Transformada de Hadamard

Manoel Marcelo da Silva
Instituto de Computação – Universidade Federal do Rio de Janeiro
manoelms@ic.ufrj.br

19 de julho de 2024

Resumo

Diferente da Transformada de Fourier, que é muito complexa, a Transformada de Hadamard traz uma simplicidade para estudar e analisar como uma transformada funciona e opera por conta de sua transformação extremamente simples que usa uma matriz simétrica e ortogonal de valores 1 e -1 para decompor um vetor em uma composição de sinais quadrados. O intuito desse projeto é demonstrar a sua utilidade e de outras transformadas mais complexas com ferramentas ferramentas básicas de álgebra linear.

1 Introdução

Sinais são basicamente uma forma física de transmitir uma informação. Podemos ver isso em ação, por exemplo, em áudios, vídeos e em quase tudo que analisamos na natureza. Por conta de sua relevância em quase todos os ramos de pesquisa, o processo de adquirir, processar e modificar esses sinais é de extrema importância para a humanidade. Não a toa, Gilbert Strang - autor extremamente renomado dos livros mais famosos de Álgebra Linear - afirmou que o algoritmo numérico mais importante de nossa vida é a Transformada Rápida de Fourier(FFT) [Strang(1994)], que é um algoritmo que decompõe qualquer sinal.

No entanto, a complexidade do Transformada de Fourier dificulta bastante a sua compreensão. Com isso, o intuito deste trabalho é mostrar uma variante da Transformada de Fourier que usa uma abordagem mais simples e intuitiva, que compartilha pontos-chave - como sua versão rápida, para uma maior acessibilidade.

2 Sinais

Como mostrado anteriormente, sinais são objetos em que necessitamos trabalhar para obtermos diversas informações da natureza. Geralmente ela pode ser considerada como uma função contínua em relação ao tempo como mostra a figura 1.

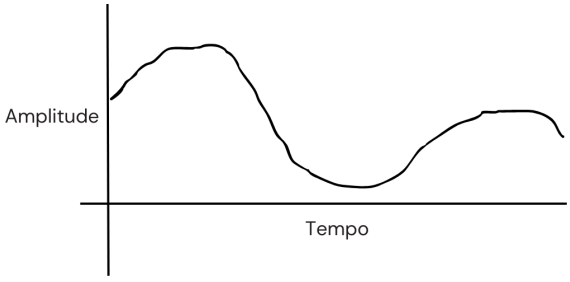

Figura 1: Exemplo de sinal.

No entanto, usualmente, não conseguimos armazenar esse sinal de forma contínua. Ao invés disso, lemos o sinal no meio com uma frequência fixa e o armazenamos, discretizando o sinal. Este sinal geralmente é armazenado em um vetor, em que cada o elemento é seu valor de amplitude e sua posição é o tempo, como vemos na figura 2.

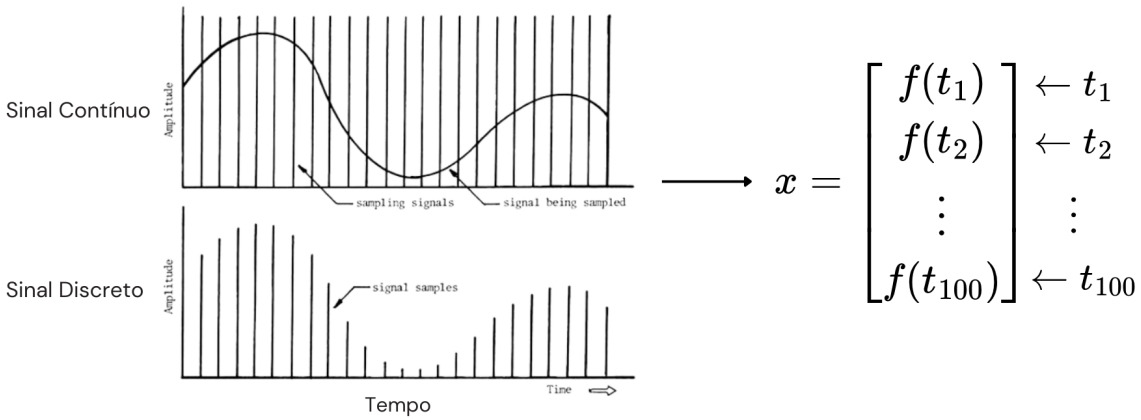

Figura 2: Discretizando o sinal.

Agora imagine a seguinte situação: Como poderíamos remover o ruído de um sinal que recebemos? Esse tipo de problema é bem comum quando trabalhamos com sinais, em que temos que descobrir uma combinação de sinais, que juntos, formam o nosso sinal original. Por exemplo, para remover o ruido, a partir da combinação de sinais, podemos remover aquelas de alta frequência, já que o ruído geralmente está nessas frequências.

3 Transformada de Hadamard (WHT)

Também conhecida como Walsh–Hadamard Transform (WHT) é uma classe de transformada de Fourier, sendo esta uma **transformação ortogonal não-senoidal e simétrica**, que diferente de Fourier, não usa números complexos. Essa transformação usa como sinais base, ou seja, os possíveis sinais que podem compor o sinal discreto original, com **apenas valores 1 ou -1 (ondas quadradas)**.

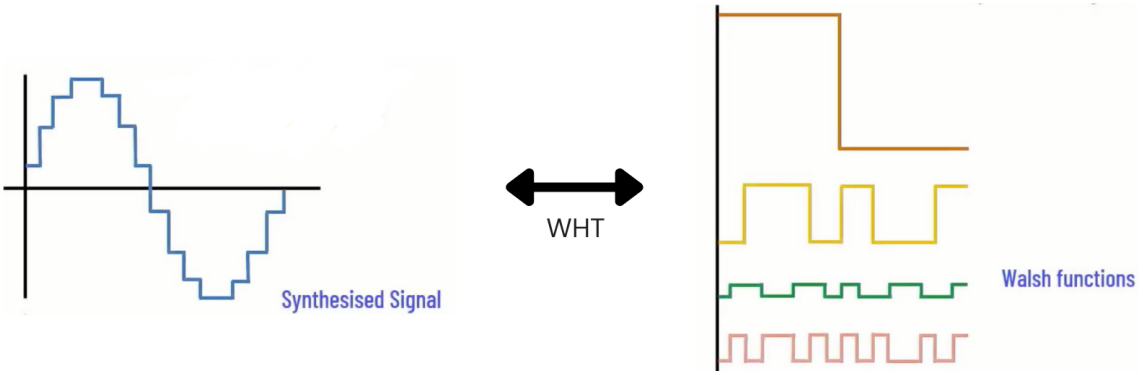

Figura 3: Decomposição em sinais quadrados.

Esses sinais base são conhecidos como funções de Walsh e por conta disso é bem mais simples, sendo uma das poucas transformadas que podem ser representadas de forma matricial usando números inteiros, de forma linear, que usa apenas **somas e subtrações**.

Outro ponto relevante é que, de forma similar à Transformada de Fourier, também possui uma versão acelerada de sua transformada, a tornando eficiente com um número bastante alto de elementos. Por conta disso, o WHT e o FWHT (Transformada Rápida de Hadamard) ganharam bastante relevância no processamento de sinais, principalmente na década de 60, já que somas e subtrações são operações "baratas"em uma implementação em hardware.

Por conta disso, ferramentas de compressão de imagens para sistemas que não possuem instruções de multiplicação usam Hadamard [mbitsnbites(2023)] - potencialmente sendo usada para a transmissão de imagens por sondas espaciais da NASA [Pratt et al.(1969)Pratt, Kane, and Andrews]. Além disso, é bastante usado na área de computação quântica e em criptografia.

3.1 Funções de Walsh

São os sinais em que usaremos como base para a transformação que formam um conjunto ortonormal de valores 1 ou -1. Tais sinais são bem similares à base usada na Transformada de Fourier, como vemos na figura 5.

Cada onda quadrada é distinguida por um parâmetro chamado *Sequência*, sendo este diretamente relacionado à frequência do sinal. O número de sequência é o número de vezes em que ocorre uma troca de sinal ao decorrer do tempo.

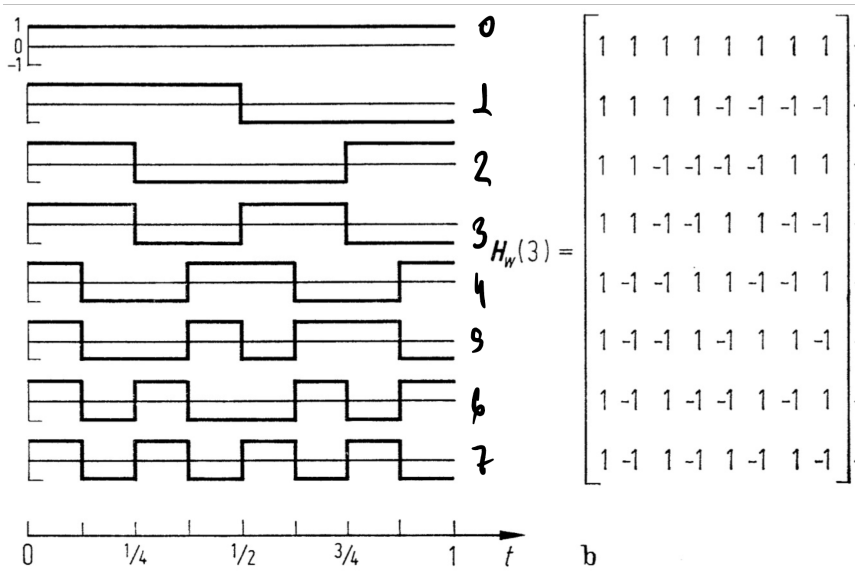

Figura 4: Funções de Walsh (a.Contínuo b.Discreto).

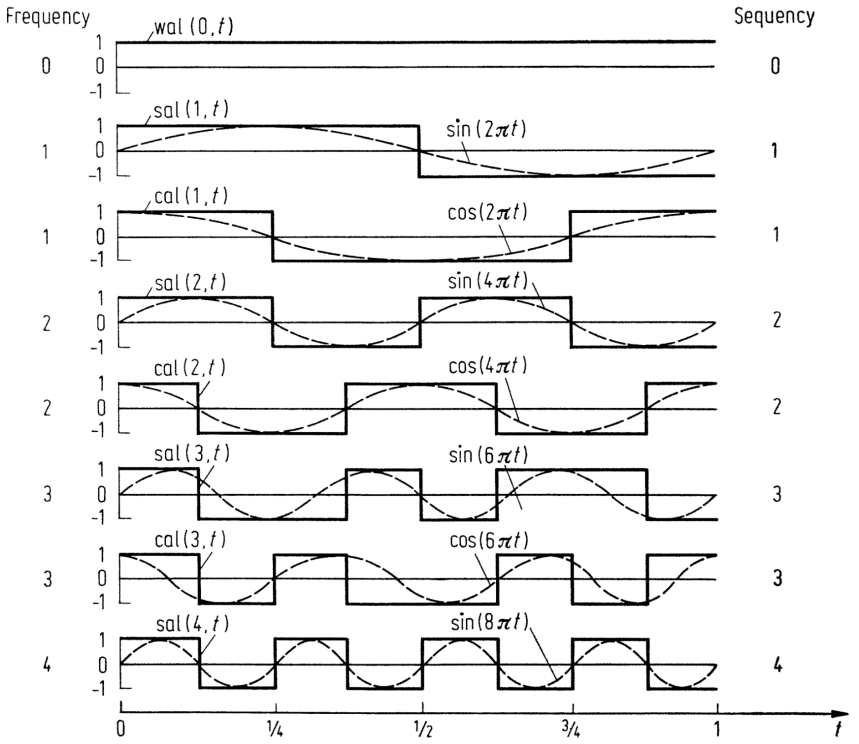

Figura 5: Funções de Walsh (N=8) e os Harmônicos de Fourier.

Apesar de poder ser gerada a partir de uma fórmula exponencial, o método que usaremos para criar a base para a nossa transformação é criando uma matriz de Hadamard - por isso o nome desta transformada.

3.2 Matriz de Hadamard

Uma forma extremamente fácil de se gerar essas funções, de forma discreta, é pela matriz de Hadamard. Uma matriz de Hadamard, por definição, é uma matriz *ortogonal* em que seus valores são apenas 1 e -1 para todo elemento pertencente nele. O método de Sylvester gera de forma recursiva uma matriz, em que cada linha/coluna é um desses sinais, que é base de nossa transformação vista anteriormente.

A recursão se dá por blocos de matriz, tal que $n = log_2(k)$:

$$H(0) = \begin{bmatrix} 1 \end{bmatrix}$$

$$H(n) = \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{bmatrix} = \begin{bmatrix} H_{n-1} & 0 \\ 0 & H_{n-1} \end{bmatrix} * \begin{bmatrix} I & I \\ I & -I \end{bmatrix}$$

Por exemplo:

$$H(1) = \begin{bmatrix} H_0 & H_0 \\ H_0 & -H_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H(2) = \begin{bmatrix} H_1 & H_1 \\ H_1 & -H_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

Como podemos perceber, tal recursão apenas nos dá números em que são potências de 2, ou seja, $k = log_2(n)$ para todo k inteiro. Isso nos limita no tamanho do vetor que podemos trabalhar, já que também deve seguir essa regra.

Por curiosidade, existem matrizes de Hadamard fora dessa recursão, porém a existência de alguns deles ainda não está provada, sendo parte de uma das conjecturas mais antigas da matemática ainda em aberto - a Conjectura de Hadamard.

Essa matriz possui três propriedades essenciais para fazermos uma transformada eficiente:

- Simétrica ($H = H^T$)
- Ortogonal ($HH^T = I$)
- $\frac{1}{2^n} H_n = H_n^{-1}$ (ou $\frac{1}{\sqrt{2}^n} H_n = \frac{1}{\sqrt{2}^n} H_n^{-1}$)

3.2.1 Simética

Para provarmos essa propriedade para todo valor de n, iremos fazer uma prova por indução.

Base:

$$H(0) = \begin{bmatrix} 1 \end{bmatrix}$$
$$\Downarrow$$
$$H(0) = H^T(0) = \begin{bmatrix} 1 \end{bmatrix}$$

Hipótese de Indução:

$$H_{n-1} = H_{n-1}^T$$

Indução:

$$H(n) = \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{bmatrix}$$
$$\Downarrow$$
$$H^T(n) = \begin{bmatrix} H_{n-1}^T & H_{n-1}^T \\ H_{n-1}^T & -H_{n-1}^T \end{bmatrix}$$
$$\Downarrow HI$$
$$H_n = H_n^T$$

■

3.2.2 Ortogonalidade

De forma similar à prova anterior, também faremos a prova por indução.

Base:

$$H(0) = \begin{bmatrix}1\end{bmatrix}$$
$$\Downarrow$$
$$H(0)H^T(0) = \begin{bmatrix}1\end{bmatrix} = I$$

Hipótese de Indução:

$$H_{n-1}H_{n-1}^T = 2^{n-1} * I$$

Indução:

$$H(n) = \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{bmatrix}$$
$$\Downarrow$$
$$H_nH_n^T = \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{bmatrix} \begin{bmatrix} H_{n-1}^T & H_{n-1}^T \\ H_{n-1}^T & -H_{n-1}^T \end{bmatrix}$$
$$\Downarrow$$
$$H_nH_n^T = \begin{bmatrix} 2H_{n-1}H_{n-1}^T & 0 \\ 0 & 2H_{n-1}H_{n-1}^T \end{bmatrix}$$
$$\Downarrow HI$$
$$H_nH_n^T = \begin{bmatrix} 2^nI & 0 \\ 0 & 2^nI \end{bmatrix}$$
$$\Downarrow$$
$$H_nH_n^T = 2^nI$$

■

3.2.3 Inversa

Usando as duas provas anteriores, com a definição de ortogonalidade e simetria, fica provado que $\frac{1}{2^n}H_n = H_n^{-1}$ (ou $\frac{1}{\sqrt{2^n}}H_n = \frac{1}{\sqrt{2^n}}H_n^{-1}$).

3.3 Algoritmo

Como a nossa matriz de Hadamard que criamos usando o método de Sylvester possui as funções de Walsh necessárias para a decomposição do sinal, podemos usá-la para encontrar os coeficientes desses sinais quadrados, que juntos formam o sinal original. Usando a matriz de Hadamard, basta fazer uma multiplicação matriz-vetor para encontrar os coeficientes.

Para fazer a volta, ou seja, a partir dos coeficientes encontrados calcular o x, basta fazer a inversa - que é a mesma Matriz de Hadamard por conta da propriedade da simetria e ortogonalidade. Assim, fica com uma complexidade $O(n^2)$, bem mais rápido que fazer um sistema linear.

O WHT de um vetor x de tamanho n ($k = log_2(n)$) é definido por uma multiplicação matriz-vetor, sendo b o vetor de coeficientes:

$$b = \frac{1}{\sqrt{n}}H(k)x$$

É valido salientar que a ordem de sequência não está ordenado na matriz de Hadamard, logo teria que fazer uma conversão.

E a volta é dada por:

$$x = \frac{1}{\sqrt{n}} H(k)b$$

Para evitar a normalização com números irracionais, usualmente é feito apenas uma multiplicação em uma das etapas, ficando:

$$b = \frac{1}{n} H(k)x$$

$$x = H(k)b$$

É possível perceber que, por conta da Matriz de Hadamard ter apenas números 1 e -1, é feito apenas somas e subtrações - desconsiderando a normalização. Isso favorece a velocidade do algoritmo, já que é uma operação muito mais barata que multiplicações e divisões em processadores.

3.3.1 Exemplo 1

Considere o sinal mostrado na figura 6 e que queremos fazer a Transformada de Hadamard.

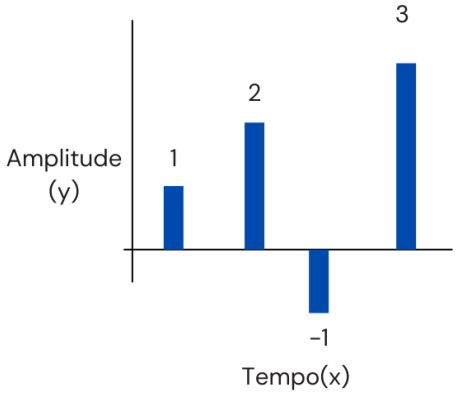

Figura 6: Exemplo de sinal discretizado, com $n = 4$

Percebemos que $x = \begin{bmatrix} 1 & 2 & -1 & 3 \end{bmatrix}^T$, logo:

$$b = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ -1 \\ 3 \end{bmatrix}$$

$$4b = \begin{bmatrix} 1+2-1+3 \\ 1-2-1-3 \\ 1+2+1-3 \\ 1-2+1+3 \end{bmatrix}$$

$$b = \begin{bmatrix} 5/4 \\ -5/4 \\ 1/4 \\ 3/4 \end{bmatrix}$$

Na volta, com o b em mãos, queremos descobrir o x:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 5/4 \\ -5/4 \\ 1/4 \\ 3/4 \end{bmatrix}$$

$$x = \begin{bmatrix} 5/4 - 5/4 + 1/4 + 3/4 \\ 5/4 + 5/4 + 1/4 - 3/4 \\ 5/4 - 5/4 - 1/4 - 3/4 \\ 5/4 + 5/4 - 1/4 + 3/4 \end{bmatrix}$$

$$x = \begin{bmatrix} 1 \\ 2 \\ -1 \\ 3 \end{bmatrix}$$

Observando os sinais base, podemos analisar sua composição:

$$w_0 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} (f = 0)$$
$$w_1 = \begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix} (f = 3)$$
$$w_2 = \begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix} (f = 1)$$
$$w_3 = \begin{bmatrix} 1 & -1 & -1 & 1 \end{bmatrix} (f = 2)$$

$$w = (5/4)w_0 + (-5/4)w_1 + (1/4)w_2 + (3/4)w_3$$

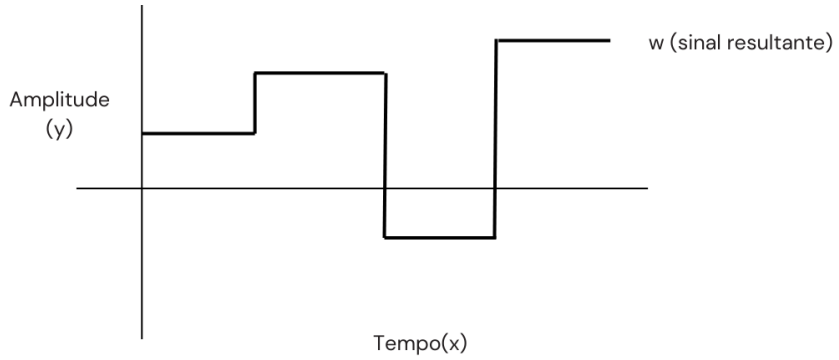

Figura 7: Sinal Resultante

3.3.2 Exemplo 2

Considere um sinal discretizado x com $n = 8$: $\begin{bmatrix} 19 & -1 & 11 & -9 & 7 & 13 & -15 & 5 \end{bmatrix}^T$

$$b = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \frac{1}{8} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 19 \\ -1 \\ 11 \\ -9 \\ 7 \\ 13 \\ -15 \\ 5 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 4 \\ 0 \\ 3 \\ 10 \\ 0 \\ 0 \end{bmatrix}$$

Logo, x é apenas composto dos sinais não-nulos:

$$w = (2)w_0 + (4)w_2 + (3)w_4 + (10)w_5$$

4 Transformada Rápida de Hadamard (FWHT)

Como vimos anteriormente, a complexidade desta transformada até esse momento é $O(n^2)$, porém, assim como a Transformada de Fourier, esta também possui sua versão rápida. Da mesma forma que o FFT, a simetria (e a ortogonalidade) nos mostram que podemos diminuir o número de contas. Estamos repetindo operações desnecessariamente.

A forma em que iremos mostrar tal propriedade é a partir do particionamento de matrizes, mostrando que é possível diminuir o número de operações. Tal ideia é similar ao raciocínio praticado em algoritmos como o QuickSort, conhecido como "Dividir e Conquistar".

4.1 Dividir e Conquistar

Considere um vetor x de tamanho n ($k = log_2(n)$):

$$H_k x = b$$

Iremos particionar, de forma recursiva, a multiplicação matricial:

$$H_k x = \begin{bmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$\Updownarrow$$

$$H_{k-1}x_1 + H_{k-1}x_2 = b_1$$
$$H_{k-1}x_1 - H_{k-1}x_2 = b_2$$

$$\Updownarrow$$

$$H_{k-1}(x_1 + x_2) = b_1 \qquad (x_1 + x_2) = x'_1$$
$$H_{k-1}(x_1 - x_2) = b_2 \qquad (x_1 - x_2) = x'_2$$

Com isso, fazendo os cálculos de cada bloco do vetor b de forma recursiva, estamos ganhando alguma coisa? Para isso, iremos analisar o pseudocódigo desse algoritmo.

4.2 Algoritmo

Algorithm 1: Transformada Rápida de Hadamard

Data: Vetor x de tamanho k, tal que $k = 2^n$
Result: Vetor b de tamanho k

FWHT x**:**

$\begin{bmatrix} x_{1_{(k/2)}} \\ x_{2_{(k/2)}} \end{bmatrix} \leftarrow x_{(k)}$; Dividindo o Vetor em 2 partes iguais

$b'_1 \leftarrow FWHT(x_1)_{(k/2)}$ ▷ $O(FWHT(2^{k-1}))$

$b'_2 \leftarrow FWHT(x_2)_{(k/2)}$ ▷ $O(FWHT(2^{k-1}))$

$b_1 \leftarrow \frac{b'_1 + b'_2}{2}$ ▷ $O(k/2)$

$b_2 \leftarrow \frac{b'_1 - b'_2}{2}$ ▷ $O(k/2)$

return $\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$

Logo, a complexidade desse algoritmo é:

$$O(2^n) = O(2^{n-1}) + O(2^{n-1}) + O(n/2) + O(n/2)$$

$$O(2^n) = 2O((2^{n-1}) + O(n)$$

Usando 2^n como k, temos:

$$O(2^n) = 2O(2^{n-1}) + 2^n$$
$$O(k) = 2O(k/2) + k$$

Mas qual é o valor disso? Para isso, iremos provar por indução que é igual a $k * log_2(k)$

4.3 Complexidade

Provando que: $O(2^n) = 2O(2^{n-1}) + 2^n = 2^n * n$

Base: $(n = 1)$

$$O(2^1) = 2^1 * 1 = 2$$

Hipótese de Indução:

$$O(2^n) = 2^n * n$$

Indução:

$$O(2^{n+1}) = 2O(2^n) + 2^{n+1}$$
$$\Downarrow HI$$
$$O(2^n) = 2(2^n * n) + 2^{n+1}$$
$$\Downarrow$$
$$O(2^n) = 2^{n+1} * n + 2^{n+1}$$
$$\Downarrow$$
$$O(2^n) = 2^{n+1}(n+1)$$

■

Logo, considerando 2^n como k, temos:

$$O(k) = k * log_2(k)$$

Isso prova que o FWHT possui complexidade $O(n) = n * log_2(n)$ ao invés de n^2, como na versão anterior. O ganho disso é enorme quando usamos entradas muito grandes, como vemos na figura 8.

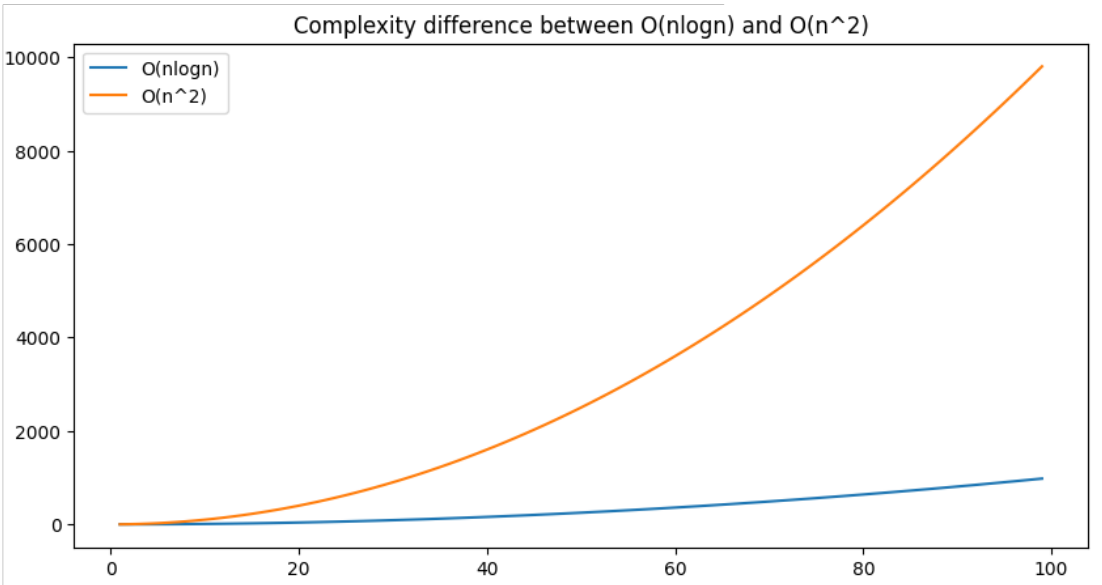

Figura 8: Complexidade $n * log_2(n)$ X n^2

Tal ganho também pode ser demonstrado no grafo abaixo de um FWHT de vetor x com 8 elementos.

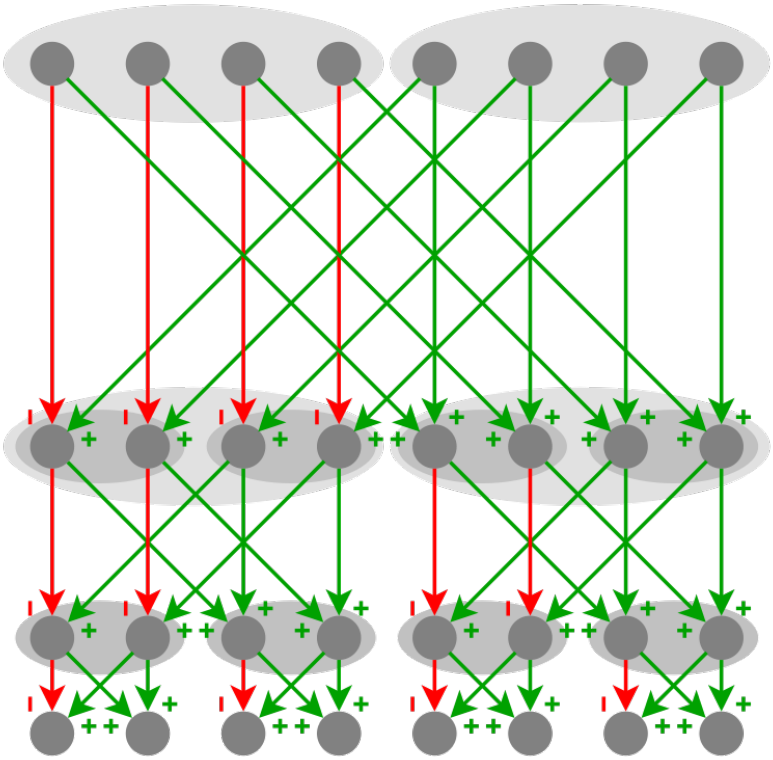

Figura 9: Grafo representando as operações em um FWHT

5 Aplicações

Uma aplicação básica do FWHT e do WHT é a remoção de ruídos em um sinal. Como foi explicado anteriormente, os ruidos geralmente se encontram nas composições de sinal de maior frequência. Assim, ao fazer a decomposição de sinais, é possível remover a parte ruidosa do sinal.

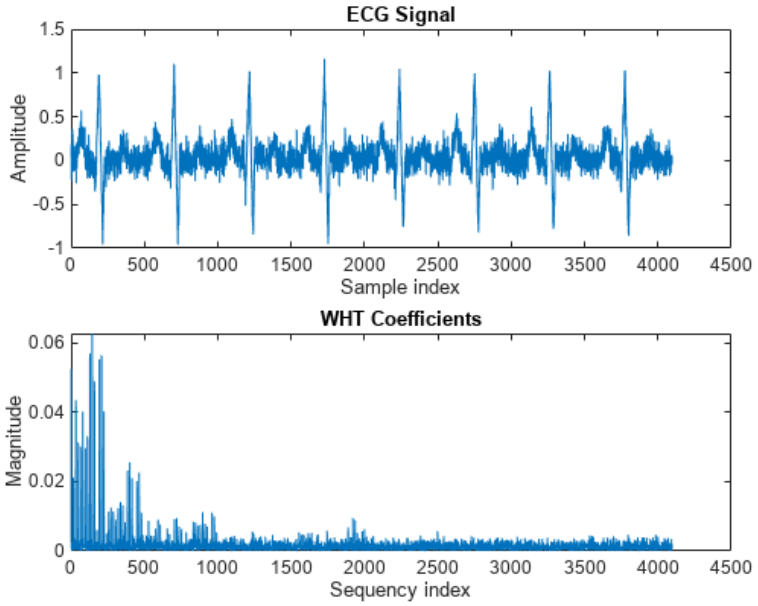

Figura 10: Eletrocardiograma ruidoso e seus coeficientes do WHT ordenados

Uma das aplicações mais importantes desse conceito é na remoção do ruído de um eletrocardiograma(ECG), que inicialmente é ruidoso por conta dos sensores, que são bastante sensíveis. Para isso, com o sinal recebido (que já vem discretizado), aplicamos o FWHT neste vetor x - que precisa ser uma potência de 2, podendo reduzir para o número mais próximo.

Após ordenar os sinais base de acordo com seu número de sequência - que é diretamente proporcional à sua frequência - podemos "cortar"esses sinais com maior frequência para um número potência de 2 menor que o original. Nesse caso iremos recompor com os 1024 maiores sinais, de 4096 originalmente. O resultado, como mostra a figura 11, é extremamente satisfatório, restaurando a característica original do sinal.

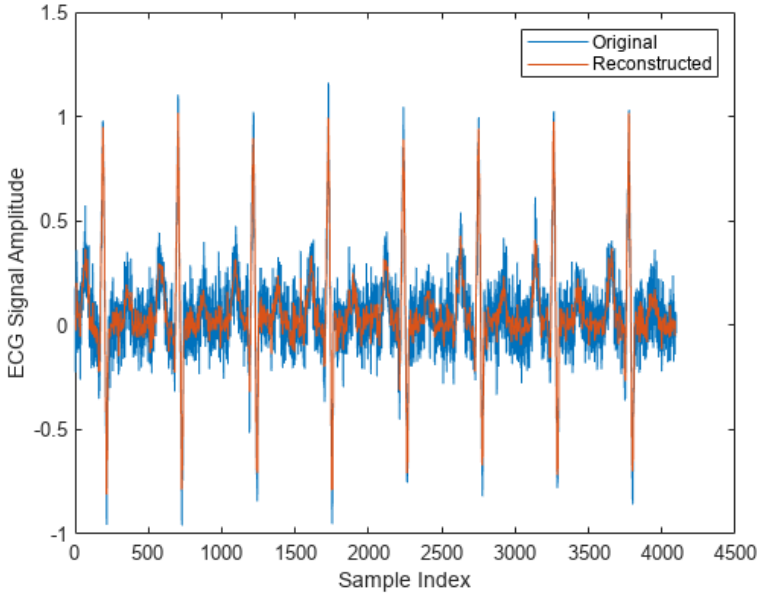

Figura 11: Resultado após a remoção da parte ruidosa

Referências

[Ahmed and Rao(2012)] N. Ahmed and K. R. Rao. *Orthogonal transforms for digital signal processing*. Springer Science & Business Media, 2012.

[E. Whelchel(1968)] F. Q. E. Whelchel. The fast fourier-hadamard transform and its use in signal representation and classification, 1968.

[Lee and Kaveh(1986)] M. Lee and M. Kaveh. Fast hadamard transform based on a simple matrix factorization. *IEEE transactions on acoustics, speech, and signal processing*, 34(6):1666–1667, 1986.

[MATLAB(2010)] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[mbitsnbites(2023)] mbitsnbites. Himg - hadamard compressed image, 2023. URL https://gitlab.com/mbitsnbites/himg.

[Pratt et al.(1969)Pratt, Kane, and Andrews] W. Pratt, J. Kane, and H. Andrews. Hadamard transform image coding. *Proceedings of the IEEE*, 57(1):58–68, 1969. doi: 10.1109/PROC.1969.6869.

[Strang(1994)] G. Strang. Wavelets. *American Scientist*, 82(3):250–255, 1994. ISSN 00030996. URL http://www.jstor.org/stable/29775194.

[Technologies(2021)] E. Technologies. Walsh hadamard transform (signal filtering image compression), 2021. URL https://www.exptech.co.in/2021/03/video-44-walsh-hadamard-transform.html.

[upobir(2019)] upobir. Fast walsh hadamard transforms and it's inner workings, 2019. URL https://codeforces.com/blog/entry/71899.