = 2*q2*xSIGNAL(size(xSIGNAL,1)-3)...
            + q1*xSIGNAL(size(xSIGNAL,1)-2)...
            + q0*xSIGNAL(size(xSIGNAL,1)-1) + q1*xSIGNAL(size(xSIGNAL,1));
        fSIGNAL(size(xSIGNAL,1)) = 2*q2*xSIGNAL(size(xSIGNAL,1)-2)...
            + 2*q1*xSIGNAL(size(xSIGNAL,1)-1)...
            + q0*xSIGNAL(size(xSIGNAL,1));
        for i = 3:(size(fSIGNAL)-2)
            fSIGNAL(i,1) = q2*xSIGNAL(i-2,1) + q1*xSIGNAL(i-1)...
                + q0*xSIGNAL(i) + q1*xSIGNAL(i+1) + q2*xSIGNAL(i+2);
        end
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%FILTROS Infinite Impulse Response%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

elseif strcmp(filter,'ifir_condat0')==1
    %VALORES DO FILTRO%
    q0 = 13/12;
    q1 = -1/24;
    %CRIANDO VETOR COM VALORES DO FILTRO
    mq0 = q0*ones(length(xSIGNAL),1);
    mq1 = q1*ones(length(xSIGNAL)-1,1);
    %CRIANDO MATRIZ SPARSA COM DIAGONAIS DE ACORDO COM VETORES CRIADOS ACIMA
    Q1 = sparse(1:length(xSIGNAL),1:length(xSIGNAL),mq0,...
        length(xSIGNAL),length(xSIGNAL));
    Q2 = sparse(2:length(xSIGNAL),1:(length(xSIGNAL)-1),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q3 = sparse(1:(length(xSIGNAL)-1),2:length(xSIGNAL),mq1,...
        length(xSIGNAL),length(xSIGNAL));
```

```
    Q = Q1 + Q2 + Q3;
    Q(1,2) = 2*Q(1,2);
    Q(length(xSIGNAL),length(xSIGNAL)-1) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-1);
    %FATORACAO LU E APLICACAO EM SOM
    [L,U] = lu(Q);
    Y = L\xSIGNAL;
    fSIGNAL = U\Y;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(filter,'ifir_condat1')==1
    %VALORES DO FILTRO%
    q0 = 5/6;
    q1 = 1/12;
    %CRIANDO VETOR COM VALORES DO FILTRO
    mq0 = q0*ones(length(xSIGNAL),1);
    mq1 = q1*ones(length(xSIGNAL)-1,1);
   %CRIANDO MATRIZ SPARSA COM DIAGONAIS DE ACORDO COM VETORES CRIADOS ACIMA
    Q1 = sparse(1:length(xSIGNAL),1:length(xSIGNAL),mq0,...
        length(xSIGNAL),length(xSIGNAL));
    Q2 = sparse(2:length(xSIGNAL),1:(length(xSIGNAL)-1),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q3 = sparse(1:(length(xSIGNAL)-1),2:length(xSIGNAL),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q = Q1 + Q2 + Q3;
    Q(1,2) = 2*Q(1,2);
    Q(length(xSIGNAL),length(xSIGNAL)-1) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-1);
    %FATORACAO LU E APLICACAO EM SOM
    [L,U] = lu(Q);
    Y = L\xSIGNAL;
    fSIGNAL = U\Y;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(filter,'ifir_condat2')==1
    %VALORES DO FILTRO%
    q0 = 233/320;
    q1 = 67/480;
    q2 = -7/1920;
    %CRIANDO VETOR COM VALORES DO FILTRO
    mq0 = q0*ones(length(xSIGNAL),1);
    mq1 = q1*ones(length(xSIGNAL)-1,1);
    mq2 = q2*ones(length(xSIGNAL)-2,1);
   %CRIANDO MATRIZ SPARSA COM DIAGONAIS DE ACORDO COM VETORES CRIADOS ACIMA
    Q1 = sparse(1:length(xSIGNAL),1:length(xSIGNAL),mq0,...
        length(xSIGNAL),length(xSIGNAL));
    Q2 = sparse(2:length(xSIGNAL),1:(length(xSIGNAL)-1),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q3 = sparse(1:(length(xSIGNAL)-1),2:length(xSIGNAL),mq1,...
```

```
        length(xSIGNAL),length(xSIGNAL));
    Q4 = sparse(1:(length(xSIGNAL)-2),3:length(xSIGNAL),mq2,...
        length(xSIGNAL),length(xSIGNAL));
    Q5 = sparse(3:length(xSIGNAL),1:(length(xSIGNAL)-2),mq2,...
        length(xSIGNAL),length(xSIGNAL));
    Q = Q1 + Q2 + Q3 + Q4 + Q5;
    Q(1,2) = 2*Q(1,2);
    Q(1,3) = 2*Q(1,3);
    Q(2,4) = 2*Q(2,4);
    Q(length(xSIGNAL),length(xSIGNAL)-1) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-1);
    Q(length(xSIGNAL),length(xSIGNAL)-2) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-2);
    Q(length(xSIGNAL)-1,length(xSIGNAL)-3) = 2*Q(length(xSIGNAL)-1,...
        length(xSIGNAL)-3);
    %FATORACAO LU E APLICACAO EM SOM
    [L,U] = lu(Q);
    Y = L\xSIGNAL;
    fSIGNAL = U\Y;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(filter,'ifir_condat3')==1
    %VALORES DO FILTRO%
    q0 = 79/120;
    q1 = 31/180;
    q2 = -1/720;
    %CRIANDO VETOR COM VALORES DO FILTRO
    mq0 = q0*ones(length(xSIGNAL),1);
    mq1 = q1*ones(length(xSIGNAL)-1,1);
    mq2 = q2*ones(length(xSIGNAL)-2,1);
   %CRIANDO MATRIZ SPARSA COM DIAGONAIS DE ACORDO COM VETORES CRIADOS ACIMA
    Q1 = sparse(1:length(xSIGNAL),1:length(xSIGNAL),mq0,...
        length(xSIGNAL),length(xSIGNAL));
    Q2 = sparse(2:length(xSIGNAL),1:(length(xSIGNAL)-1),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q3 = sparse(1:(length(xSIGNAL)-1),2:length(xSIGNAL),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q4 = sparse(1:(length(xSIGNAL)-2),3:length(xSIGNAL),mq2,...
        length(xSIGNAL),length(xSIGNAL));
    Q5 = sparse(3:length(xSIGNAL),1:(length(xSIGNAL)-2),mq2,...
        length(xSIGNAL),length(xSIGNAL));
    Q = Q1 + Q2 + Q3 + Q4 + Q5;
    Q(1,2) = 2*Q(1,2);
    Q(1,3) = 2*Q(1,3);
    Q(2,4) = 2*Q(2,4);
    Q(length(xSIGNAL),length(xSIGNAL)-1) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-1);
    Q(length(xSIGNAL),length(xSIGNAL)-2) = 2*Q(length(xSIGNAL),...
```

```
        length(xSIGNAL)-2);
    Q(length(xSIGNAL)-1,length(xSIGNAL)-3) = 2*Q(length(xSIGNAL)-1,...
        length(xSIGNAL)-3);
    %FATORACAO LU E APLICACAO EM SOM
    [L,U] = lu(Q);
    Y = L\xSIGNAL;
    fSIGNAL = U\Y;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(filter,'ifir_condat_omoms3')==1
    %VALORES DO FILTRO%
    q0 = 523/840;
    q1 = 79/420;
    q2 = 1/1680;
    %CRIANDO VETOR COM VALORES DO FILTRO
    mq0 = q0*ones(length(xSIGNAL),1);
    mq1 = q1*ones(length(xSIGNAL)-1,1);
    mq2 = q2*ones(length(xSIGNAL)-2,1);
   %CRIANDO MATRIZ SPARSA COM DIAGONAIS DE ACORDO COM VETORES CRIADOS ACIMA
    Q1 = sparse(1:length(xSIGNAL),1:length(xSIGNAL),mq0,...
        length(xSIGNAL),length(xSIGNAL));
    Q2 = sparse(2:length(xSIGNAL),1:(length(xSIGNAL)-1),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q3 = sparse(1:(length(xSIGNAL)-1),2:length(xSIGNAL),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q4 = sparse(1:(length(xSIGNAL)-2),3:length(xSIGNAL),mq2,...
        length(xSIGNAL),length(xSIGNAL));
    Q5 = sparse(3:length(xSIGNAL),1:(length(xSIGNAL)-2),mq2,...
        length(xSIGNAL),length(xSIGNAL));
    Q = Q1 + Q2 + Q3 + Q4 + Q5;
    Q(1,2) = 2*Q(1,2);
    Q(1,3) = 2*Q(1,3);
    Q(2,4) = 2*Q(2,4);
    Q(length(xSIGNAL),length(xSIGNAL)-1) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-1);
    Q(length(xSIGNAL),length(xSIGNAL)-2) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-2);
    Q(length(xSIGNAL)-1,length(xSIGNAL)-3) = 2*Q(length(xSIGNAL)-1,...
        length(xSIGNAL)-3);
    %FATORACAO LU E APLICACAO EM SOM
    [L,U] = lu(Q);
    Y = L\xSIGNAL;
    fSIGNAL = U\Y;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(filter,'ifir_blu35')==1
    %VALORES DO FILTRO%
    q0 = 7.36952;
    q1 = 4.67858;
```

```
q2 = 1.20608;
q3 = 0.10541;
q4 = 0.00196;
q5 = 0.00001;
%CRIANDO VETOR COM VALORES DO FILTRO
mq0 = q0*ones(length(xSIGNAL),1);
mq1 = q1*ones(length(xSIGNAL)-1,1);
mq2 = q2*ones(length(xSIGNAL)-2,1);
mq3 = q3*ones(length(xSIGNAL)-3,1);
mq4 = q4*ones(length(xSIGNAL)-4,1);
mq5 = q5*ones(length(xSIGNAL)-5,1);
%CRIANDO MATRIZ SPARSA COM DIAGONAIS DE ACORDO COM VETORES CRIADOS ACIMA
Q1 = sparse(1:length(xSIGNAL),1:length(xSIGNAL),mq0,...
    length(xSIGNAL),length(xSIGNAL));
Q2 = sparse(2:length(xSIGNAL),1:(length(xSIGNAL)-1),mq1,...
    length(xSIGNAL),length(xSIGNAL));
Q3 = sparse(1:(length(xSIGNAL)-1),2:length(xSIGNAL),mq1,...
    length(xSIGNAL),length(xSIGNAL));
Q4 = sparse(1:(length(xSIGNAL)-2),3:length(xSIGNAL),mq2,...
    length(xSIGNAL),length(xSIGNAL));
Q5 = sparse(3:length(xSIGNAL),1:(length(xSIGNAL)-2),mq2,...
    length(xSIGNAL),length(xSIGNAL));
Q6 = sparse(1:(length(xSIGNAL)-3),4:length(xSIGNAL),mq3,...
    length(xSIGNAL),length(xSIGNAL));
Q7 = sparse(4:length(xSIGNAL),1:(length(xSIGNAL)-3),mq3,...
    length(xSIGNAL),length(xSIGNAL));
Q8 = sparse(1:(length(xSIGNAL)-4),5:length(xSIGNAL),mq4,...
    length(xSIGNAL),length(xSIGNAL));
Q9 = sparse(5:length(xSIGNAL),1:(length(xSIGNAL)-4),mq4,...
    length(xSIGNAL),length(xSIGNAL));
Q10 = sparse(1:(length(xSIGNAL)-5),6:length(xSIGNAL),mq5,...
    length(xSIGNAL),length(xSIGNAL));
Q11 = sparse(6:length(xSIGNAL),1:(length(xSIGNAL)-5),mq5,...
    length(xSIGNAL),length(xSIGNAL));
Q = Q1 + Q2 + Q3 + Q4 + Q5 + Q6 + Q7 + Q8 + Q9 + Q10 + Q11;
Q(1,2) = 2*Q(1,2);
Q(1,3) = 2*Q(1,3);
Q(1,4) = 2*Q(1,4);
Q(1,5) = 2*Q(1,5);
Q(1,6) = 2*Q(1,6);
Q(2,4) = 2*Q(2,4);
Q(2,5) = 2*Q(2,5);
Q(2,6) = 2*Q(2,6);
Q(2,7) = 2*Q(2,7);
Q(3,6) = 2*Q(3,6);
Q(3,7) = 2*Q(3,7);
Q(3,8) = 2*Q(3,8);
```

```
    Q(4,8) = 2*Q(4,8);
    Q(4,9) = 2*Q(4,9);
    Q(5,9) = 2*Q(5,9);
    Q(length(xSIGNAL),length(xSIGNAL)-1) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-1);
    Q(length(xSIGNAL),length(xSIGNAL)-2) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-2);
    Q(length(xSIGNAL),length(xSIGNAL)-3) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-3);
    Q(length(xSIGNAL),length(xSIGNAL)-4) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-4);
    Q(length(xSIGNAL),length(xSIGNAL)-5) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-5);
    Q(length(xSIGNAL)-1,length(xSIGNAL)-3) = 2*Q(length(xSIGNAL)-1,...
        length(xSIGNAL)-3);
    Q(length(xSIGNAL)-1,length(xSIGNAL)-4) = 2*Q(length(xSIGNAL)-1,...
        length(xSIGNAL)-4);
    Q(length(xSIGNAL)-1,length(xSIGNAL)-5) = 2*Q(length(xSIGNAL)-1,...
        length(xSIGNAL)-5);
    Q(length(xSIGNAL)-1,length(xSIGNAL)-6) = 2*Q(length(xSIGNAL)-1,...
        length(xSIGNAL)-6);
    Q(length(xSIGNAL)-2,length(xSIGNAL)-5) = 2*Q(length(xSIGNAL)-1,...
        length(xSIGNAL)-5);
    Q(length(xSIGNAL)-2,length(xSIGNAL)-6) = 2*Q(length(xSIGNAL)-2,...
        length(xSIGNAL)-6);
    Q(length(xSIGNAL)-2,length(xSIGNAL)-7) = 2*Q(length(xSIGNAL)-2,...
        length(xSIGNAL)-7);
    Q(length(xSIGNAL)-3,length(xSIGNAL)-7) = 2*Q(length(xSIGNAL)-2,...
        length(xSIGNAL)-7);
    Q(length(xSIGNAL)-3,length(xSIGNAL)-8) = 2*Q(length(xSIGNAL)-2,...
        length(xSIGNAL)-8);
    Q(length(xSIGNAL)-4,length(xSIGNAL)-9) = 2*Q(length(xSIGNAL)-2,...
        length(xSIGNAL)-9);
    %FATORACAO LU E APLICACAO EM SOM
    [L,U] = lu(Q);
    Y = L\xSIGNAL;
    fSIGNAL = U\Y;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(filter,'ifir_sacht_nehab1')==1
    %VALORES DO FILTRO%
    q0 = 0.77412669;
    q1 = 0.11566267;
    q2 = -0.00272602;
    %CRIANDO VETOR COM VALORES DO FILTRO
    mq0 = q0*ones(length(xSIGNAL),1);
    mq1 = q1*ones(length(xSIGNAL)-1,1);
    mq2 = q2*ones(length(xSIGNAL)-2,1);
```

```
    %CRIANDO MATRIZ SPARSA COM DIAGONAIS DE ACORDO COM VETORES CRIADOS ACIMA
     Q1 = sparse(1:length(xSIGNAL),1:length(xSIGNAL),mq0,...
         length(xSIGNAL),length(xSIGNAL));
     Q2 = sparse(2:length(xSIGNAL),1:(length(xSIGNAL)-1),mq1,...
         length(xSIGNAL),length(xSIGNAL));
     Q3 = sparse(1:(length(xSIGNAL)-1),2:length(xSIGNAL),mq1,...
         length(xSIGNAL),length(xSIGNAL));
     Q4 = sparse(1:(length(xSIGNAL)-2),3:length(xSIGNAL),mq2,...
         length(xSIGNAL),length(xSIGNAL));
     Q5 = sparse(3:length(xSIGNAL),1:(length(xSIGNAL)-2),mq2,...
         length(xSIGNAL),length(xSIGNAL));
     Q = Q1 + Q2 + Q3 + Q4 + Q5;
     Q(1,2) = 2*Q(1,2);
     Q(1,3) = 2*Q(1,3);
     Q(2,4) = 2*Q(2,4);
     Q(length(xSIGNAL),length(xSIGNAL)-1) = 2*Q(length(xSIGNAL),...
         length(xSIGNAL)-1);
     Q(length(xSIGNAL),length(xSIGNAL)-2) = 2*Q(length(xSIGNAL),...
         length(xSIGNAL)-2);
     Q(length(xSIGNAL)-1,length(xSIGNAL)-3) = 2*Q(length(xSIGNAL)-1,...
         length(xSIGNAL)-3);
     %FATORACAO LU E APLICACAO EM SOM
     [L,U] = lu(Q);
     Y = L\xSIGNAL;
     fSIGNAL = U\Y;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(filter,'ifir_sacht_nehab2')==1
     %VALORES DO FILTRO%
     q0 = 0.65314970;
     q1 = 0.17889730;
     q2 = -0.00547216;
     %CRIANDO VETOR COM VALORES DO FILTRO
     mq0 = q0*ones(length(xSIGNAL),1);
     mq1 = q1*ones(length(xSIGNAL)-1,1);
     mq2 = q2*ones(length(xSIGNAL)-2,1);
    %CRIANDO MATRIZ SPARSA COM DIAGONAIS DE ACORDO COM VETORES CRIADOS ACIMA
     Q1 = sparse(1:length(xSIGNAL),1:length(xSIGNAL),mq0,...
         length(xSIGNAL),length(xSIGNAL));
     Q2 = sparse(2:length(xSIGNAL),1:(length(xSIGNAL)-1),mq1,...
         length(xSIGNAL),length(xSIGNAL));
     Q3 = sparse(1:(length(xSIGNAL)-1),2:length(xSIGNAL),mq1,...
         length(xSIGNAL),length(xSIGNAL));
     Q4 = sparse(1:(length(xSIGNAL)-2),3:length(xSIGNAL),mq2,...
         length(xSIGNAL),length(xSIGNAL));
     Q5 = sparse(3:length(xSIGNAL),1:(length(xSIGNAL)-2),mq2,...
         length(xSIGNAL),length(xSIGNAL));
     Q = Q1 + Q2 + Q3 + Q4 + Q5;
```

```
    Q(1,2) = 2*Q(1,2);
    Q(1,3) = 2*Q(1,3);
    Q(2,4) = 2*Q(2,4);
    Q(length(xSIGNAL),length(xSIGNAL)-1) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-1);
    Q(length(xSIGNAL),length(xSIGNAL)-2) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-2);
    Q(length(xSIGNAL)-1,length(xSIGNAL)-3) = 2*Q(length(xSIGNAL)-1,...
        length(xSIGNAL)-3);
    %FATORACAO LU E APLICACAO EM SOM
    [L,U] = lu(Q);
    Y = L\xSIGNAL;
    fSIGNAL = U\Y;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(filter,'ifir_bspline1') == 1
    fSIGNAL = xSIGNAL;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(filter,'ifir_sacht_nehab3')==1
    %VALORES DO FILTRO%
    q0 = 0.56528428;
    q1 = 0.21523557;
    q2 = 0.00212228;
    %CRIANDO VETOR COM VALORES DO FILTRO
    mq0 = q0*ones(length(xSIGNAL),1);
    mq1 = q1*ones(length(xSIGNAL)-1,1);
    mq2 = q2*ones(length(xSIGNAL)-2,1);
   %CRIANDO MATRIZ SPARSA COM DIAGONAIS DE ACORDO COM VETORES CRIADOS ACIMA
    Q1 = sparse(1:length(xSIGNAL),1:length(xSIGNAL),mq0,...
        length(xSIGNAL),length(xSIGNAL));
    Q2 = sparse(2:length(xSIGNAL),1:(length(xSIGNAL)-1),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q3 = sparse(1:(length(xSIGNAL)-1),2:length(xSIGNAL),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q4 = sparse(1:(length(xSIGNAL)-2),3:length(xSIGNAL),mq2,...
        length(xSIGNAL),length(xSIGNAL));
    Q5 = sparse(3:length(xSIGNAL),1:(length(xSIGNAL)-2),mq2,...
        length(xSIGNAL),length(xSIGNAL));
    Q = Q1 + Q2 + Q3 + Q4 + Q5;
    Q(1,2) = 2*Q(1,2);
    Q(1,3) = 2*Q(1,3);
    Q(2,4) = 2*Q(2,4);
    Q(length(xSIGNAL),length(xSIGNAL)-1) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-1);
    Q(length(xSIGNAL),length(xSIGNAL)-2) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-2);
    Q(length(xSIGNAL)-1,length(xSIGNAL)-3) = 2*Q(length(xSIGNAL)-1,...
        length(xSIGNAL)-3);
```

```
    %FATORACAO LU E APLICACAO EM SOM
    [L,U] = lu(Q);
    Y = L\xSIGNAL;
    fSIGNAL = U\Y;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(filter,'ifir_bspline2')==1
    %VALORES DO FILTRO%
    q0 = 3/4;
    q1 = 1/8;
    %CRIANDO VETOR COM VALORES DO FILTRO
    mq0 = q0*ones(length(xSIGNAL),1);
    mq1 = q1*ones(length(xSIGNAL)-1,1);
   %CRIANDO MATRIZ SPARSA COM DIAGONAIS DE ACORDO COM VETORES CRIADOS ACIMA
    Q1 = sparse(1:length(xSIGNAL),1:length(xSIGNAL),mq0,...
        length(xSIGNAL),length(xSIGNAL));
    Q2 = sparse(2:length(xSIGNAL),1:(length(xSIGNAL)-1),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q3 = sparse(1:(length(xSIGNAL)-1),2:length(xSIGNAL),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q = Q1 + Q2 + Q3;
    Q(1,2) = 2*Q(1,2);
    Q(length(xSIGNAL),length(xSIGNAL)-1) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-1);
    %FATORACAO LU E APLICACAO EM SOM
    [L,U] = lu(Q);
    Y = L\xSIGNAL;
    fSIGNAL = U\Y;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(filter,'ifir_bspline3')==1
    %VALORES DO FILTRO%
    q0 = 4/6;
    q1 = 1/6;
    %CRIANDO VETOR COM VALORES DO FILTRO
    mq0 = q0*ones(length(xSIGNAL),1);
    mq1 = q1*ones(length(xSIGNAL)-1,1);
   %CRIANDO MATRIZ SPARSA COM DIAGONAIS DE ACORDO COM VETORES CRIADOS ACIMA
    Q1 = sparse(1:length(xSIGNAL),1:length(xSIGNAL),mq0,...
        length(xSIGNAL),length(xSIGNAL));
    Q2 = sparse(2:length(xSIGNAL),1:(length(xSIGNAL)-1),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q3 = sparse(1:(length(xSIGNAL)-1),2:length(xSIGNAL),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q = Q1 + Q2 + Q3;
    Q(1,2) = 2*Q(1,2);
    Q(length(xSIGNAL),length(xSIGNAL)-1) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-1);
    %FATORACAO LU E APLICACAO EM SOM
```

```
    [L,U] = lu(Q);
    Y = L\xSIGNAL;
    fSIGNAL = U\Y;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(filter,'ifir_omoms2')==1
    %VALORES DO FILTRO%
    q0 = 86/120;
    q1 = 17/120;
    %CRIANDO VETOR COM VALORES DO FILTRO
    mq0 = q0*ones(length(xSIGNAL),1);
    mq1 = q1*ones(length(xSIGNAL)-1,1);
   %CRIANDO MATRIZ SPARSA COM DIAGONAIS DE ACORDO COM VETORES CRIADOS ACIMA
    Q1 = sparse(1:length(xSIGNAL),1:length(xSIGNAL),mq0,...
        length(xSIGNAL),length(xSIGNAL));
    Q2 = sparse(2:length(xSIGNAL),1:(length(xSIGNAL)-1),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q3 = sparse(1:(length(xSIGNAL)-1),2:length(xSIGNAL),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q = Q1 + Q2 + Q3;
    Q(1,2) = 2*Q(1,2);
    Q(length(xSIGNAL),length(xSIGNAL)-1) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-1);
    %FATORACAO LU E APLICACAO EM SOM
    [L,U] = lu(Q);
    Y = L\xSIGNAL;
    fSIGNAL = U\Y;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(filter,'ifir_omoms3')==1
    %VALORES DO FILTRO%
    q0 = 0.6190;
    q1 = 0.1905;
    %CRIANDO VETOR COM VALORES DO FILTRO
    mq0 = q0*ones(length(xSIGNAL),1);
    mq1 = q1*ones(length(xSIGNAL)-1,1);
   %CRIANDO MATRIZ SPARSA COM DIAGONAIS DE ACORDO COM VETORES CRIADOS ACIMA
    Q1 = sparse(1:length(xSIGNAL),1:length(xSIGNAL),mq0,...
        length(xSIGNAL),length(xSIGNAL));
    Q2 = sparse(2:length(xSIGNAL),1:(length(xSIGNAL)-1),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q3 = sparse(1:(length(xSIGNAL)-1),2:length(xSIGNAL),mq1,...
        length(xSIGNAL),length(xSIGNAL));
    Q = Q1 + Q2 + Q3;
    Q(1,2) = 2*Q(1,2);
    Q(length(xSIGNAL),length(xSIGNAL)-1) = 2*Q(length(xSIGNAL),...
        length(xSIGNAL)-1);
    %FATORACAO LU E APLICACAO EM SOM
    [L,U] = lu(Q);
```

```
    Y = L\xSIGNAL;
    fSIGNAL = U\Y;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%GENERATING FUNCTIONS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if strcmp(generator,'bspline0')==1
    if size(xSIGNAL,2) == 2
        nSIGNAL = zeros(n*size(xSIGNAL,1),2);
        for i = 1:size(t,2)-2
            a = floor(t(i));
            b = a+1;
            nSIGNAL(i,1) = fSIGNAL(a,1) + fSIGNAL(b,1);
            nSIGNAL(i,2) = fSIGNAL(a,2) + fSIGNAL(b,2);
        end
        nSIGNAL(end)=fSIGNAL(end);
    else
        nSIGNAL = zeros(n*size(xSIGNAL,1),1);
        for i = 1:size(t,2)-1
            a = floor(t(i));
            b = a+1;
            nSIGNAL(i) = fSIGNAL(a) + fSIGNAL(b);
        end
       nSIGNAL(end)=fSIGNAL(end);
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(generator,'bspline1')
    if size(fSIGNAL,2) == 2
        nSIGNAL = zeros(size(t,2),2);
        for i = 1:size(t,2)-1
            a = floor(t(i));
            b = a+1;
            nSIGNAL(i,1) = (1 - abs(t(i) - a))*fSIGNAL(a,1)...
                + (1 - abs(t(i) -b))*fSIGNAL(b,1);
            nSIGNAL(i,2) = (1 - abs(t(i) - a))*fSIGNAL(a,2)...
                + (1 - abs(t(i) -b))*fSIGNAL(b,2);
        end
    else
        nSIGNAL = zeros(size(t,2),1);
        for i = 1:size(t,2)-1
            a = floor(t(i));
            b = a+1;
            nSIGNAL(i) = (1 - abs(t(i) - a))*fSIGNAL(a)...
                + (1 - abs(t(i) -b))*fSIGNAL(b);
        end
```

```
    end
    nSIGNAL(end) = fSIGNAL(end);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(generator,'bspline2')
    %|r|<1.5
        %(9 + r*(-12 + 4*r))/8
    %|r|<0.5
        %(6 - 8*r*r)/8
    if size(xSIGNAL,2) == 2
        nSIGNAL = zeros(size(t,2),2);
        for i = 1:size(t,2)
            a = floor(t(i));
            if t(i) >= 1 && t(i) < 1.5
                b = a + 1;
                nSIGNAL(i,1) = fSIGNAL(a,1)*((6 - 8*abs(t(i) - a)...
                    *abs(t(i) - a))/8)...
                    + 2*fSIGNAL(b,1)*((9 + abs(t(i) - b)...
                    *(-12 + 4*abs(t(i) - b)))/8);
                nSIGNAL(i,2) = fSIGNAL(a,2)*((6 - 8*abs(t(i) - a)...
                    *abs(t(i) - a))/8)...
                    + 2*fSIGNAL(b,2)*((9 + abs(t(i) - b)...
                    *(-12 + 4*abs(t(i) - b)))/8);
            elseif t(i) > 1.5 && abs(t(i) - a) == 0.5
                b = a + 1;
                nSIGNAL(i,1) = fSIGNAL(a,1)*((9 + abs(t(i) - a)...
                    *(-12 + 4*abs(t(i) - a)))/8)...
                    + fSIGNAL(b,1)*((9 + abs(t(i) - b)...
                    *(-12 + 4*abs(t(i) - b)))/8);
                nSIGNAL(i,2) = fSIGNAL(a,2)*((9 + abs(t(i) - a)...
                    *(-12 + 4*abs(t(i) - a)))/8)...
                    + fSIGNAL(b,2)*((9 + abs(t(i) - b)...
                    *(-12 + 4*abs(t(i) - b)))/8);
            elseif t(i) > 1.5 && t(i) < (size(xSIGNAL,1)-0.4)...
                    && abs(t(i) - a) > 0.5
                b = a + 1;
                c = a + 2;
                nSIGNAL(i,1) = fSIGNAL(a,1)*((9 + abs(t(i) - a)...
                    *(-12 + 4*abs(t(i) - a)))/8)...
                    + fSIGNAL(b,1)*((6 - 8*abs(t(i) - b)...
                    *abs(t(i) - b))/8) + fSIGNAL(c,1)...
                    *((9 + abs(t(i) - c)*(-12 + 4*abs(t(i) - c)))/8);
                nSIGNAL(i,2) = fSIGNAL(a,2)*((9 + abs(t(i) - a)...
                    *(-12 + 4*abs(t(i) - a)))/8)...
                    + fSIGNAL(b,2)*((6 - 8*abs(t(i) - b)...
                    *abs(t(i) - b))/8)...
                    + fSIGNAL(c,2)*((9 + abs(t(i) - c)...
                    *(-12 + 4*abs(t(i) - c)))/8);
```

```
        elseif t(i) > 1.5 && t(i) < (size(xSIGNAL,1)-0.4) ...
                && abs(t(i) - a) < 0.5
            b = a - 1;
            c = a + 1;
            nSIGNAL(i,1) = fSIGNAL(b,1)*(((9 + abs(t(i) - b)...
                *(-12 + 4*abs(t(i) - b)))/8))...
                + fSIGNAL(a,1)*(((6 - 8*abs(t(i) - a)...
                *abs(t(i) - a))/8))...
                + fSIGNAL(c,1)*(((9 + abs(t(i) - c)...
                *(-12 + 4*abs(t(i) - c)))/8));
            nSIGNAL(i,2) = fSIGNAL(b,2)*(((9 + abs(t(i) - b)...
                *(-12 + 4*abs(t(i) - b)))/8))...
                + fSIGNAL(a,2)*(((6 - 8*abs(t(i) - a)...
                *abs(t(i) - a))/8))...
                + fSIGNAL(c,2)*(((9 + abs(t(i) - c)...
                *(-12 + 4*abs(t(i) - c)))/8));
        elseif t(i) > (size(xSIGNAL,1)-0.5) && t(i) < size(xSIGNAL,1)
            b = a + 1;
            nSIGNAL(i,1) = 2*fSIGNAL(a,1)*((9 + abs(t(i) - a)...
                *(-12 + 4*abs(t(i) - a)))/8)...
                + fSIGNAL(b,1)*((6 - 8*abs(t(i) - b)*abs(t(i) - b))/8);
            nSIGNAL(i,2) = 2*fSIGNAL(a,2)*((9 + abs(t(i) - a)...
                *(-12 + 4*abs(t(i) - a)))/8)...
                + fSIGNAL(b,2)*((6 - 8*abs(t(i) - b)*abs(t(i) - b))/8);
        elseif t(i) == size(xSIGNAL,1)
            b = a - 1;
            nSIGNAL(i,1) = 2*fSIGNAL(b,1)*((9 + abs(t(i) - b)...
                *(-12 + 4*abs(t(i) - b)))/8)...
                + fSIGNAL(a,1)*((6 - 8*abs(t(i) - a)*abs(t(i) - a))/8);
            nSIGNAL(i,2) = 2*fSIGNAL(b,2)*((9 + abs(t(i) - b)...
                *(-12 + 4*abs(t(i) - b)))/8)...
                + fSIGNAL(a,2)*((6 - 8*abs(t(i) - a)*abs(t(i) - a))/8);
        end
    end
else
    nSIGNAL = zeros(size(t,2),1);
    for i = 1:size(t,2)
        a = floor(t(i));
        if t(i) >= 1 && t(i) < 1.5
            b = a + 1;
            nSIGNAL(i) = fSIGNAL(a)*((6 - 8*abs(t(i) - a)...
                *abs(t(i) - a))/8)...
                + 2*fSIGNAL(b)*((9 + abs(t(i) - b)...
                *(-12 + 4*abs(t(i) - b)))/8);
        elseif t(i) > 1.5 && abs(t(i) - a) == 0.5
            b = a + 1;
            nSIGNAL(i) = fSIGNAL(a)*((9 + abs(t(i) - a)...
```

```
                    *(-12 + 4*abs(t(i) - a)))/8)...
                    + fSIGNAL(b)*((9 + abs(t(i) - b)...
                    *(-12 + 4*abs(t(i) - b)))/8);
            elseif t(i) > 1.5 && t(i) < (size(xSIGNAL,1)-0.5)...
                    && abs(t(i) - a) > 0.5
                b = a + 1;
                c = a + 2;
                nSIGNAL(i) = fSIGNAL(a)*((9 + abs(t(i) - a)...
                    *(-12 + 4*abs(t(i) - a)))/8)...
                    + fSIGNAL(b)*((6 - 8*abs(t(i) - b)...
                    *abs(t(i) - b))/8)...
                    + fSIGNAL(c)*((9 + abs(t(i) - c)...
                    *(-12 + 4*abs(t(i) - c)))/8);
            elseif t(i) > 1.5 && t(i) < (size(xSIGNAL,1)-0.4)...
                    && abs(t(i) - a) < 0.5
                b = a - 1;
                c = a + 1;
                nSIGNAL(i) = fSIGNAL(b)*(((9 + abs(t(i) - b)...
                    *(-12 + 4*abs(t(i) - b)))/8))...
                    + fSIGNAL(a)*(((6 - 8*abs(t(i) - a)...
                    *abs(t(i) - a))/8))...
                    + fSIGNAL(c)*(((9 + abs(t(i) - c)...
                    *(-12 + 4*abs(t(i) - c)))/8));
            elseif t(i) >= (size(xSIGNAL,1)-0.4) && t(i) < size(xSIGNAL,1)
                b = a + 1;
                nSIGNAL(i) = 2*fSIGNAL(a)*((9 + abs(t(i) - a)...
                    *(-12 + 4*abs(t(i) - a)))/8)...
                    + fSIGNAL(b)*((6 - 8*abs(t(i) - b)*abs(t(i) - b))/8);
            elseif t(i) == size(xSIGNAL,1)
                b = a - 1;
                nSIGNAL(i) = 2*fSIGNAL(b)*((9 + abs(t(i) - b)...
                    *(-12 + 4*abs(t(i) - b)))/8)...
                    + fSIGNAL(a)*((6 - 8*abs(t(i) - a)*abs(t(i) - a))/8);
            end
        end
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(generator,'bspline3')
    %|r|<1
    %(4 + r*r*(-6 + 3*r))/6;
    %|r|<2
    %(8 + r*(-12 + (6 - r)*r))/6;
    if size(xSIGNAL,2) == 2
        nSIGNAL = zeros(size(t,2),2);
        for i = 1:size(t,2)
            a = floor(t(i));
            if t(i) == 1
```

```
        b = a + 1;
        nSIGNAL(i,1) = fSIGNAL(a,1)*((4 + abs(t(i)-a)...
            *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6)...
            + 2*fSIGNAL(b,1)*((8 + abs(t(i)-b)...
            *(-12 + (6 - abs(t(i)-b))*abs(t(i)-b)))/6);
        nSIGNAL(i,2) = fSIGNAL(a,2)*((4 + abs(t(i)-a)...
            *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6)...
            + 2*fSIGNAL(b,2)*((8 + abs(t(i)-b)...
            *(-12 + (6 - abs(t(i)-b))*abs(t(i)-b)))/6);
    elseif t(i) > 1 && t(i) < 2
        b = a + 1;
        c = a + 2;
        nSIGNAL(i,1) = fSIGNAL(a,1)*((4 + abs(t(i)-a)...
            *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6)...
            + fSIGNAL(b,1)*((4 + abs(t(i)-b)*abs(t(i)-b)...
            *(-6 + 3*abs(t(i)-b)))/6)...
            + 2*fSIGNAL(c,1)*((8 + abs(t(i)-c)...
            *(-12 + (6 - abs(t(i)-c))*abs(t(i)-c)))/6);
        nSIGNAL(i,2) = fSIGNAL(a,2)*((4 + abs(t(i)-a)...
            *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6)...
            + fSIGNAL(b,2)*((4 + abs(t(i)-b)...
            *abs(t(i)-b)*(-6 + 3*abs(t(i)-b)))/6)...
            + 2*fSIGNAL(c,2)*((8 + abs(t(i)-c)...
            *(-12 + (6 - abs(t(i)-c))*abs(t(i)-c)))/6);
    elseif t(i) > 1 && abs(t(i) - a) == 0 ...
            && t(i) <= (size(xSIGNAL,1)-1)
        b = a - 1;
        c = a + 1;
        nSIGNAL(i,1) = fSIGNAL(b,1)*((8 + abs(t(i)-b)...
            *(-12 + (6 - abs(t(i)-b))*abs(t(i)-b)))/6)...
            + fSIGNAL(a,1)*((4 + abs(t(i)-a)...
            *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6)...
            + fSIGNAL(c,1)*((8 + abs(t(i)-c)...
            *(-12 + (6 - abs(t(i)-c))*abs(t(i)-c)))/6);
        nSIGNAL(i,2) = fSIGNAL(b,2)*((8 + abs(t(i)-b)...
            *(-12 + (6 - abs(t(i)-b))*abs(t(i)-b)))/6)...
            + fSIGNAL(a,2)*((4 + abs(t(i)-a)...
            *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6)...
            + fSIGNAL(c,2)*((8 + abs(t(i)-c)...
            *(-12 + (6 - abs(t(i)-c))*abs(t(i)-c)))/6);
    elseif t(i) > 2 && t(i) < (size(xSIGNAL,1)-1)
        b = a - 1;
        c = a + 1;
        d = a + 2;
        nSIGNAL(i,1) = fSIGNAL(b,1)*((8 + abs(t(i)-b)...
            *(-12 + (6 - abs(t(i)-b))*abs(t(i)-b)))/6)...
            + fSIGNAL(a,1)*((4 + abs(t(i)-a)...
```

```
                *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6)...
                + fSIGNAL(c,1)*((4 + abs(t(i)-c)...
                *abs(t(i)-c)*(-6 + 3*abs(t(i)-c)))/6)...
                + fSIGNAL(d,1)*((8 + abs(t(i)-d)...
                *(-12 + (6 - abs(t(i)-d))*abs(t(i)-d)))/6);
            nSIGNAL(i,2) = fSIGNAL(b,2)*((8 + abs(t(i)-b)...
                *(-12 + (6 - abs(t(i)-b))*abs(t(i)-b)))/6)...
                + fSIGNAL(a,2)*((4 + abs(t(i)-a)...
                *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6)...
                + fSIGNAL(c,2)*((4 + abs(t(i)-c)...
                *abs(t(i)-c)*(-6 + 3*abs(t(i)-c)))/6)...
                + fSIGNAL(d,2)*((8 + abs(t(i)-d)...
                *(-12 + (6 - abs(t(i)-d))*abs(t(i)-d)))/6);
        elseif t(i) > (size(xSIGNAL,1)-1) && t(i) < size(xSIGNAL,1)
            b = a + 1;
            c = a - 1;
            nSIGNAL(i,1) = 2*fSIGNAL(c,1)*((8 + abs(t(i)-c)...
                *(-12 + (6 - abs(t(i)-c))*abs(t(i)-c)))/6)...
                + fSIGNAL(a,1)*((4 + abs(t(i)-a)...
                *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6)...
                + fSIGNAL(b,1)*((4 + abs(t(i)-b)...
                *abs(t(i)-b)*(-6 + 3*abs(t(i)-b)))/6);
            nSIGNAL(i,2) = 2*fSIGNAL(c,2)*((8 + abs(t(i)-c)...
                *(-12 + (6 - abs(t(i)-c))*abs(t(i)-c)))/6)...
                + fSIGNAL(a,2)*((4 + abs(t(i)-a)...
                *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6)...
                + fSIGNAL(b,2)*((4 + abs(t(i)-b)...
                *abs(t(i)-b)*(-6 + 3*abs(t(i)-b)))/6);
        elseif t(i) == size(xSIGNAL,1)
            b = a - 1;
            nSIGNAL(i,1) = 2*fSIGNAL(b,1)*((8 + abs(t(i)-b)...
                *(-12 + (6 - abs(t(i)-b))*abs(t(i)-b)))/6)...
                + fSIGNAL(a,1)*((4 + abs(t(i)-a)...
                *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6);
            nSIGNAL(i,2) = 2*fSIGNAL(b,2)*((8 + abs(t(i)-b)...
                *(-12 + (6 - abs(t(i)-b))*abs(t(i)-b)))/6)...
                + fSIGNAL(a,2)*((4 + abs(t(i)-a)...
                *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6);
        end
    end
else
    %|r|<1
    %(4 + r*r*(-6 + 3*r))/6;
    %|r|<2
    %(8 + r*(-12 + (6 - r)*r))/6;
    nSIGNAL = zeros(size(t,2),1);
    for i = 1:size(t,2)
```

```
a = floor(t(i));
if t(i) == 1
    b = a + 1;
    nSIGNAL(i) = fSIGNAL(a)*((4 + abs(t(i)-a)...
        *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6)...
        + 2*fSIGNAL(b)*((8 + abs(t(i)-b)...
        *(-12 + (6 - abs(t(i)-b))*abs(t(i)-b)))/6);
elseif t(i) > 1 && t(i) < 2
    b = a + 1;
    c = a + 2;
    nSIGNAL(i) = fSIGNAL(a)*((4 + abs(t(i)-a)...
        *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6)...
        + fSIGNAL(b)*((4 + abs(t(i)-b)...
        *abs(t(i)-b)*(-6 + 3*abs(t(i)-b)))/6)...
        + 2*fSIGNAL(c)*((8 + abs(t(i)-c)...
        *(-12 + (6 - abs(t(i)-c))*abs(t(i)-c)))/6);
elseif t(i) > 1 && abs(t(i) - a) == 0 ...
        && t(i) <= (size(xSIGNAL,1)-1)
    b = a - 1;
    c = a + 1;
    nSIGNAL(i) = fSIGNAL(b)*((8 + abs(t(i)-b)...
        *(-12 + (6 - abs(t(i)-b))*abs(t(i)-b)))/6)...
        + fSIGNAL(a)*((4 + abs(t(i)-a)...
        *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6)...
        + fSIGNAL(c)*((8 + abs(t(i)-c)...
        *(-12 + (6 - abs(t(i)-c))*abs(t(i)-c)))/6);
elseif t(i) > 2 && t(i) < (size(xSIGNAL,1)-1)
    b = a - 1;
    c = a + 1;
    d = a + 2;
    nSIGNAL(i) = fSIGNAL(b)*((8 + abs(t(i)-b)...
        *(-12 + (6 - abs(t(i)-b))*abs(t(i)-b)))/6)...
        + fSIGNAL(a)*((4 + abs(t(i)-a)...
        *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6) ...
        + fSIGNAL(c)*((4 + abs(t(i)-c)*abs(t(i)-c)...
        *(-6 + 3*abs(t(i)-c)))/6) ...
        + fSIGNAL(d)*((8 + abs(t(i)-d)*...
        (-12 + (6 - abs(t(i)-d))*abs(t(i)-d)))/6);
elseif t(i) > (size(xSIGNAL,1)-1) && t(i) < size(xSIGNAL,1)
    b = a + 1;
    c = a - 1;
    nSIGNAL(i) = 2*fSIGNAL(c)*((8 + abs(t(i)-c)...
        *(-12 + (6 - abs(t(i)-c))*abs(t(i)-c)))/6) ...
        + fSIGNAL(a)*((4 + abs(t(i)-a)...
        *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6) ...
        + fSIGNAL(b)*((4 + abs(t(i)-b)...
        *abs(t(i)-b)*(-6 + 3*abs(t(i)-b)))/6);
```

```
            elseif t(i) == size(xSIGNAL,1)
                b = a - 1;
                nSIGNAL(i) = 2*fSIGNAL(b)*((8 + abs(t(i)-b)...
                    *(-12 + (6 - abs(t(i)-b))*abs(t(i)-b)))/6) ...
                    + fSIGNAL(a)*((4 + abs(t(i)-a)...
                    *abs(t(i)-a)*(-6 + 3*abs(t(i)-a)))/6);
            end
        end

    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(generator,'omoms2')
    %r = fabsf(r);
     %           if (r < .5f) return (86.f-120.f*r*r)/120.f;
      %          else if (r == .5f) return 59.f/120.f; // discontinuity!
       %         else if (r < 1.5f) return (137.f+r*(-180.f+60*r))/120.f;
        %        else if (r == 1.5f) return 1.f/120.f; // discontinuity!
         %       else return 0.f;
    if size(xSIGNAL,2) == 2
        nSIGNAL = zeros(size(t,2),2);
        for i = 1:size(t,2)

        end
    else
        nSIGNAL = zeros(size(t,2),1);
        for i = 1:size(t,2)
            a = floor(t(i));
            if t(i) < 1.5
                b = a + 1;
                nSIGNAL(i) = fSIGNAL(a)*((86-120*abs(t(i)-a)...
                    *abs(t(i)-a))/120) ...
                    + 2*fSIGNAL(b)*((137+abs(t(i)-b)...
                    *(-180+60*abs(t(i)-b)))/120);
            elseif t(i) == 1.5
                b = a + 1;
                c = a + 2;
                nSIGNAL(i) = fSIGNAL(a)*(59/120) + fSIGNAL(b)*(59/120) ...
                    + 2*fSIGNAL(c)*(1/120);
            elseif t(i) > 1.5 && t(i) < size(xSIGNAL,1) - 0.5 ...
                    && abs(t(i) - a) > 0.5
                b = a + 1;
                c = a + 2;
                nSIGNAL(i) = fSIGNAL(a)*((137+abs(t(i)-a)...
                    *(-180+60*abs(t(i)-a)))/120) ...
                    + fSIGNAL(b)*((86-120*abs(t(i)-b)...
                    *abs(t(i)-b))/120) + ...
                    + fSIGNAL(c)*((137+abs(t(i)-c)...
```

```
                *(-180+60*abs(t(i)-c)))/120);
        elseif t(i) > 1.5 && t(i) < size(xSIGNAL,1) - 0.5 ...
                && abs(t(i) - a) < 0.5
            b = a - 1;
            c = a + 1;
            nSIGNAL(i) = fSIGNAL(b)*((137+abs(t(i)-b)...
                *(-180+60*abs(t(i)-b)))/120) ...
                + fSIGNAL(a)*((86-120*abs(t(i)-a)*abs(t(i)-a))/120) ...
                + fSIGNAL(c)*((137+abs(t(i)-c)...
                *(-180+60*abs(t(i)-c)))/120);
        elseif t(i) > 1.5 && t(i) < size(xSIGNAL,1) - 0.5 ...
                && abs(t(i) - a) == 0.5
            b = a - 1;
            c = a + 1;
            d = a + 2;
            nSIGNAL(i) = fSIGNAL(b)*(1/120) + fSIGNAL(a)*(59/120) ...
                + fSIGNAL(c)*(59/120) + fSIGNAL(d)*(1/120);
        elseif t(i) == size(xSIGNAL,1) - 0.5
            b = a - 1;
            c = a + 1;
            nSIGNAL(i) = 2*fSIGNAL(b)*(1/120) + fSIGNAL(a)*(59/120) ...
                + fSIGNAL(c)*(59/120);
        elseif t(i) > size(xSIGNAL,1) - 0.5 && t(i) < size(xSIGNAL,1)
            b = a + 1;
            nSIGNAL(i) = 2*fSIGNAL(a)*((137+abs(t(i)-a)...
                *(-180+60*abs(t(i)-a)))/120) ...
                + fSIGNAL(b)*((86-120*abs(t(i)-b)*abs(t(i)-b))/120);
        elseif t(i) == size(xSIGNAL,1)
            b = a - 1;
            nSIGNAL(i) = 2*fSIGNAL(b)*((137+abs(t(i)-b)...
                *(-180+60*abs(t(i)-b)))/120) ...
                + fSIGNAL(a)*((86-120*abs(t(i)-a)*abs(t(i)-a))/120);
        end
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(generator,'omoms3')
    %|r| < 2
    %(58+r*(-85+(42-7*r)*r))/42
    %|r| < 1
    %(26+r*(3+r*(-42+21*r)))/42
    if size(xSIGNAL,2) == 2
        nSIGNAL = zeros(size(t,2),2);
        for i = 1:size(t,2)
            a = floor(t(i));
            if t(i) == 1
                b = a + 1;
```

```
        nSIGNAL(i,1) = fSIGNAL(a,1)*((26+abs(t(i)-a)*(3+abs(t(i)-a)...
            *(-42+21*abs(t(i)-a))))/42) ...
            + 2*fSIGNAL(b,1)*((58+abs(t(i)-b)...
            *(-85+(42-7*abs(t(i)-b))*abs(t(i)-b)))/42);
        nSIGNAL(i,2) = fSIGNAL(a,2)*((26+abs(t(i)-a)*(3+abs(t(i)-a)...
            *(-42+21*abs(t(i)-a))))/42) ...
            + 2*fSIGNAL(b,2)*((58+abs(t(i)-b)...
            *(-85+(42-7*abs(t(i)-b))*abs(t(i)-b)))/42);
    elseif t(i) < 2
        b = a + 1;
        c = a + 2;
        nSIGNAL(i,1) = fSIGNAL(a,1)*((26+abs(t(i)-a)...
            *(3+abs(t(i)-a)*(-42+21*abs(t(i)-a))))/42) ...
            + fSIGNAL(b,1)*((26+abs(t(i)-b)*(3+abs(t(i)-b)...
            *(-42+21*abs(t(i)-b))))/42) ...
            + 2*fSIGNAL(c,1)*((58+abs(t(i)-c)...
            *(-85+(42-7*abs(t(i)-c))*abs(t(i)-c)))/42);
        nSIGNAL(i,2) = fSIGNAL(a,2)*((26+abs(t(i)-a)...
            *(3+abs(t(i)-a)*(-42+21*abs(t(i)-a))))/42) ...
            + fSIGNAL(b,2)*((26+abs(t(i)-b)*(3+abs(t(i)-b)...
            *(-42+21*abs(t(i)-b))))/42) ...
            + 2*fSIGNAL(c,2)*((58+abs(t(i)-c)...
            *(-85+(42-7*abs(t(i)-c))*abs(t(i)-c)))/42);
    elseif t(i) >= 2 && t(i) <= (size(xSIGNAL,1)-1) ...
            && abs(t(i) - a) == 0
        b = a - 1;
        c = a + 1;
        nSIGNAL(i,1) = fSIGNAL(b,1)*((58+abs(t(i)-b)...
            *(-85+(42-7*abs(t(i)-b))*abs(t(i)-b)))/42) ...
            + fSIGNAL(a,1)*((26+abs(t(i)-a)*(3+abs(t(i)-a)...
            *(-42+21*abs(t(i)-a))))/42) ...
            + fSIGNAL(c,1)*((58+abs(t(i)-c)...
            *(-85+(42-7*abs(t(i)-c))*abs(t(i)-c)))/42);
        nSIGNAL(i,2) = fSIGNAL(b,2)*((58+abs(t(i)-b)...
            *(-85+(42-7*abs(t(i)-b))*abs(t(i)-b)))/42) ...
            + fSIGNAL(a,2)*((26+abs(t(i)-a)*(3+abs(t(i)-a)...
            *(-42+21*abs(t(i)-a))))/42) ...
            + fSIGNAL(c,2)*((58+abs(t(i)-c)...
            *(-85+(42-7*abs(t(i)-c))*abs(t(i)-c)))/42);
    elseif t(i) > 2 && t(i) < (size(xSIGNAL,1)-1) ...
            && abs(t(i) - a) > 0
        b = a - 1;
        c = a + 1;
        d = a + 2;
        nSIGNAL(i,1) = fSIGNAL(b,1)*((58+abs(t(i)-b)...
            *(-85+(42-7*abs(t(i)-b))*abs(t(i)-b)))/42) ...
            + fSIGNAL(a,1)*((26+abs(t(i)-a)*(3+abs(t(i)-a)...
```

```
                *(-42+21*abs(t(i)-a))))/42) ...
                + fSIGNAL(c,1)*((26+abs(t(i)-c)*(3+abs(t(i)-c)...
                *(-42+21*abs(t(i)-c))))/42) ...
                + fSIGNAL(d,1)*((58+abs(t(i)-d)...
                *(-85+(42-7*abs(t(i)-d))*abs(t(i)-d)))/42);
            nSIGNAL(i,2) = fSIGNAL(b,1)*((58+abs(t(i)-b)...
                *(-85+(42-7*abs(t(i)-b))*abs(t(i)-b)))/42) ...
                + fSIGNAL(a,2)*((26+abs(t(i)-a)*(3+abs(t(i)-a)...
                *(-42+21*abs(t(i)-a))))/42) ...
                + fSIGNAL(c,2)*((26+abs(t(i)-c)*(3+abs(t(i)-c)...
                *(-42+21*abs(t(i)-c))))/42) ...
                + fSIGNAL(d,2)*((58+abs(t(i)-d)...
                *(-85+(42-7*abs(t(i)-d))*abs(t(i)-d)))/42);
        elseif t(i) > (size(xSIGNAL,1)-1) && t(i) < size(xSIGNAL,1)
            b = a - 1;
            c = a + 1;
            nSIGNAL(i,1) = 2*fSIGNAL(b,1)*((58+abs(t(i)-b)...
                *(-85+(42-7*abs(t(i)-b))*abs(t(i)-b)))/42) ...
                + fSIGNAL(a,1)*((26+abs(t(i)-a)*(3+abs(t(i)-a)...
                *(-42+21*abs(t(i)-a))))/42) ...
                + fSIGNAL(c,1)*((26+abs(t(i)-c)*(3+abs(t(i)-c)...
                *(-42+21*abs(t(i)-c))))/42);
            nSIGNAL(i,2) = 2*fSIGNAL(b,2)*((58+abs(t(i)-b)...
                *(-85+(42-7*abs(t(i)-b))*abs(t(i)-b)))/42) ...
                + fSIGNAL(a,2)*((26+abs(t(i)-a)*(3+abs(t(i)-a)...
                *(-42+21*abs(t(i)-a))))/42) ...
                + fSIGNAL(c,2)*((26+abs(t(i)-c)*(3+abs(t(i)-c)...
                *(-42+21*abs(t(i)-c))))/42);
        elseif t(i) == size(xSIGNAL,1)
            b = a - 1;
            nSIGNAL(i,1) = 2*fSIGNAL(b,1)*((58+abs(t(i)-b)...
                *(-85+(42-7*abs(t(i)-b))*abs(t(i)-b)))/42) ...
                + fSIGNAL(a,1)*((26+abs(t(i)-a)*(3+abs(t(i)-a)...
                *(-42+21*abs(t(i)-a))))/42);
            nSIGNAL(i,2) = 2*fSIGNAL(b,2)*((58+abs(t(i)-b)...
                *(-85+(42-7*abs(t(i)-b))*abs(t(i)-b)))/42) ...
                + fSIGNAL(a,2)*((26+abs(t(i)-a)*(3+abs(t(i)-a)...
                *(-42+21*abs(t(i)-a))))/42);
        end
    end
else
    nSIGNAL = zeros(size(t,2),1);
    for i = 1:size(t,2)
        a = floor(t(i));
        if t(i) == 1
            b = a + 1;
            nSIGNAL(i) = fSIGNAL(a)*((26+abs(t(i)-a)*(3+abs(t(i)-a)...
```

```
            *(-42+21*abs(t(i)-a))))/42) ...
            + 2*fSIGNAL(b)*((58+abs(t(i)-b)...
            *(-85+(42-7*abs(t(i)-b))*abs(t(i)-b)))/42);
    elseif t(i) < 2
        b = a + 1;
        c = a + 2;
        nSIGNAL(i) = fSIGNAL(a)*((26+abs(t(i)-a)*(3+abs(t(i)-a)...
            *(-42+21*abs(t(i)-a))))/42) ...
            + fSIGNAL(b)*((26+abs(t(i)-b)*(3+abs(t(i)-b)...
            *(-42+21*abs(t(i)-b))))/42) ...
            + 2*fSIGNAL(c)*((58+abs(t(i)-c)...
            *(-85+(42-7*abs(t(i)-c))*abs(t(i)-c)))/42);
    elseif t(i) >= 2 && t(i) <= (size(xSIGNAL,1)-1) ...
            && abs(t(i) - a) == 0
        b = a - 1;
        c = a + 1;
        nSIGNAL(i) = fSIGNAL(b)*((58+abs(t(i)-b)...
            *(-85+(42-7*abs(t(i)-b))*abs(t(i)-b)))/42) ...
            + fSIGNAL(a)*((26+abs(t(i)-a)*(3+abs(t(i)-a)...
            *(-42+21*abs(t(i)-a))))/42) ...
            + fSIGNAL(c)*((58+abs(t(i)-c)...
            *(-85+(42-7*abs(t(i)-c))*abs(t(i)-c)))/42);
    elseif t(i) > 2 && t(i) < (size(xSIGNAL,1)-1) ...
            && abs(t(i) - a) > 0
        b = a - 1;
        c = a + 1;
        d = a + 2;
        nSIGNAL(i) = fSIGNAL(b)*((58+abs(t(i)-b)...
            *(-85+(42-7*abs(t(i)-b))*abs(t(i)-b)))/42) ...
            + fSIGNAL(a)*((26+abs(t(i)-a)*(3+abs(t(i)-a)...
            *(-42+21*abs(t(i)-a))))/42) ...
            + fSIGNAL(c)*((26+abs(t(i)-c)*(3+abs(t(i)-c)...
            *(-42+21*abs(t(i)-c))))/42) ...
            + fSIGNAL(d)*((58+abs(t(i)-d)...
            *(-85+(42-7*abs(t(i)-d))*abs(t(i)-d)))/42);
    elseif t(i) > (size(xSIGNAL,1)-1) && t(i) < size(xSIGNAL,1)
        b = a - 1;
        c = a + 1;
        nSIGNAL(i) = 2*fSIGNAL(b)*((58+abs(t(i)-b)...
            *(-85+(42-7*abs(t(i)-b))*abs(t(i)-b)))/42) ...
            + fSIGNAL(a)*((26+abs(t(i)-a)*(3+abs(t(i)-a)...
            *(-42+21*abs(t(i)-a))))/42) ...
            + fSIGNAL(c)*((26+abs(t(i)-c)*(3+abs(t(i)-c)...
            *(-42+21*abs(t(i)-c))))/42);
    elseif t(i) == size(xSIGNAL,1)
        b = a - 1;
        nSIGNAL(i) = 2*fSIGNAL(b)*((58+abs(t(i)-b)...
```

```
                    *(-85+(42-7*abs(t(i)-b))*abs(t(i)-b)))/42) ...
                    + fSIGNAL(a)*((26+abs(t(i)-a)*(3+abs(t(i)-a)...
                    *(-42+21*abs(t(i)-a))))/42);
            end
        end
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(generator,'sacht_nehab1')
    %r = fabsf(r);
     %           if (r < 1.f) return 0.89538176f - 0.79076352f*r;
       %         else if (r==1.f) return 0.05230912f;
        %        else return 0.f;
        if size(xSIGNAL,2) == 2
        nSIGNAL = zeros(size(t,2),2);
            for i = 1:size(t,2)
                a = floor(t(i));
                if t(i) == 1
                    b = a + 1;
                    nSIGNAL(i,1) = fSIGNAL(a,1)*(0.89538176 ...
                        - 0.79076352*abs(t(i)-a)) ...
                        + 2*fSIGNAL(b,1)*0.05230912;
                    nSIGNAL(i,2) = fSIGNAL(a,2)*(0.89538176 ...
                        - 0.79076352*abs(t(i)-a)) ...
                        + 2*fSIGNAL(b,2)*0.05230912;
                elseif t(i) > 1 && t(i) < size(xSIGNAL,1) ...
                        && abs(t(i) - a) ~= 0
                    b = a + 1;
                    nSIGNAL(i,1) = fSIGNAL(a,1)*(0.89538176 ...
                        - 0.79076352*abs(t(i)-a)) ...
                        + fSIGNAL(b,1)*(0.89538176 ...
                        - 0.79076352*abs(t(i)-b));
                    nSIGNAL(i,2) = fSIGNAL(a,2)*(0.89538176 ...
                        - 0.79076352*abs(t(i)-a)) ...
                        + fSIGNAL(b,2)*(0.89538176 ...
                        - 0.79076352*abs(t(i)-b));
                elseif t(i) > 1 && t(i) < size(xSIGNAL,1) ...
                        && abs(t(i) - a) == 0
                    b = a - 1;
                    c = a + 1;
                    nSIGNAL(i,1) = fSIGNAL(b,1)*0.05230912 ...
                        + fSIGNAL(a,1)*(0.89538176 ...
                        - 0.79076352*abs(t(i)-a)) ...
                        + fSIGNAL(c,1)*0.05230912;
                    nSIGNAL(i,2) = fSIGNAL(b,2)*0.05230912 ...
                        + fSIGNAL(a,2)*(0.89538176 ...
                        - 0.79076352*abs(t(i)-a)) ...
                        + fSIGNAL(c,2)*0.05230912;
```

```
                elseif t(i) == size(xSIGNAL,1)
                    b = a - 1;
                    nSIGNAL(i,1) = 2*fSIGNAL(b,1)*0.05230912 ...
                        + fSIGNAL(a,1)*(0.89538176 - 0.79076352...
                        *abs(t(i)-a));
                    nSIGNAL(i,2) = 2*fSIGNAL(b,2)*0.05230912 ...
                        + fSIGNAL(a,2)*(0.89538176 - 0.79076352...
                        *abs(t(i)-a));
                end
            end
        else
            nSIGNAL = zeros(size(t,2),1);
            for i = 1:size(t,2)
                a = floor(t(i));
                if t(i) == 1
                    b = a + 1;
                    nSIGNAL(i) = fSIGNAL(a)*(0.89538176 ...
                        - 0.79076352*abs(t(i)-a)) ...
                        + 2*fSIGNAL(b)*0.05230912;
                elseif t(i) > 1 && t(i) < size(xSIGNAL,1) ...
                        && abs(t(i) - a) ~= 0
                    b = a + 1;
                    nSIGNAL(i) = fSIGNAL(a)*(0.89538176 ...
                        - 0.79076352*abs(t(i)-a)) ...
                        + fSIGNAL(b)*(0.89538176 ...
                        - 0.79076352*abs(t(i)-b));
                elseif t(i) > 1 && t(i) < size(xSIGNAL,1) ...
                        && abs(t(i) - a) == 0
                    b = a - 1;
                    c = a + 1;
                    nSIGNAL(i) = fSIGNAL(b,1)*0.05230912 ...
                        + fSIGNAL(a)*(0.89538176 ...
                        - 0.79076352*abs(t(i)-a)) ...
                        + fSIGNAL(c)*0.05230912;
                elseif t(i) == size(xSIGNAL,1)
                    b = a - 1;
                    nSIGNAL(i) = 2*fSIGNAL(b)*0.05230912 ...
                        + fSIGNAL(a)*(0.89538176 - 0.79076352*abs(t(i)-a));
                end
            end
        end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(generator,'sacht_nehab2')
   %r = fabsf(r);
    %            if (r < 0.5f)
     %               return 0.66118661f - 0.75627421f*r*r;
      %          else if (r==0.5f)
```

```
    %             return 0.49205903f;
     %        else if (r < 1.5f)
      %           return 1.04366188f - 1.25239228f*r + 0.37813711f*r*r;
       %      else if (r==1.5f)
        %         return 0.00794097f;
         %    else return 0.f;
if size(xSIGNAL,2) == 2
    nSIGNAL = zeros(size(t,2),2);
        for i = 1:size(t,2)
            a = floor(t(i));
            if t(i) < 1.5
                b = a + 1;
                nSIGNAL(i,1) = fSIGNAL(a,1)*(0.66118661 ...
                    - 0.75627421*abs(t(i)-a)*abs(t(i)-a)) ...
                    + 2*fSIGNAL(b,1)*(1.04366188 ...
                    - 1.25239228*abs(t(i)-b) ...
                    + 0.37813711*abs(t(i)-b)*abs(t(i)-b));
                nSIGNAL(i,2) = fSIGNAL(a,2)*(0.66118661 ...
                    - 0.75627421*abs(t(i)-a)*abs(t(i)-a)) ...
                    + 2*fSIGNAL(b,2)*(1.04366188 ...
                    - 1.25239228*abs(t(i)-b) ...
                    + 0.37813711*abs(t(i)-b)*abs(t(i)-b));
            elseif t(i) == 1.5
                b = a + 1;
                c = a + 2;
                nSIGNAL(i,1) = fSIGNAL(a,1)*0.49205903 ...
                    + fSIGNAL(b,1)*0.49205903 ...
                    + 2*fSIGNAL(c,1)*0.00794097;
                nSIGNAL(i,2) = fSIGNAL(a,2)*0.49205903 ...
                    + fSIGNAL(b,2)*0.49205903 ...
                    + 2*fSIGNAL(c,2)*0.00794097;
            elseif t(i) > 1.5 && t(i) < (size(xSIGNAL,1) - 0.5) ...
                    && abs(t(i)-a) > 0.5
                b = a + 1;
                c = a + 2;
                nSIGNAL(i,1) = fSIGNAL(a,1)*(1.04366188 ...
                    - 1.25239228*abs(t(i)-a) ...
                    + 0.37813711*abs(t(i)-a)*abs(t(i)-a)) ...
                    + fSIGNAL(b,1)*(0.66118661 ...
                    - 0.75627421*abs(t(i) - b)*abs(t(i) - b)) ...
                    + fSIGNAL(c,1)*(1.04366188 ...
                    - 1.25239228*abs(t(i) - c) ...
                    + 0.37813711*abs(t(i) - c)*abs(t(i) - c));
                nSIGNAL(i,2) = fSIGNAL(a,2)*(1.04366188 ...
                    - 1.25239228*abs(t(i)-a) ...
                    + 0.37813711*abs(t(i)-a)*abs(t(i)-a)) ...
                    + fSIGNAL(b,2)*(0.66118661 ...
```

```
            - 0.75627421*abs(t(i) - b)*abs(t(i) - b)) ...
            + fSIGNAL(c,2)*(1.04366188 ...
            - 1.25239228*abs(t(i) - c) ...
            + 0.37813711*abs(t(i) - c)*abs(t(i) - c));
    elseif t(i) > 1.5 && t(i) < (size(xSIGNAL,1) - 0.5) ...
            && abs(t(i)-a) < 0.5
        b = a - 1;
        c = a + 1;
        nSIGNAL(i,1) = fSIGNAL(b,1)*(1.04366188 ...
            - 1.25239228*abs(t(i) - b) ...
            + 0.37813711*abs(t(i) - b)*abs(t(i) - b)) ...
            + fSIGNAL(a,1)*(0.66118661 ...
            - 0.75627421*abs(t(i) - a)*abs(t(i) - a)) ...
            + fSIGNAL(c,1)*(1.04366188 ...
            - 1.25239228*abs(t(i) - c) ...
            + 0.37813711*abs(t(i) - c)*abs(t(i) - c));
        nSIGNAL(i,2) = fSIGNAL(b,2)*(1.04366188 ...
            - 1.25239228*abs(t(i) - b) ...
            + 0.37813711*abs(t(i) - b)*abs(t(i) - b)) ...
            + fSIGNAL(a,2)*(0.66118661 ...
            - 0.75627421*abs(t(i) - a)*abs(t(i) - a)) ...
            + fSIGNAL(c,2)*(1.04366188 ...
            - 1.25239228*abs(t(i) - c) ...
            + 0.37813711*abs(t(i) - c)*abs(t(i) - c));
    elseif t(i) > 1.5 && t(i) < (size(xSIGNAL,1) - 0.5) ...
            && abs(t(i)-a) == 0.5
        b = a - 1;
        c = a + 1;
        d = a + 2;
        nSIGNAL(i,1) = fSIGNAL(b,1)*0.00794097 ...
            + fSIGNAL(a,1)*0.49205903 ...
            + fSIGNAL(c,1)*0.49205903 ...
            + fSIGNAL(d,1)*0.00794097;
        nSIGNAL(i,2) = fSIGNAL(b,2)*0.00794097 ...
            + fSIGNAL(a,2)*0.49205903 ...
            + fSIGNAL(c,2)*0.49205903 ...
            + fSIGNAL(d,2)*0.00794097;
    elseif t(i) == (size(xSIGNAL,1) - 0.5)
        b = a - 1;
        c = a + 1;
        nSIGNAL(i,1) = 2*fSIGNAL(b,1)*0.00794097 ...
            + fSIGNAL(a,1)*0.49205903 ...
            + fSIGNAL(c,1)*0.49205903;
        nSIGNAL(i,2) = 2*fSIGNAL(b,2)*0.00794097 ...
            + fSIGNAL(a,2)*0.49205903 ...
            + fSIGNAL(c,2)*0.49205903;
    elseif t(i) > (size(xSIGNAL,1) - 0.5) ...
```

```
                    && t(i) < size(xSIGNAL,1)
                b = a + 1;
                nSIGNAL(i,1) = 2*fSIGNAL(a,1)*(1.04366188 ...
                    - 1.25239228*abs(t(i) - a) ...
                    + 0.37813711*abs(t(i) - a)*abs(t(i) - a)) ...
                    + fSIGNAL(b,1)*(0.66118661 - 0.75627421...
                    *abs(t(i) - b)*abs(t(i) - b));
                nSIGNAL(i,2) = 2*fSIGNAL(a,2)*(1.04366188 ...
                    - 1.25239228*abs(t(i) - a) ...
                    + 0.37813711*abs(t(i) - a)*abs(t(i) - a)) ...
                    + fSIGNAL(b,2)*(0.66118661 - 0.75627421...
                    *abs(t(i) - b)*abs(t(i) - b));
            elseif t(i) == size(xSIGNAL,1)
                b = a - 1;
                nSIGNAL(i,1) = 2*fSIGNAL(b,1)*(1.04366188 ...
                    - 1.25239228*abs(t(i) - b) ...
                    + 0.37813711*abs(t(i) - b)*abs(t(i) - b)) ...
                    + fSIGNAL(a,1)*(0.66118661 ...
                    - 0.75627421*abs(t(i) - a)*abs(t(i) - a));
                nSIGNAL(i,2) = 2*fSIGNAL(b,2)*(1.04366188 ...
                    - 1.25239228*abs(t(i) - b) ...
                    + 0.37813711*abs(t(i) - b)*abs(t(i) - b)) ...
                    + fSIGNAL(a,2)*(0.66118661 ...
                    - 0.75627421*abs(t(i) - a)*abs(t(i) - a));
            end
        end
    else
        nSIGNAL = zeros(size(t,2),1);
        for i = 1:size(t,2)
            a = floor(t(i));
            if t(i) < 1.5
                b = a + 1;
                nSIGNAL(i) = fSIGNAL(a)*(0.66118661 ...
                    - 0.75627421*abs(t(i)-a)*abs(t(i)-a)) ...
                    + 2*fSIGNAL(b)*(1.04366188 ...
                    - 1.25239228*abs(t(i)-b) ...
                    + 0.37813711*abs(t(i)-b)*abs(t(i)-b));
            elseif t(i) == 1.5
                b = a + 1;
                c = a + 2;
                nSIGNAL(i) = fSIGNAL(a)*0.49205903 ...
                    + fSIGNAL(b)*0.49205903 ...
                    + 2*fSIGNAL(c)*0.00794097;
            elseif t(i) > 1.5 && t(i) < (size(xSIGNAL,1) - 0.5) ...
                    && abs(t(i)-a) > 0.5
                b = a + 1;
                c = a + 2;
```

```
        nSIGNAL(i) = fSIGNAL(a)*(1.04366188 ...
            - 1.25239228*abs(t(i)-a) ...
            + 0.37813711*abs(t(i)-a)*abs(t(i)-a)) ...
            + fSIGNAL(b)*(0.66118661 ...
            - 0.75627421*abs(t(i) - b)*abs(t(i) - b)) ...
            + fSIGNAL(c)*(1.04366188 ...
            - 1.25239228*abs(t(i) - c) ...
            + 0.37813711*abs(t(i) - c)*abs(t(i) - c));
    elseif t(i) > 1.5 && t(i) < (size(xSIGNAL,1) - 0.5) ...
            && abs(t(i)-a) < 0.5
        b = a - 1;
        c = a + 1;
        nSIGNAL(i) = fSIGNAL(b)*(1.04366188 ...
            - 1.25239228*abs(t(i) - b) ...
            + 0.37813711*abs(t(i) - b)*abs(t(i) - b)) ...
            + fSIGNAL(a)*(0.66118661 ...
            - 0.75627421*abs(t(i) - a)*abs(t(i) - a)) ...
            + fSIGNAL(c)*(1.04366188 ...
            - 1.25239228*abs(t(i) - c) ...
            + 0.37813711*abs(t(i) - c)*abs(t(i) - c));
    elseif t(i) > 1.5 && t(i) < (size(xSIGNAL,1) - 0.5) ...
            && abs(t(i)-a) == 0.5
        b = a - 1;
        c = a + 1;
        d = a + 2;
        nSIGNAL(i) = fSIGNAL(b)*0.00794097 ...
            + fSIGNAL(a)*0.49205903 ...
            + fSIGNAL(c)*0.49205903 ...
            + fSIGNAL(d)*0.00794097;
    elseif t(i) == (size(xSIGNAL,1) - 0.5)
        b = a - 1;
        c = a + 1;
        nSIGNAL(i) = 2*fSIGNAL(b)*0.00794097 ...
            + fSIGNAL(a)*0.49205903 ...
            + fSIGNAL(c)*0.49205903;
    elseif t(i) > (size(xSIGNAL,1) - 0.5) ...
            && t(i) < size(xSIGNAL,1)
        b = a + 1;
        nSIGNAL(i) = 2*fSIGNAL(a)*(1.04366188 ...
            - 1.25239228*abs(t(i) - a) ...
            + 0.37813711*abs(t(i) - a)*abs(t(i) - a)) ...
            + fSIGNAL(b)*(0.66118661 ...
            - 0.75627421*abs(t(i) - b)*abs(t(i) - b));
    elseif t(i) == size(xSIGNAL,1)
        b = a - 1;
        nSIGNAL(i) = 2*fSIGNAL(b)*(1.04366188 ...
            - 1.25239228*abs(t(i) - b) ...
```

```
                        + 0.37813711*abs(t(i) - b)*abs(t(i) - b)) ...
                        + fSIGNAL(a)*(0.66118661 ...
                        - 0.75627421*abs(t(i) - a)*abs(t(i) - a));
                end
            end

    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
elseif strcmp(generator,'sacht_nehab3')
%r = fabsf(r);
 %               if (r==0.f)
  %                  return 0.56259136f;
   %             else if (r < 1.f)
    %                return 0.56259136f + 0.04465421f*r - 0.70001154f*r*r
     %                  + 0.30938685f*r*r*r*r;
      %           else if (r==1.f)
       %              return 0.21773426f;
        %         else if (r < 2.f)
         %           return 1.17739188f - 1.50921205f*r + 0.64059253f*r*r
          %             - 0.08992473f*r*r*r*r;
           %      else if (r==2.f)
            %         return 0.00097005f;
             %    else return 0.f;
    if size(xSIGNAL,2) == 2
        nSIGNAL = zeros(size(t,2),2);
        for i = 1:size(t,2)
            a = floor(t(i));
            if t(i) == 1
                b = a + 1;
                c = a + 2;
                nSIGNAL(i,1) = fSIGNAL(a,1)*0.56259136 ...
                    + 2*fSIGNAL(b,1)*0.21773426 ...
                    + 2*fSIGNAL(c,1)*0.00097005;
                nSIGNAL(i,2) = fSIGNAL(a,2)*0.56259136 ...
                    + 2*fSIGNAL(b,2)*0.21773426 ...
                    + 2*fSIGNAL(c,2)*0.00097005;
            elseif t(i) > 1 && t(i) < 2
                b = a + 1;
                c = a + 2;
                nSIGNAL(i,1) = fSIGNAL(a,1)*(0.56259136 ...
                    + 0.04465421*abs(t(i) - a) ...
                    - 0.70001154*abs(t(i) - a)*abs(t(i) - a) ...
                    + 0.30938685*abs(t(i) - a)*abs(t(i) - a)...
                    *abs(t(i) - a)) + fSIGNAL(b,1)*(0.56259136 + ...
                    0.04465421*abs(t(i) - b) ...
                    - 0.70001154*abs(t(i) - b)*abs(t(i) - b) ...
                    + 0.30938685*abs(t(i) - b)*abs(t(i) - b)...
```

```
            *abs(t(i) - b)) + 2*fSIGNAL(c,1)*(1.17739188 ...
            - 1.50921205*abs(t(i) - c) ...
            + 0.64059253*abs(t(i) - c)*abs(t(i) - c) ...
            - 0.08992473*abs(t(i) - c)...
            *abs(t(i) - c)*abs(t(i) - c));
        nSIGNAL(i,2) = fSIGNAL(a,2)*(0.56259136 ...
            + 0.04465421*abs(t(i) - a) ...
            - 0.70001154*abs(t(i) - a)*abs(t(i) - a) ...
            + 0.30938685*abs(t(i) - a)*abs(t(i) - a)...
            *abs(t(i) - a)) + fSIGNAL(b,2)*(0.56259136 + ...
            0.04465421*abs(t(i) - b) ...
            - 0.70001154*abs(t(i) - b)*abs(t(i) - b) ...
            + 0.30938685*abs(t(i) - b)*abs(t(i) - b)...
            *abs(t(i) - b)) + 2*fSIGNAL(c,2)*(1.17739188 ...
            - 1.50921205*abs(t(i) - c) ...
            + 0.64059253*abs(t(i) - c)*abs(t(i) - c) ...
            - 0.08992473*abs(t(i) - c)*abs(t(i) - c)...
            *abs(t(i) - c));
    elseif t(i) == 2
        b = a - 1;
        c = a + 1;
        d = a + 2;
        nSIGNAL(i,1) = fSIGNAL(b,1)*0.21773426 ...
            + fSIGNAL(a,1)*0.56259136 ...
            + fSIGNAL(c,1)*0.21773426 ...
            + 2*fSIGNAL(d,1)*0.00097005;
        nSIGNAL(i,2) = fSIGNAL(b,2)*0.21773426 ...
            + fSIGNAL(a,2)*0.56259136 ...
            + fSIGNAL(c,2)*0.21773426 ...
            + 2*fSIGNAL(d,2)*0.00097005;
    elseif t(i) > 2 && t(i) < (size(xSIGNAL,1) - 1) ...
            && abs(t(i) - a) ~= 0
        b = a - 1;
        c = a + 1;
        d = a + 2;
        nSIGNAL(i,1) = fSIGNAL(b,1)*(1.17739188 ...
            - 1.50921205*abs(t(i) - b) ...
            + 0.64059253*abs(t(i) - b)*abs(t(i) - b) ...
            - 0.08992473*abs(t(i) - b)*abs(t(i) - b)...
            *abs(t(i) - b)) + fSIGNAL(a,1)*(0.56259136 ...
            + 0.04465421*abs(t(i) - a) ...
            - 0.70001154*abs(t(i) - a)*abs(t(i) - a) ...
            + 0.30938685*abs(t(i) - a)*abs(t(i) - a)...
            *abs(t(i) - a)) + fSIGNAL(c,1)*(0.56259136 ...
            + 0.04465421*abs(t(i) - c) ...
            - 0.70001154*abs(t(i) - c)*abs(t(i) - c) ...
            + 0.30938685*abs(t(i) - c)*abs(t(i) - c)...
```

```
            *abs(t(i) - c)) + fSIGNAL(d,1)*(1.17739188 ...
            - 1.50921205*abs(t(i) - d) ...
            + 0.64059253*abs(t(i) - d)*abs(t(i) - d) ...
            - 0.08992473*abs(t(i) - d)*abs(t(i) - d)...
            *abs(t(i) - d));
        nSIGNAL(i,2) = fSIGNAL(b,2)*(1.17739188 ...
            - 1.50921205*abs(t(i) - b) ...
            + 0.64059253*abs(t(i) - b)*abs(t(i) - b) ...
            - 0.08992473*abs(t(i) - b)*abs(t(i) - b)...
            *abs(t(i) - b)) + fSIGNAL(a,2)*(0.56259136 ...
            + 0.04465421*abs(t(i) - a) ...
            - 0.70001154*abs(t(i) - a)*abs(t(i) - a) ...
            + 0.30938685*abs(t(i) - a)*abs(t(i) - a)...
            *abs(t(i) - a)) + fSIGNAL(c,2)*(0.56259136 ...
            + 0.04465421*abs(t(i) - c) ...
            - 0.70001154*abs(t(i) - c)*abs(t(i) - c) ...
            + 0.30938685*abs(t(i) - c)*abs(t(i) - c)...
            *abs(t(i) - c)) + fSIGNAL(d,2)*(1.17739188 ...
            - 1.50921205*abs(t(i) - d) ...
            + 0.64059253*abs(t(i) - d)*abs(t(i) - d) ...
            - 0.08992473*abs(t(i) - d)*abs(t(i) - d)...
            *abs(t(i) - d));
    elseif t(i) > 2 && t(i) < (size(xSIGNAL,1) - 1) ...
            && abs(t(i) - a) == 0
        b = a - 2;
        c = a - 1;
        d = a + 1;
        e = a + 2;
        nSIGNAL(i,1) = fSIGNAL(b,1)*0.00097005 ...
            + fSIGNAL(c,1)*0.21773426 ...
            + fSIGNAL(a,1)*0.56259136 ...
            + fSIGNAL(d,1)*0.21773426 ...
            + fSIGNAL(e,1)*0.00097005;
        nSIGNAL(i,2) = fSIGNAL(b,2)*0.00097005 ...
            + fSIGNAL(c,2)*0.21773426 ...
            + fSIGNAL(a,2)*0.56259136 ...
            + fSIGNAL(d,2)*0.21773426 ...
            + fSIGNAL(e,2)*0.00097005;
    elseif t(i) == (size(xSIGNAL,1) - 1)
        b = a - 2;
        c = a - 1;
        d = a + 1;
        nSIGNAL(i,1) = 2*fSIGNAL(b,1)*0.00097005 ...
            + fSIGNAL(c,1)*0.21773426 ...
            + fSIGNAL(a,1)*0.56259136 ...
            + fSIGNAL(d,1)*0.21773426;
        nSIGNAL(i,2) = 2*fSIGNAL(b,2)*0.00097005 ...
```

```
                + fSIGNAL(c,2)*0.21773426 ...
                + fSIGNAL(a,2)*0.56259136 ...
                + fSIGNAL(d,2)*0.21773426;
        elseif t(i) > (size(xSIGNAL,1) - 1) && t(i) < size(xSIGNAL,1)
            b = a - 1;
            c = a + 1;
            nSIGNAL(i,1) = 2*fSIGNAL(b,1)*(1.17739188 ...
                - 1.50921205*abs(t(i) - b) ...
                + 0.64059253*abs(t(i) - b)*abs(t(i) - b) ...
                - 0.08992473*abs(t(i) - b)*abs(t(i) - b)...
                *abs(t(i) - b)) + fSIGNAL(a,1)*(0.56259136 ...
                + 0.04465421*abs(t(i) - a) ...
                - 0.70001154*abs(t(i) - a)*abs(t(i) - a) ...
                + 0.30938685*abs(t(i) - a)*abs(t(i) - a)...
                *abs(t(i) - a)) + fSIGNAL(c,1)*(1.17739188 ...
                - 1.50921205*abs(t(i) - c) ...
                + 0.64059253*abs(t(i) - c)*abs(t(i) - c) ...
                - 0.08992473*abs(t(i) - c)*abs(t(i) - c)...
                *abs(t(i) - c));
            nSIGNAL(i,2) = 2*fSIGNAL(b,2)*(1.17739188 ...
                - 1.50921205*abs(t(i) - b) ...
                + 0.64059253*abs(t(i) - b)*abs(t(i) - b) ...
                - 0.08992473*abs(t(i) - b)*abs(t(i) - b)...
                *abs(t(i) - b)) + fSIGNAL(a,2)*(0.56259136 ...
                + 0.04465421*abs(t(i) - a) ...
                - 0.70001154*abs(t(i) - a)*abs(t(i) - a) ...
                + 0.30938685*abs(t(i) - a)*abs(t(i) - a)...
                *abs(t(i) - a)) + fSIGNAL(c,2)*(1.17739188 ...
                - 1.50921205*abs(t(i) - c) ...
                + 0.64059253*abs(t(i) - c)*abs(t(i) - c) ...
                - 0.08992473*abs(t(i) - c)*abs(t(i) - c)...
                *abs(t(i) - c));
        elseif t(i) == size(xSIGNAL,1)
            b = a - 2;
            c = a - 1;
            nSIGNAL(i,1) = 2*fSIGNAL(b,1)*0.00097005 ...
                + 2*fSIGNAL(c,1)*0.21773426...
                + fSIGNAL(a,1)*0.56259136;
            nSIGNAL(i,2) = 2*fSIGNAL(b,2)*0.00097005 ...
                + 2*fSIGNAL(c,2)*0.21773426...
                + fSIGNAL(a,2)*0.56259136;
        end
    end
else
    nSIGNAL = zeros(size(t,2),1);
    for i = 1:size(t,2)
        a = floor(t(i));
```

```
if t(i) == 1
    b = a + 1;
    c = a + 2;
    nSIGNAL(i) = fSIGNAL(a)*0.56259136 ...
        + 2*fSIGNAL(b)*0.21773426 ...
        + 2*fSIGNAL(c)*0.00097005;
elseif t(i) > 1 && t(i) < 2
    b = a + 1;
    c = a + 2;
    nSIGNAL(i) = fSIGNAL(a)*(0.56259136 ...
        + 0.04465421*abs(t(i) - a) ...
        - 0.70001154*abs(t(i) - a)*abs(t(i) - a) ...
        + 0.30938685*abs(t(i) - a)*abs(t(i) - a)...
        *abs(t(i) - a)) + fSIGNAL(b)*(0.56259136 + ...
        0.04465421*abs(t(i) - b) ...
        - 0.70001154*abs(t(i) - b)*abs(t(i) - b) ...
        + 0.30938685*abs(t(i) - b)*abs(t(i) - b)...
        *abs(t(i) - b)) + 2*fSIGNAL(c)*(1.17739188 ...
        - 1.50921205*abs(t(i) - c) ...
        + 0.64059253*abs(t(i) - c)*abs(t(i) - c) ...
        - 0.08992473*abs(t(i) - c)*abs(t(i) - c)...
        *abs(t(i) - c));
elseif t(i) == 2
    b = a - 1;
    c = a + 1;
    d = a + 2;
    nSIGNAL(i) = fSIGNAL(b)*0.21773426 ...
        + fSIGNAL(a)*0.56259136 ...
        + fSIGNAL(c)*0.21773426 ...
        + 2*fSIGNAL(d)*0.00097005;
elseif t(i) > 2 && t(i) < (size(xSIGNAL,1) - 1) ...
        && abs(t(i) - a) ~= 0
    b = a - 1;
    c = a + 1;
    d = a + 2;
    nSIGNAL(i) = fSIGNAL(b)*(1.17739188 ...
        - 1.50921205*abs(t(i) - b) ...
        + 0.64059253*abs(t(i) - b)*abs(t(i) - b) ...
        - 0.08992473*abs(t(i) - b)*abs(t(i) - b)...
        *abs(t(i) - b)) + fSIGNAL(a)*(0.56259136 ...
        + 0.04465421*abs(t(i) - a) ...
        - 0.70001154*abs(t(i) - a)*abs(t(i) - a) ...
        + 0.30938685*abs(t(i) - a)*abs(t(i) - a)...
        *abs(t(i) - a)) + fSIGNAL(c)*(0.56259136 ...
        + 0.04465421*abs(t(i) - c) ...
        - 0.70001154*abs(t(i) - c)*abs(t(i) - c) ...
        + 0.30938685*abs(t(i) - c)*abs(t(i) - c)...
```

```
        *abs(t(i) - c)) + fSIGNAL(d)*(1.17739188 ...
        - 1.50921205*abs(t(i) - d) ...
        + 0.64059253*abs(t(i) - d)*abs(t(i) - d) ...
        - 0.08992473*abs(t(i) - d)*abs(t(i) - d)...
        *abs(t(i) - d));
elseif t(i) > 2 && t(i) < (size(xSIGNAL,1) - 1) ...
        && abs(t(i) - a) == 0
    b = a - 2;
    c = a - 1;
    d = a + 1;
    e = a + 2;
    nSIGNAL(i) = fSIGNAL(b)*0.00097005 ...
        + fSIGNAL(c)*0.21773426 ...
        + fSIGNAL(a)*0.56259136 ...
        + fSIGNAL(d)*0.21773426 ...
        + fSIGNAL(e)*0.00097005;
elseif t(i) == (size(xSIGNAL,1) - 1)
    b = a - 2;
    c = a - 1;
    d = a + 1;
    nSIGNAL(i) = 2*fSIGNAL(b)*0.00097005 ...
        + fSIGNAL(c)*0.21773426 ...
        + fSIGNAL(a)*0.56259136 ...
        + fSIGNAL(d)*0.21773426;
elseif t(i) > (size(xSIGNAL,1) - 1) && t(i) < size(xSIGNAL,1)
    b = a - 1;
    c = a + 1;
    nSIGNAL(i) = 2*fSIGNAL(b)*(1.17739188 ...
        - 1.50921205*abs(t(i) - b) ...
        + 0.64059253*abs(t(i) - b)*abs(t(i) - b) ...
        - 0.08992473*abs(t(i) - b)*abs(t(i) - b)...
        *abs(t(i) - b)) + fSIGNAL(a)*(0.56259136 ...
        + 0.04465421*abs(t(i) - a) ...
        - 0.70001154*abs(t(i) - a)*abs(t(i) - a) ...
        + 0.30938685*abs(t(i) - a)*abs(t(i) - a)...
        *abs(t(i) - a)) + fSIGNAL(c)*(1.17739188 ...
        - 1.50921205*abs(t(i) - c) ...
        + 0.64059253*abs(t(i) - c)*abs(t(i) - c) ...
        - 0.08992473*abs(t(i) - c)*abs(t(i) - c)...
        *abs(t(i) - c));
elseif t(i) == size(xSIGNAL,1)
    b = a - 2;
    c = a - 1;
    nSIGNAL(i) = 2*fSIGNAL(b)*0.00097005 ...
        + 2*fSIGNAL(c)*0.21773426...
        + fSIGNAL(a)*0.56259136;
end
```

```
        end
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
end
```

Referências

BHALSHANKAR, S.; GULVE, A. K. Audio steganography: Lsb technique using a pyramid structure and range of bytes. *arXiv preprint arXiv:1509.02630*, 2015.

BLU, T.; THCVENAZ, P.; UNSER, M. Moms: Maximal-order interpolation of minimal support. *IEEE Transactions on Image Processing*, IEEE, v. 10, n. 7, p. 1069–1080, 2001.

BLU, T.; UNSER, M. Approximation error for quasi-interpolators and (multi-) wavelet expansions. *Applied and Computational Harmonic Analysis*, Elsevier, v. 6, n. 2, p. 219–251, 1999.

BRUCKNER, A. M.; BRUCKNER, J. B.; THOMSON, B. S. *Real analysis*. [S.l.]: ClassicalRealAnalysis.com, 1997.

CAVALCANTI, M. M.; CAVALCANTI, V. D. Introdução à teoria das distribuiçoes e aos espaços de sobolev. *[Introduction to distribution theory and Sobolev spaces] Editora da Universidade Estadual de Maringá (Eduem), Maringá*, p. 452, 2009.

CONDAT, L.; BLU, T.; UNSER, M. Beyond interpolation: Optimal reconstruction by quasi-interpolation. In: IEEE. *Image Processing, 2005. ICIP 2005. IEEE International Conference on*. [S.l.], 2005. v. 1, p. I–33.

CUNHA, A.; TEIXEIRA, R.; VELHO, L. Espaços de escala discretos. 2001.

DALAI, M.; LEONARDI, R.; MIGLIORATI, P. Efficient digital pre-filtering for least-squares linear approximation. In: SPRINGER. *International Workshop on Visual Content Processing and Representation*. [S.l.], 2005. p. 161–169.

FFT and Spectrogram. Disponível em <https://www.princeton.edu/ cuff/ ele201/files/spectrogram.pdf>. Online; acesso em 26 de out. de 2017.

FIGUEIREDO, D. G. d. *Análise I*. [S.l.]: LTC, 1996.

FIGUEIREDO, D. G. d. *Análise de Fourier e equações diferenciais parciais*. [S.l.]: Instituto de Matemática Pura e Aplicada: Rio de Janeiro, 2014.

GASQUET, C.; WITOMSKI, P. *Analyse de Fourier et application: Filtrage, calcul numérique, ondelettes*. [S.l.]: Paris: Masson, 1990.

GIANNAKOPOULOS, T.; PIKRAKIS, A. *Introduction to Audio Analysis: A MATLAB® Approach*. [S.l.]: Academic Press, 2014.

GUIDORIZZI, H. L. *Um curso de cálculo, vol. 4* . [S.l.]: Grupo Gen-LTC, 2000.

LIMA, E. L. Álgebra linear, 4ª edição. *Instituto de Matemática Pura e Aplicada.(Coleção Matemática Universitária). Rio de Janeiro*, 2000.

LIMA, E. L. Curso de análise, vol. 2.(6a ediçao). *Projeto Euclides, IMPA*, 2000.

LIMA, E. L. Curso de análise, volume 1. *Projeto Euclides, IMPA, décima primeira ediçao*, 2004.

MULLER, M. et al. Signal processing for music analysis. *IEEE Journal of Selected Topics in Signal Processing*, IEEE, v. 5, n. 6, p. 1088–1110, 2011.

OLIVEIRA, J. C. Cálculo iv. *mar. 2017 a jun. 2017*, 162 f. Notas de Aula, 2017.

OSGOOD, B. The fourier transform and its applications. *Lecture Notes for EE*, v. 261, p. 20, 2009.

SACHT, L.; NEHAB, D. Optimized quasi-interpolators for image reconstruction. *IEEE Transactions on Image Processing*, v. 24, n. 12, p. 5249–5259, 2015.

SACHT, L. K. *Optimized Quasi-interpolators for Image Reconstruction and Consistent Volumetric Discretizations Inside Self-Intersecting Surfaces*. Tese (Doutorado) — IMPA-Rio, 2014.

SHANNON, C. E. Communication theory of secrecy systems. *Bell Labs Technical Journal*, Wiley Online Library, v. 28, n. 4, p. 656–715, 1949.

UNSER, M.; ALDROUBI, A.; EDEN, M. Fast b-spline transforms for continuous image representation and interpolation. *IEEE Transactions on pattern analysis and machine intelligence*, v. 13, n. 3, p. 277–285, 1991.