



MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES  
**INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS**

## **LISTA FOURIER 1 - CAP-384**

Leonardo Sattler Cassara

Lista de Exercícios apresentada aos professores Margarete Domingues e Luciano Magrini como parte da avaliação do curso CAP-384.

Repositório desta lista:  
[github/CAP-384](https://github.com/LeonardoSattler/CAP-384)

INPE  
São José dos Campos  
09 de outubro de 2020

## PREFÁCIO

Os códigos desta lista utilizam a linguagem Python. As análises foram realizadas com a biblioteca `numpy`, em particular com a rotina `numpy.fft`, que implementa a transformada discreta de Fourier através de um algoritmo de transformada rápida de Fourier. As visualizações foram geradas com a biblioteca `matplotlib`. Todos os códigos e as imagens que eles geram estão organizados na pasta `scripts` do repositório deste manuscrito. Os arquivos estão separados por exercício conforme descrito abaixo.

- pasta **exercicio2**: contém o seguinte script:
  - `ft_linear.py`: gera um gráfico ilustrando a linearidade da Transformada de Fourier. Quaisquer duas funções  $f_1$  e  $f_2$  podem ser declaradas. Ele gera uma figura com 6 gráficos: três das funções  $f_1$ ,  $f_2$  e  $f_3 = f_1 + f_2$ , e outros três de suas Transformadas de Fourier.
- pasta **exercicio3**: contém scripts para ilustração e estudo de algumas propriedades da Transformada de Fourier. São eles:
  - `ft_scaling_exp.py`: script que explora a função  $e^{-|t|}$  e sua transformada de modo a ilustrar as propriedades de *time scaling* e *frequency scaling*.
  - `ft_scaling_rect.py`: script que gera um pulso retangular de altura e largura definidos, bem como sua transformada, de modo a ilustrar as propriedades de *time scaling* e *frequency scaling*.
  - `ft_shifting.py`: script que explora uma função trigonométrica qualquer e sua transformada, de modo a ilustrar as propriedades de *time shifting* e *frequency shifting*.
- pasta **exercicio5**: contém os scripts para estudo das propriedades de simetria da Transformada de Fourier. São eles:
  - `ft_symmetries_trig.py`: script que, a partir de uma função cosseno (uma função par) e outra seno (uma função ímpar), reais e complexas, ilustra a natureza de suas transformadas (se par ou ímpar, real ou complexa).
  - `ft_symmetries_poly.py`: script equivalente ao `ft_symmetries_trig.py`, porém explora duas funções polinomiais:  $f_1(t) = t^2$  (uma função par) e  $f_2(t) = t^3$  (uma função ímpar).

A correta compilação deste manuscrito depende das imagens presentes na pasta `scripts`. Os scripts Python podem ser executados de qualquer local. Os exercícios marcados como *In prep.* estão em preparação e serão adicionados a este manuscrito em breve através de atualizações do seu repositório.

## LISTA DE ABREVIATURAS E SIGLAS

FT	– do inglês, <b>F</b> ourier <b>T</b> ransform, ou Transformada de Fourier
IFT	– do inglês, <b>I</b> nverse <b>F</b> ourier <b>T</b> ransform, ou Transformada Inversa de Fourier
FFT	– do inglês, <b>F</b> ast <b>F</b> ourier <b>T</b> ransform, ou Transformada Rápida de Fourier
DFT	– do inglês, <b>D</b> iscrete <b>F</b> ourier <b>T</b> ransform, ou Transformada Discreta de Fourier

## SUMÁRIO

	<u>Pág.</u>
<b>Exercício 1 . . . . .</b>	<b>1</b>
1.1 . . . . .	1
1.2 . . . . .	2
<b>Exercício 2 . . . . .</b>	<b>3</b>
<b>Exercício 3 . . . . .</b>	<b>4</b>
3.1 . . . . .	4
3.2 . . . . .	6
<b>Exercício 4 . . . . .</b>	<b>7</b>
<b>Exercício 5 . . . . .</b>	<b>8</b>
<b>Exercício 6 . . . . .</b>	<b>12</b>
<b>Exercício 7 . . . . .</b>	<b>14</b>
<b>Exercício 8 . . . . .</b>	<b>16</b>
<b>Exercício 9 . . . . .</b>	<b>18</b>
<b>Exercício 10 . . . . .</b>	<b>19</b>
<b>REFERÊNCIAS BIBLIOGRÁFICAS . . . . .</b>	<b>20</b>

## Exercício 1

### 1.1

Mostrarei que

$$\int f(t)\overline{g(t)}dt = \frac{1}{2\pi} \int \hat{f}(\xi)\overline{\hat{g}(\xi)}d\xi.$$

**Resolução:**

Pelas definições de Transformada de Fourier,

$$\text{FT}: \hat{f}(\xi) = \int_{-\infty}^{+\infty} f(t)e^{-i\xi t}dt, \quad (1)$$

e Transformada Inversa de Fourier,

$$\text{IFT}: f(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \hat{f}(\xi)e^{i\xi t}d\xi, \quad (2)$$

podemos escrever (abandonando os limites de integração por redundância):

$$\begin{aligned} \int f(t)\overline{g(t)}dt &= \int \left( \frac{1}{2\pi} \int \hat{f}(\xi)e^{i\xi t}d\xi \right) \left( \frac{1}{2\pi} \int \overline{\hat{g}(\xi')}e^{-i\xi' t}d\xi' \right) dt \\ &= \left( \frac{1}{2\pi} \right)^2 \int \int \hat{f}(\xi)\overline{\hat{g}(\xi')} \left( \int e^{i(\xi-\xi')t}dt \right) d\xi' d\xi. \end{aligned}$$

A última expressão entre parênteses acima pode ser reescrita pois ela é a função delta:

$$\int_{-\infty}^{+\infty} e^{i(\xi-\xi')t}dt = 2\pi\delta(\xi - \xi').$$

Substituindo esse resultado:

$$\begin{aligned} \int f(t)\overline{g(t)}dt &= \frac{1}{2\pi} \int \hat{f}(\xi) \left( \int \overline{\hat{g}(\xi')} \delta(\xi - \xi') d\xi' \right) d\xi \\ &= \frac{1}{2\pi} \int f(\xi) \overline{g(\xi)} dt. \end{aligned} \quad (\text{Q.E.D.})$$

Neste último passo, utilizou-se a propriedade geral da função delta:  $\int_{-\infty}^{+\infty} F(\xi') \delta(\xi - \xi') d\xi' = F(\xi)$ .

## 1.2

Mostrarei que, considerando a relação de Parseval, a IFT pode ser escrita como:

$$f(t) = \int_{-\infty}^{+\infty} \hat{f}(\xi) e^{\imath \xi t} dt.$$

**Resolução:**

Da Eq. 2:

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \hat{f}(\xi) e^{\imath \xi t} d\xi.$$

Mas, conforme a relação de Parseval, a norma  $\mathbb{L}^2(\mathbb{R})$  se conserva entre o espaço não transformado e no de Fourier, ou seja,

$$\int_{-\infty}^{+\infty} |f(t)|^2 dt = \frac{1}{2\pi} \int_{-\infty}^{+\infty} |\hat{f}(\xi)|^2 d\xi,$$

portanto:

$$f(t) = \int_{-\infty}^{+\infty} \hat{f}(\xi) e^{\imath \xi t} dt. \quad (\text{Q.E.D.})$$

## Exercício 2

Mostrarei que

$$\text{FT}[\alpha f + g](t) = \alpha \text{FT}[f(t)] + \text{FT}[g(t)].$$

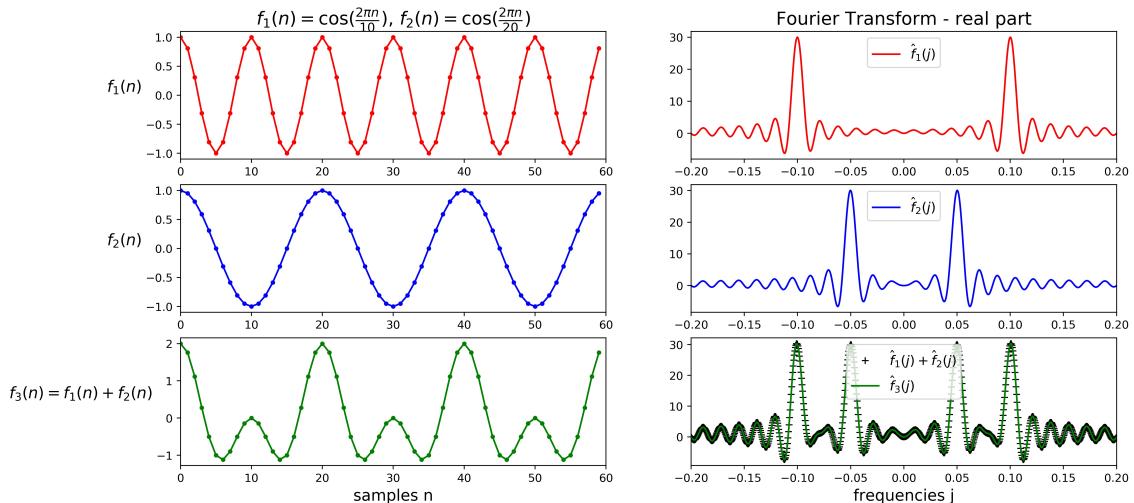
**Resolução:**

Da Eq. 1:

$$\begin{aligned} \text{FT}[\alpha f + g](t) &= \frac{1}{2\pi} \int_{-\infty}^{+\infty} (\alpha \hat{f}(\xi) + \hat{g}(\xi)) e^{i\xi t} d\xi \\ &= \alpha \frac{1}{2\pi} \int_{-\infty}^{+\infty} \hat{f}(\xi) e^{i\xi t} d\xi + \frac{1}{2\pi} \int_{-\infty}^{+\infty} \hat{g}(\xi) e^{i\xi t} d\xi \\ &= \alpha \text{FT}[f(t)] + \text{FT}[g(t)]. \end{aligned} \quad (\text{Q.E.D.})$$

A Figura 2.1 abaixo ilustra a linearidade da FT, que é o significado da resolução acima. Duas funções cosseno, uma de período igual a dez ( $f_1$ ) e outra de período igual a vinte ( $f_2$ ) são somadas para gerar uma terceira função ( $f_3$ ). A parte real de suas Transformadas de Fourier ( $\hat{f}_1$ ,  $\hat{f}_2$  e  $\hat{f}_3$ ) estão à direita, e o último plot (canto direito inferior) atesta que  $\hat{f}_3 = \hat{f}_1 + \hat{f}_2$ .

Figura 2.1: Exemplo da linearidade da FT.



## Exercício 3

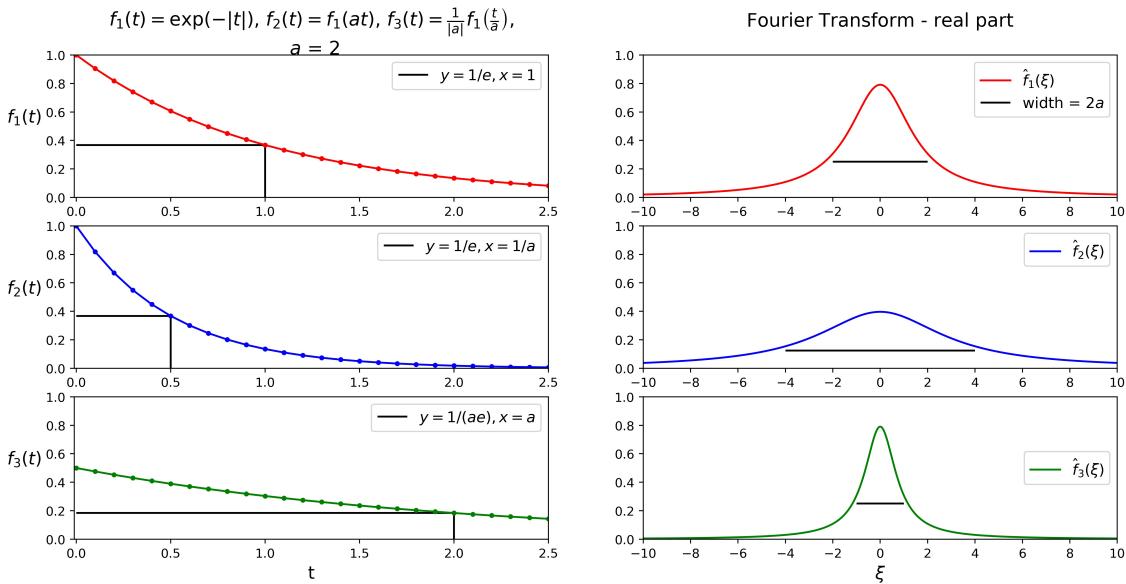
### 3.1

As Figuras 3.1 e 3.2 a seguir ilustram as propriedades de *time scaling* e *frequency scaling*:

$$f(at) \Leftrightarrow \frac{1}{|a|} \hat{f}\left(\frac{\xi}{a}\right) \quad \text{time scaling}$$

$$\frac{1}{|a|} f\left(\frac{t}{a}\right) \Leftrightarrow \hat{f}(a\xi) \quad \text{frequency scaling}$$

Figura 3.1: Exemplo de *time scaling* e *frequency scaling* sobre a função  $f_1(t) = e^{-|t|}$ .

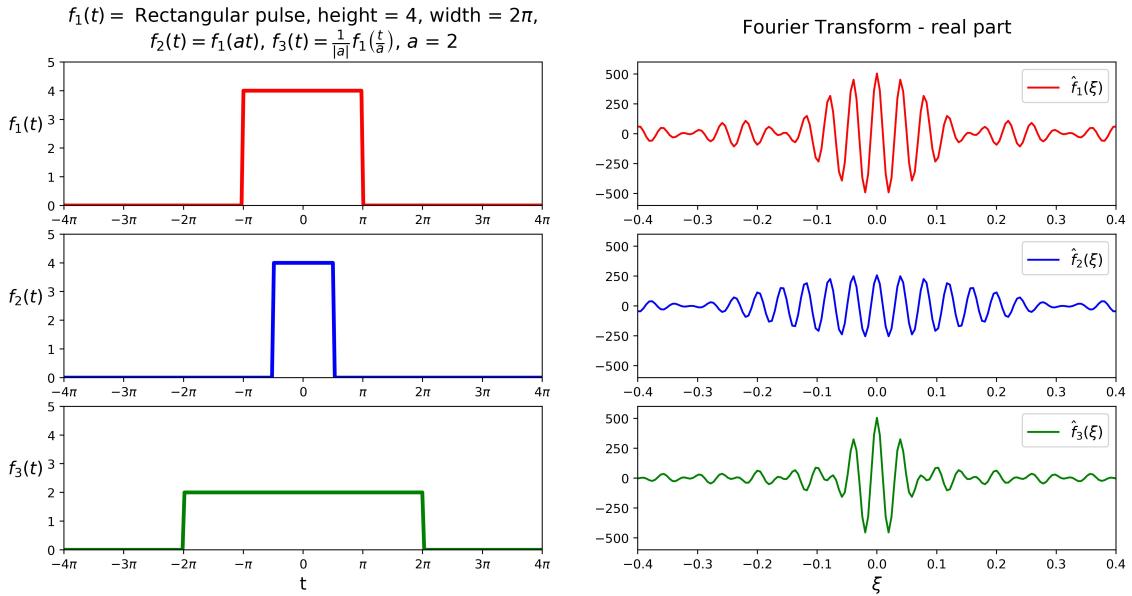


A Figura 3.1 explora a função  $f_1(t) = e^{-|t|}$  (topo à esquerda), cuja transformada é  $\hat{f}(\xi) = 2/(\xi^2 + 1)$  (topo à direita). Também na esquerda, gráficos da função original e de suas consequentes funções noutras escalas:  $f_2(t) = f_1(at)$  (meio) e  $f_3(t) = f_1(t/a)/|a|$  (abaixo). A escolha de  $a = 2$  foi feita. Os valores de  $x$  quando  $y = 1/e$  (ou  $y = 1/(|a|e)$  no caso de  $f_3(t)$ ) ilustram o efeito da escala nas funções  $f_2(t)$  e  $f_3(t)$  com relação à função original  $f_1(t)$ . À direita, suas respectivas transformadas de Fourier também exibem o efeito da escala, porém no domínio da frequência, com a largura à meia altura da transformada sinalizada em linha preta. O gráfico de

$\hat{f}_2(\xi)$  (meio à direita) tem largura à meia altura que é o dobro de  $\hat{f}_1(\xi)$ , enquanto que sua altura é a metade deste. A largura à meia altura de  $\hat{f}_3(\xi)$  (abaixo à direita) é metade da de  $\hat{f}_1(\xi)$ , enquanto que sua altura é igual à deste. Ou seja, em comparação a  $\hat{f}_1(\xi)$ ,  $\hat{f}_2(\xi)$  foi achatao e  $\hat{f}_3(\xi)$  foi comprimido.

Essa análise foi realizada com a rotina `numpy.fft.hfft`, que computa a transformada de um input com simetria Hermitiana, ou seja, gera um espectro real. As diferentes alturas da transformada resultante não foi quantitativamente analisada pois depende da normalização do algoritmo implementado. O output desta rotina não é normalizada, portanto foi aplicada a normalização  $1/\sqrt{n}$ . Deste modo, a análise relativa (alturas de  $\hat{f}_2(\xi)$  e  $\hat{f}_3(\xi)$  com relação à altura de  $\hat{f}_1(\xi)$ ) permite verificação das propriedades sob estudo sem perda de generalidade.

Figura 3.2: Exemplo de *time scaling* e *frequency scaling* sobre a função de pulso retangular com altura 4 e largura  $2\pi$ .



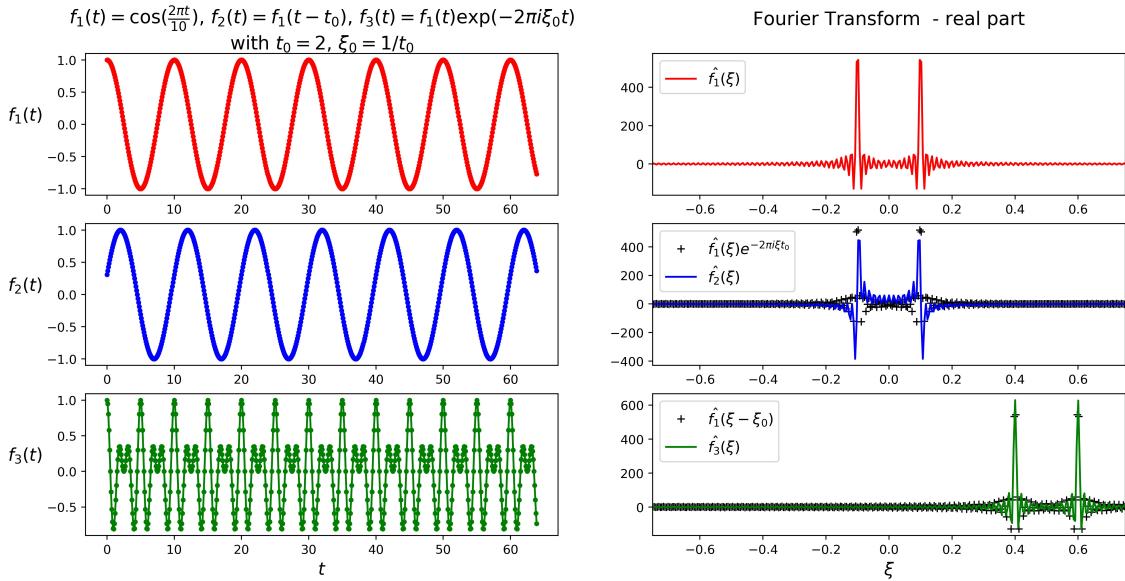
A Figura 3.2 exemplifica as propriedades de *time* e *frequency scaling* a partir de uma função pulso retangular. A altura do pulso original (topo à esquerda) é igual a quatro e a largura é  $2\pi$ . Os demais gráficos da esquerda são sinais sob diferentes efeitos da escala  $a = 2$ , e na direita estão suas respectivas transformadas. Assim como na Figura 3.1, os efeitos de achatar, esticar e comprimir o sinal no domínio do tempo são evidentes: achatá-lo no tempo estica-o na frequência (*time scaling*), enquanto que esticá-lo e comprimi-lo no tempo mantém sua magnitude porém achata-o na frequência (*frequency scaling*).

### 3.2

A Figura a seguir ilustra as propriedades de *time shifting* e *frequency shifting*:

$$\begin{aligned} f(t - t_0) &\Leftrightarrow \hat{f}(\xi)e^{2\pi i \xi t_0} && \text{time shifting} \\ f(t)e^{2\pi i \xi_0 t} &\Leftrightarrow \hat{f}(\xi - \xi_0) && \text{frequency shifting} \end{aligned}$$

Figura 3.3: Exemplo de *time shifting* e *frequency shifting* sobre a função  $\cos(2\pi t/10)$ .



No topo à esquerda a função  $\cos(2\pi t/10)$  é graficada com sua transformada de Fourier à direita. Também à esquerda estão a função  $f_2(t)$  (meio) e  $f_3(t)$  (abaixo) que representam as propriedades de *time shifting* e *frequency shifting*, respectivamente. Suas transformadas de Fourier à direita ( $\hat{f}_2(\xi)$  no meio e  $\hat{f}_3(\xi)$  abaixo) são graficadas junto com pontos de ajuste da função  $\hat{f}_1(\xi)$  com sinais de '+' na cor preta. Os pontos resultantes estão sobrepostos às transformadas das funções  $f_2(t)$  e  $f_3(t)$ , evidenciando as propriedades de *time* e *frequency shifting*.

## Exercício 4

Mostrarei que (comutatividade da convolução):

$$[f \star g] = [g \star f].$$

**Resolução:**

Da definição de convolução, podemos escrever a convolução da função  $f$  com a função  $g$  como:

$$(f \star g)(t) = \int_{-\infty}^{+\infty} f(u)g(t-u)du.$$

Fazendo  $\tau = t - u$ :

$$\begin{aligned}(f \star g)(t) &= - \int_{t+\infty}^{t-\infty} f(t-\tau)g(\tau)d\tau \\&= - \int_{+\infty}^{-\infty} f(t-\tau)g(\tau)d\tau \\&= \int_{-\infty}^{+\infty} f(t-\tau)g(\tau)d\tau \\&= (g \star f).\end{aligned}\tag{Q.E.D.}$$

## Exercício 5

Verificarei a propriedade de simetria nos domínios tempo e frequência. Ela é importante pois a Análise de Fourier aproxima funções por uma soma de senos e cossenos. Portanto, funções pares necessitam apenas os termos dos cossenos, enquanto funções ímpares apenas os termos da soma associados à função seno. Isso ocorre pois, em cada caso, a outra metade dos coeficientes é igual a zero. Sendo assim, explorar simetrias pode ser importante para poupar esforço (tempo) computacional.

### Resolução:

Uma função geral pode ser escrita como a soma de uma função par  $f_e$  (ou em inglês, *even*) e outra ímpar  $f_o$  (ou em inglês, *odd*):

$$f(x) = f_e(x) + f_o(x).$$

Uma função par é tal que  $f(x) = f(-x)$ , ou seja, ela é simétrica com relação ao eixo das ordenadas. Uma função ímpar é tal que  $f(x) = -f(-x)$ , ou seja, ela é antissimétrica. Além disso, tem-se que  $\int_{-\infty}^{\infty} f_e(x)dx = 2 \int_0^{\infty} f_e(x)dx$ , ao passo que  $\int_{-\infty}^{\infty} f_o(x)dx = 0$ . Por último, a multiplicação de funções pares por funções ímpares é tal que:  $f_e \times g_e = h_e$ ,  $f_o \times g_o = h_e$ , e  $f_e \times g_o = h_o$ .

A partir destas propriedades de uma função geral, e da propriedade de linearidade da Transformada de Fourier (verificada no Exercício 2), podemos escrever a Eq. 1 como:

$$\begin{aligned}\hat{f}(\xi) &= \int_{-\infty}^{+\infty} f(t)e^{-i\xi t}dt \\ &= \int_{-\infty}^{+\infty} f_e(x)e^{-i\xi t}dt + \int_{-\infty}^{+\infty} f_o(x)e^{-i\xi t}dt \\ &= \int_{-\infty}^{+\infty} f_e(x) \cos(\xi t)dt - i \underbrace{\int_{-\infty}^{+\infty} f_e(x) \sin(\xi t)dt}_0 + \\ &\quad \underbrace{\int_{-\infty}^{+\infty} f_o(x) \cos(\xi t)dt}_0 - i \int_{-\infty}^{+\infty} f_o(x) \sin(\xi t)dt \\ &= 2 \int_0^{+\infty} f_e(x) \cos(\xi t)dt - 2i \int_0^{+\infty} f_o(x) \sin(\xi t)dt.\end{aligned}$$

Similarmente, podemos escrever para uma função complexa  $f(x) = f_{re}(x) + i f_{im}(x)$ :

$$\begin{aligned}
\hat{f}(\xi) &= \int_{-\infty}^{+\infty} f(t) e^{-i\xi t} dt \\
&= \int_{-\infty}^{+\infty} f_{re}(x) e^{-i\xi t} dt + i \int_{-\infty}^{+\infty} f_{im}(x) e^{-i\xi t} dt \\
&= \int_{-\infty}^{+\infty} f_{re}(x) \cos(\xi t) dt - i \int_{-\infty}^{+\infty} f_{re}(x) \sin(\xi t) dt + \\
&\quad i \int_{-\infty}^{+\infty} f_{im}(x) \cos(\xi t) dt + i \int_{-\infty}^{+\infty} f_{im}(x) \sin(\xi t) dt.
\end{aligned}$$

Dessa maneira, a depender da natureza da função  $f(x)$ , o produto de funções em cada uma das quatro integrais somadas acima resultará numa função par ou ímpar, de modo que somente uma destas integrais será diferente de zero. A tabela abaixo resume os possíveis resultados.

Tabela 5.1: Propriedades de simetria nos domínios tempo e frequência.

$f(x)$ é	Integral restante	O resultado de $\hat{f}(\xi)$ é
real e par	1 <sup>a</sup>	real e par
real e ímpar	2 <sup>a</sup>	imaginária e ímpar
imaginária e par	3 <sup>a</sup>	imaginária e par
imaginária e ímpar	4 <sup>a</sup>	real e ímpar

As Figuras a seguir ilustram as propriedades da Tabela 5.1.

A Figura 5.1 exibe as funções trigonométricas reais, uma par  $f_1(t)$  (gráfico do topo à esquerda com a função cosseno) e outra ímpar  $f_2(t)$  (gráfico de baixo à esquerda com a função seno), ambas de período igual a 10. Suas respectivas transformadas à direita,  $\hat{f}_1(t)$  (real e par) e  $\hat{f}_2(t)$  (imaginária e ímpar), atestam as propriedades da Tabela 5.1. A Figura 5.2 complementa as informações da tabela exibindo as mesmas funções porém puramente imaginárias (multiplicadas pelo número imaginário  $i = \sqrt{-1}$ ). Nesse caso,  $\hat{f}_1(t)$  é imaginária e par enquanto  $\hat{f}_2(t)$  é real e ímpar. As Figuras 5.3 e 5.4 refazem a análise das Figuras 5.1 e 5.2, respectivamente, para duas novas funções: uma polinomial par  $f_1(t) = t^2$ , e outra polinomial ímpar  $f_2(t) = t^3$ . Suas

respectivas transformadas fomentam a análise implementada durante a resolução deste exercício, pois também ilustram as propriedades de simetria da transformada de Fourier presentes na Tabela 5.1.

Figura 5.1: Transformada de Fourier de duas funções trigonométricas reais, uma par e outra ímpar. A transformada da primeira é real e par, já da segunda é imaginária e ímpar.

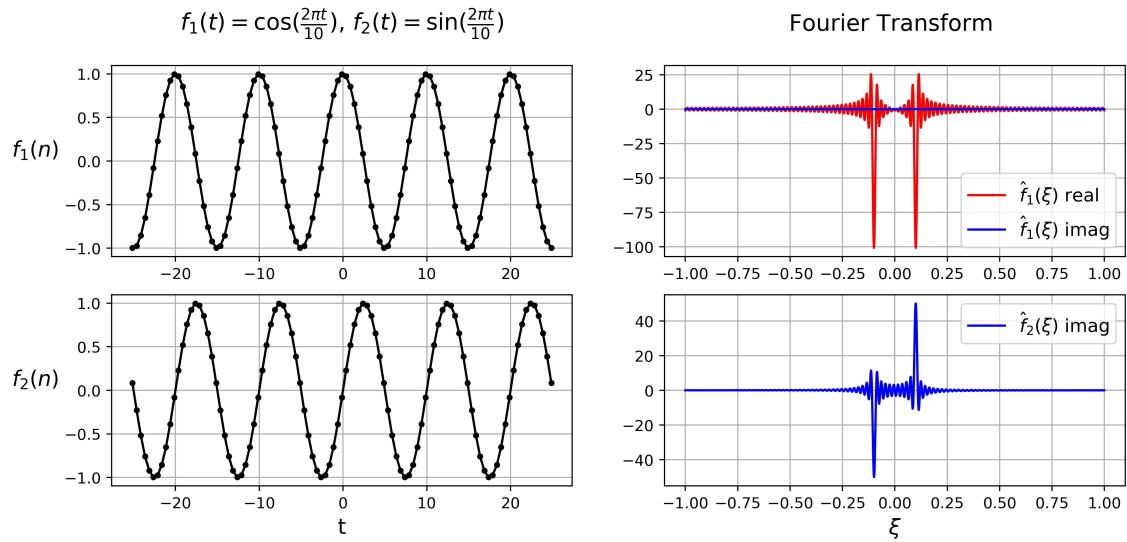


Figura 5.2: Transformada de Fourier de duas funções trigonométricas imaginárias, uma par e outra ímpar. A transformada da primeira é imaginária e par, já da segunda é real e ímpar.

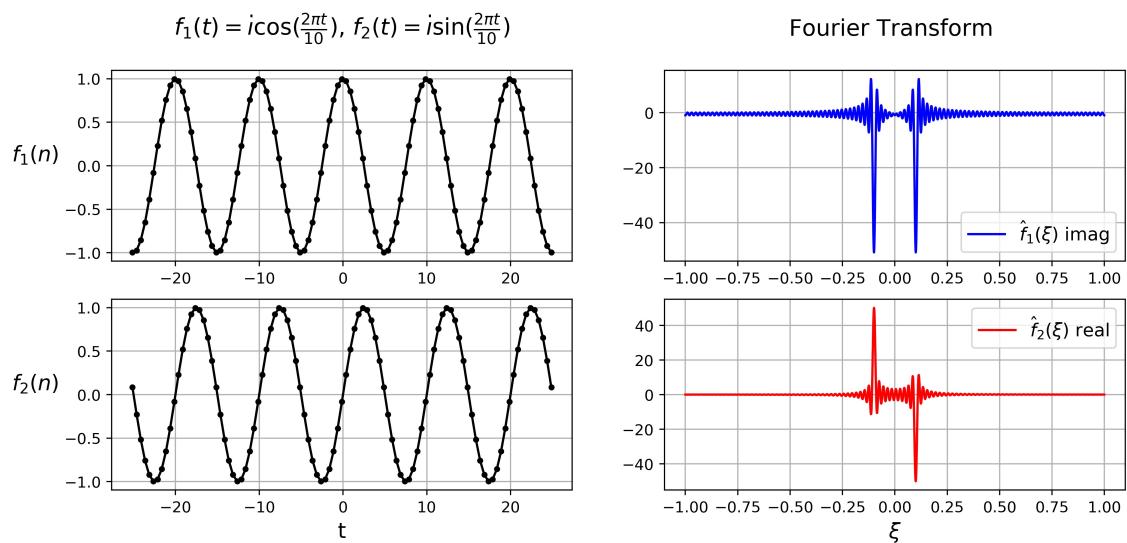


Figura 5.3: Transformada de Fourier de duas funções polinomiais reais, uma par e outra ímpar. A transformada da primeira é real e par, já da segunda é imaginária e ímpar.

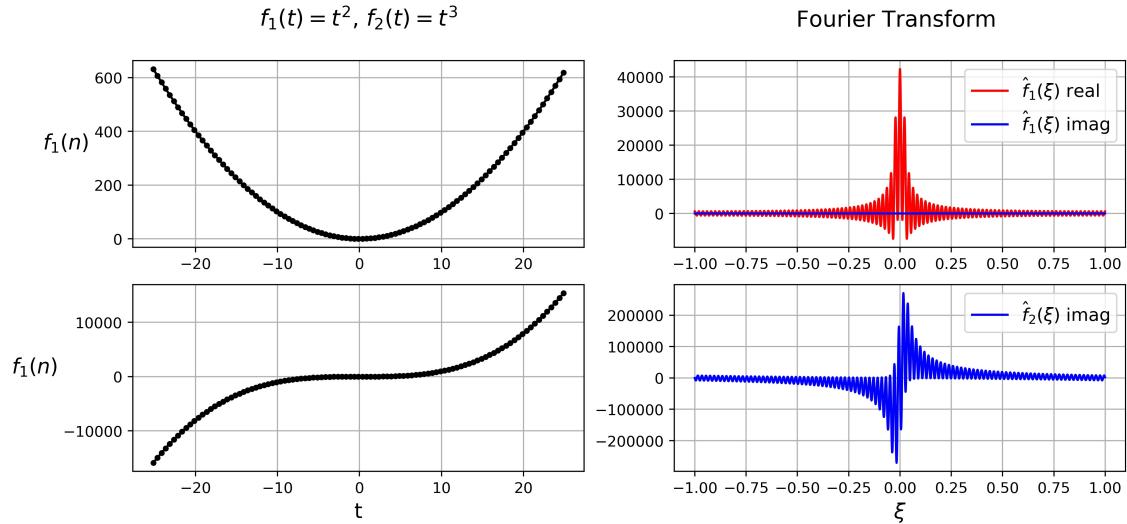
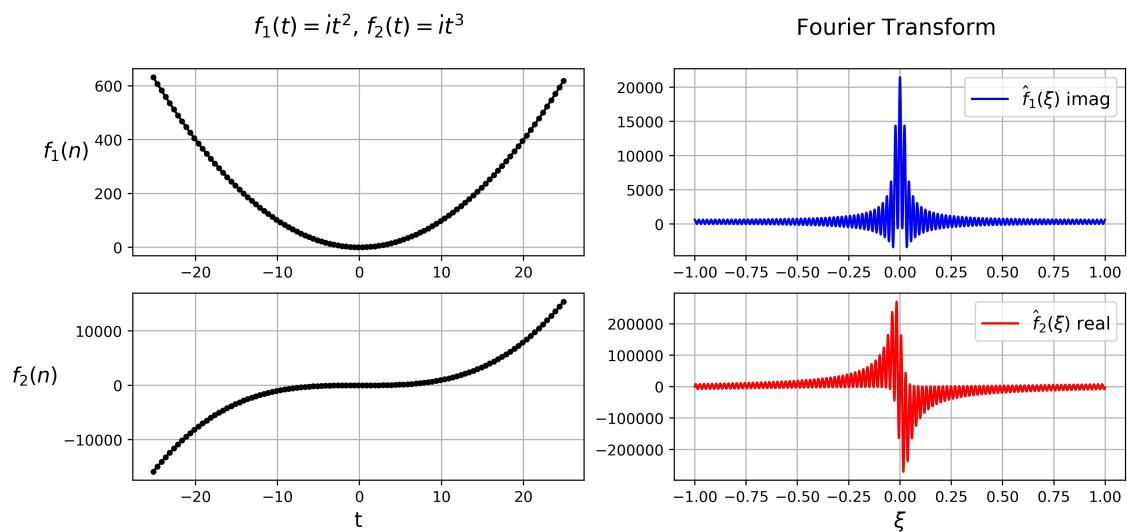


Figura 5.4: Transformada de Fourier de duas funções polinomiais imaginárias, uma par e outra ímpar. A transformada da primeira é imaginária e par, já da segunda é real e ímpar.



## Exercício 6

Apresentarei a decomposição da matriz da DFT para um input de tamanho 8 (por exemplo, um sinal representado por um vetor de números reais):

$$F_8 = \begin{bmatrix} W_8^0 & W_8^0 \\ W_8^0 & W_8^1 & W_8^2 & W_8^3 & W_8^4 & W_8^5 & W_8^6 & W_8^7 \\ W_8^0 & W_8^2 & W_8^4 & W_8^6 & W_8^8 & W_8^{10} & W_8^{12} & W_8^{14} \\ W_8^0 & W_8^3 & W_8^6 & W_8^9 & W_8^{12} & W_8^{15} & W_8^{18} & W_8^{21} \\ W_8^0 & W_8^4 & W_8^8 & W_8^{12} & W_8^{16} & W_8^{20} & W_8^{24} & W_8^{28} \\ W_8^0 & W_8^5 & W_8^{10} & W_8^{15} & W_8^{20} & W_8^{25} & W_8^{30} & W_8^{35} \\ W_8^0 & W_8^6 & W_8^{12} & W_8^{18} & W_8^{24} & W_8^{30} & W_8^{36} & W_8^{42} \\ W_8^0 & W_8^7 & W_8^{14} & W_8^{21} & W_8^{28} & W_8^{35} & W_8^{42} & W_8^{49} \end{bmatrix},$$

onde  $W_N = e^{-2\pi i/N}$  é a  $N$ -ésima raiz de um.

**Resolução:**

De modo geral, pode-se decompor a matriz da DFT de tamanho  $2N \times 2N$  no seguinte produto de matrizes:

$$F_{2N} = \begin{bmatrix} I_N & D_N \\ I_N & -D_N \end{bmatrix} \begin{bmatrix} F_N & \\ & F_N \end{bmatrix} P_{2N},$$

onde  $I_N$  é a matriz identidade de tamanho  $N \times N$ ,  $D_N$  é a matriz diagonal com entradas  $1, W, \dots, W^{N-1}$ , ou seja, potências consecutivas de  $W$ :

$$D_N = \begin{bmatrix} 1 & & & & \\ & W & & & \\ & & W^2 & & \\ & & & \ddots & \\ & & & & W^{N-1} \end{bmatrix},$$

e  $P_{2N}$  é a matriz de permutação de tamanho  $2N \times 2N$ :

$$P_{2N} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ & & & & \vdots & & \\ 0 & 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ & & & & \vdots & & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix},$$

que organiza o vetor de input em termos pares e ímpares.

Desse modo, a decomposiçao de  $F_8$  é:

$$F_4 = \begin{bmatrix} I_4 & D_4 \\ I_4 & -D_4 \end{bmatrix} \begin{bmatrix} F_4 & \\ & F_4 \end{bmatrix} P_8,$$

onde  $F_4$  é:

$$F_8 = \begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^4 & W_4^6 \\ W_4^0 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix}.$$

Portanto, se temos um sinal com oito amostras, representado pelo vetor  $\mathbf{f}$ , e estamos interessados em calcular sua transformada  $\hat{\mathbf{f}}$ , podemos realizar a seguinte operação:

$$\hat{\mathbf{f}} = F_8 \mathbf{f}.$$

Se ao invés do sinal nossos dados forem de um vetor com informações no domínio da frequênciia, podemos realizar a operação inversa para obter de volta o sinal:

$$\begin{aligned} \hat{\mathbf{f}} &= F_8 \mathbf{f} \\ F_8^H \hat{\mathbf{f}} &= F_8^H F_8 \mathbf{f} = \mathbf{f} \therefore \\ \mathbf{f} &= F_8^H \hat{\mathbf{f}}, \end{aligned}$$

onde  $F_8^H$  denota o conjugado Hermitiano (ou conjugado transposto) da matriz  $F_8$ .

## Exercício 7

Mostrarei, por indução, a complexidade  $\mathcal{O}(n \log_2 n)$  de um algoritmo.

**Resolução:**

O algoritmo FFT funciona dividindo a soma discreta da DFT de um input de tamanho  $n = 2^k$  em duas somas: uma em  $n/2$  termos pares e outra em  $n/2$  termos ímpares. Esse procedimento pode ser implementado recursivamente, de modo que a cada passo de divisão pela metade teremos de computar  $n$  termos através de somas distintas.

Um algoritmo com essa característica, que precisa realizar  $n$  operações antes, durante e após dividi-las em duas metades, pode ser expresso pela seguinte equação de recorrência ( $T_n$  denota o valor de uma função  $T(n)$  de complexidade de tempo em função do tamanho do input  $n$ )

$$\begin{cases} T_n = 2T_{\frac{n}{2}} + n, & n \geq 2 \\ T_1 = 0. \end{cases}$$

Reescrevendo  $T_n$ :

$$\begin{aligned} T_n &= T_{2^k} = 2T_{2^{k-1}} + 2^k \\ \frac{T_{2^k}}{2^k} &= \frac{2T_{2^{k-1}} + 2^k}{2^k} = \frac{2T_{2^{k-1}}}{2^k} + \frac{2^k}{2^k} \\ \frac{T_{2^k}}{2^k} &= \frac{T_{2^{k-1}}}{2^{k-1}} + 1 \\ \frac{T_{2^k}}{2^k} &= \frac{T_{2^{k-2}}}{2^{k-2}} + 1 + 1 \\ &\vdots \\ \frac{T_{2^k}}{2^k} &= \frac{T_{2^0}}{2^0} + \dots + 1 + 1 \\ \frac{T_{2^k}}{2^k} &= 0 + \dots + 1 + 1 \\ \frac{T_{2^k}}{2^k} &= k. \end{aligned}$$

Lembrando que  $n = 2^k$ , temos que  $k = \log_2 n$ , e a última relação pode ser reescrita:

$$\frac{T_{2^k}}{2^k} = \frac{T_n}{n} = \log_2 n$$
$$T_n = n \log_2 n \quad (\text{Q.E.D.})$$

## Exercício 8

Relatório:

### Algoritmo FFT de Cooley-Tukey e Abordagens Similares

O algoritmo Fast Fourier Transform (FFT) é um algoritmo eficiente para calcular a transformada discreta de Fourier. Para um input de tamanho  $n$ , este algoritmo calcula a transformada de Fourier em tempo  $\mathcal{O}(n \log_2 n)$ . Ele foi primeiro discutido por [Cooley e Tukey \(1965\)](#), que o popularizou e ofereceu uma implementação (doravante chamada C-T). Aparentemente, o algoritmo por eles apresentado já havia sido inventado por Gauss em 1865 ([STRANG, 1993](#)). Ele funciona para qualquer input que pode ser reescrito como  $n = n_1 n_2$  ao expressar a transformada discreta sobre o input  $n$  em termos menores de tamanhos  $n_1$  e  $n_2$ , que por sua vez podem ser divididos em partes menores, e assim sucessivamente (e recursivamente) até que se chegue a fatores primos.

Dado um input  $n$ , diversas escolhas de sua fatoração são possíveis. Usualmente, ou  $n_1$  ou  $n_2$  é um fator constante pequeno e limitado (chamado de *radix*). Se  $n_1 = radix$  o procedimento é conhecido como DIT (Decimation in Time), e se  $n_2 = radix$ , DIF (Decimation in Frequency). Quando o input  $n$  é uma potência de dois, por exemplo  $n = 2^k$ , o *radix-2* DIT pode ser implementado, que é a versão mais simples e comum do algoritmo C-T. Ela divide a transformada discreta de tamanho  $n$  em duas transformadas intercaladas de tamanho  $n/2$  em cada estágio recursivo. As implementações conhecidas como *radix-4* e *radix-8*, para inputs que são potência de quatro e oito, respectivamente, reduzem o número de passos intermediários (multiplicações complexas) e podem ser de 20%-30% mais rápidas que a *radix-2*. Por sua vez, o algoritmo *split-radix* utiliza tanto o *radix-2* quanto o *radix-4* em sua implementação, oferecendo a eficiência do *radix-4* para inputs que são potência de dois.

Além das diferentes fatorações de  $n$ , a transposição do input para o output requer rearranjo de dados que pode ser executada de diferentes maneiras. Isso permite novas abordagens. Por isso as FFT's são ainda hoje muito estudadas, e o limite máximo de otimização um assunto debatido. Buscando eficiência computacional global e de casos específicos (e.g.,  $n$  primo), diversos algoritmos propõem novas estratégias de implementação do algoritmo original de C-T.

O prime-factor algorithm FPA (algoritmo de fatoração de primos) também conhe-

cido como FFT de Goot-Thomas ([GOOD, 1958](#)), pode ser usado como uma implementação FFT eficiente no caso de  $n = n_1 n_2$  sendo  $n_1$  e  $n_2$  mutuamente primos. Ou seja, enquanto o algoritmo de C-T pode ser usado para quaisquer fatores, o PFA está restrito a fatores mutuamente primos. Além disso, a manipulação dos dados de entrada é mais complexa e se baseia no Teorema chinês do resto ([WEISSTEIN, -a](#)). Outro algoritmo que explora a fatoração de primos é a FFT de Winograd, que pode ser usada para  $n = 2, 34, 5, 7, 8, 11, 13$  e  $16$  ([WEISSTEIN, -b](#)).

Já para  $n$  primo, o algoritmo de Rader é uma FFT que computa a transformada discreta de Fourier ao reescrevê-la como uma convolução cíclica ([RADER, 1968](#)). Outro algoritmo que realiza procedimento similar é a FFT de Bluestein, porém este não está restrito a  $n$  primo. A FFT de Winograd citada acima utiliza dos princípios do algoritmo de Rader para o caso de  $n$  primo. A FFT de Rader pode ser otimizada por um fator de dois no caso de inputs reais, mas assim como os demais algoritmos de FFT citados até aqui, sua complexidade é  $\mathcal{O}(n \log_2 n)$ .

Se aplicarmos as diferentes FFT's existentes ao mesmo input, obteremos (obviamente) o mesmo output. Entretanto, cada algoritmo difere na maneira em que os resultados intermediários são armazenados e alterados. Isso motiva abordagens de otimização não somente sobre o software, mas também sobre o hardware, de modo a explorar vetorização, otimização de acessos à cache e paralelização. Em particular, a subrotina FFTW ([FRIGO; JOHNSON, 1998](#)) em C é amplamente usada. Além de automaticamente selecionar a FFT mais efetiva baseando-se na natureza do input, ela explora a arquitetura da máquina implementando otimizações durante a compilação (e.g., *loop unrolling*). Desde a versão 1.1, a FFTW inclui uma subrotina para transformada paralela de Fourier, compatível com sistemas de memória compartilhada e de memória distribuída.

Algoritmos *radix-2* e *radix-4* podem ser implementados em paralelo ao executar cada uma das sub-transformadas discretas (sobre os diferentes fatores de  $n$ ) ao mesmo tempo. Já o *split-radix*, que é composto dos algoritmos *radix-2* e *radix-4*, pode ser dividido em dois processos independentes para cada um destes. Estes, por sua vez, podem ser implementados paralelamente, configurando o cenário de máxima utilização do hardware paralelo. Para estas e outras implementações em paralelo de transformadas baseadas em C-T, ver [Balducci et al. \(1996\)](#). Outra abordagem existente é a de [Swarztrauber \(1984\)](#), que oferece uma adaptação do C-T e de outros algoritmos de FFT para computadores com vetorização.

## **Exercício 9**

Entregue em 13/10/2020.

## **Exercício 10**

*In prep.*

## REFERÊNCIAS BIBLIOGRÁFICAS

- BALDUCCI, M.; CHOUDARY, A.; HAMAKER, J. Comparative analysis of fft algorithms in sequential and parallel form. In: **Mississippi State University Conference on Digital Signal Processing**. [S.l.: s.n.], 1996. p. 5–16. 17
- COOLEY, J. W.; TUKEY, J. W. An algorithm for the machine calculation of complex fourier series. **Mathematics of computation**, JSTOR, v. 19, n. 90, p. 297–301, 1965. 16
- FRIGO, M.; JOHNSON, S. G. Fftw: An adaptive software architecture for the fft. In: **IEEE. Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181)**. [S.l.], 1998. v. 3, p. 1381–1384. 17
- GOOD, I. J. The interaction algorithm and practical fourier analysis. **Journal of the Royal Statistical Society: Series B (Methodological)**, Wiley Online Library, v. 20, n. 2, p. 361–372, 1958. 16
- RADER, C. M. Discrete fourier transforms when the number of data samples is prime. **Proceedings of the IEEE**, IEEE, v. 56, n. 6, p. 1107–1108, 1968. 17
- STRANG, G. Wavelet transforms versus fourier transforms. **Bulletin of the American Mathematical Society**, v. 28, n. 2, p. 288–305, 1993. 16
- SWARZTRAUBER, P. N. Fft algorithms for vector computers. **Parallel Computing**, Elsevier, v. 1, n. 1, p. 45–63, 1984. 17
- WEISSTEIN, E. W. **Chinese Remainder Theorem**. [S.l.]: From MathWorld—A Wolfram Web Resource., –.  
<https://mathworld.wolfram.com/ChineseRemainderTheorem.html>. 17
- \_\_\_\_\_. **Fast Fourier Transform**. [S.l.]: From MathWorld—A Wolfram Web Resource., –. <https://mathworld.wolfram.com/FastFourierTransform.html>. 17