TI0119 - PROCESSAMENTO DIGITAL DE SINAIS

TRANSFORMADA RÁPIDA DE FOURIER
2º Avaliação Parcial

Otacílio Bezerra Leite Neto
Universidade Federal do Ceará
Departamento de Engenharia de Teleinformática
minhotmog@gmail.com

1 Exercícios - Página 13

Faça a convulação entre duas janelas de largura diferente, com valores não-nulos unitários. Uso algoritmos rápidos e o cálculo de convolução discreta para comparação.

Resposta:

Demonstra-se abaixo os resultados obtidos para e exercício proposto. A explicação matemática e sobrea a implementação dos algoritmos está detalhada no final deste documento. Para efeitos práticos, consideramos as duas janelas de largura diferente através das seguências $x_1[n]$ e $x_2[n]$ definidas como

$$x_1[n] = \begin{cases} 1 & \text{se } 0 \leq n < 6 \\ 0 & \text{caso contrário} \end{cases}; \qquad x_2[n] = \begin{cases} 1 & \text{se } 0 \leq n < 12 \\ 0 & \text{caso contrário} \end{cases}. \tag{1.1}$$

A segunda janela é definida como tendo o dobro de largura da primeira. Para realização dos experimentos, considerou-se um número total de $N = 2^5 = 32$ amostras discretas consecutivas dessa sequência. Uma visualização das duas sequências, no domínio do tempo discreto, é exibida na Figura 1.1 abaixo.

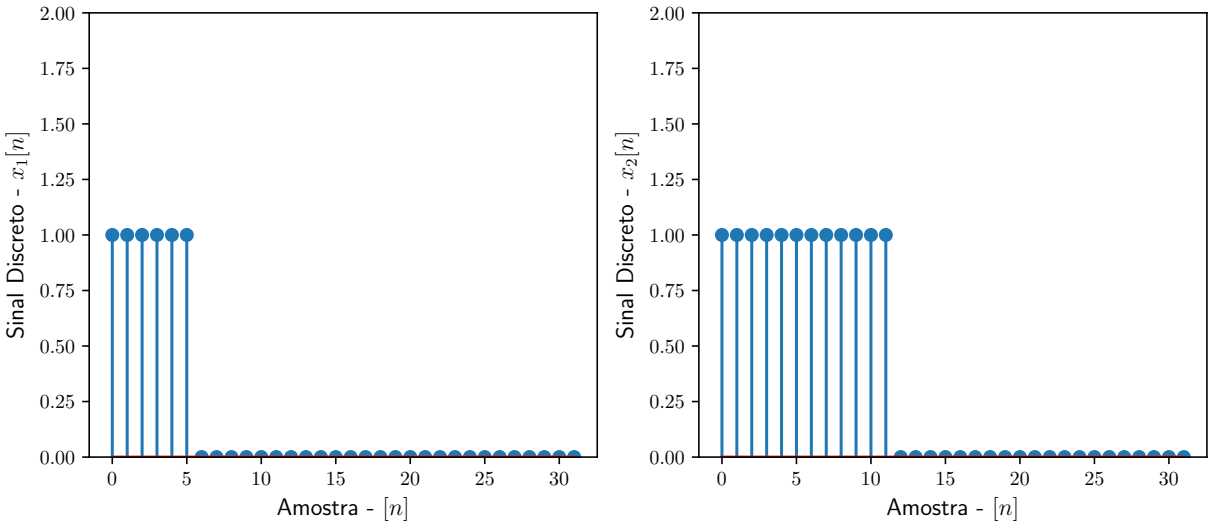


Figure 1.1: Visualização das sequências $x_1[n]$ e $x_2[n]$.

O objetivo do exercício consiste em realizar uma convolução no tempo entre esses dois sinais, ou seja, calcular $x[n] = x_1[n] * x_2[n]$. Uma famosa propriedade da Transformada de Fourier (que também é válida para transformadas discretas) estabelece que a operação de convolução, no domínio do tempo, entre dois sinais é equivalente a realizar à operação de multiplicação entre suas respectivas transformadas, no domínio da frequência. Essa propriedade pode ser denotada matematicamente:

$$x[n] = \sum_{m=0}^{N-1} x_1[n]x_2[n-m] \overset{\mathcal{FD}}{\rightleftharpoons} X_1[k]X_2[k] = X[k]. \tag{1.2}$$

Portanto, é possível realizar o seguinte procedimento a fim de obter o resultado da convolução:

1. Calcular as Transformadas de Fourier Discretas (DFT) para as duas sequências em questão. Denota-se às sequências no domínio da frequência, $X_1[k]$ e $X_2[k]$, a definição da transformada direta:

$$X_1[k] = \sum_{n=0}^{N-1} x_1[n]e^{-j\frac{2\pi k}{N}n}, \quad X_2[k] = \sum_{n=0}^{N-1} x_2[n]e^{-j\frac{2\pi k}{N}n}, \quad (k = 0, 1, \cdots, N-1). \tag{1.3}$$

2. Realizar a operação de multiplicação *ponto-a-ponto* no domínio da frequência, obtendo a sequência resultante

$$X_f[k] = X_1[k]X_2[k], \quad (k = 0, 1, \cdots, N-1). \tag{1.4}$$

3. Calcular a Transformada Inversa de Fourier Discreta (IDFT) para a sequência $X_f[k]$, obtendo a sequência $x_f[n]$ que representa a convolução $x_f[n] = x_1[n] * x_2[n]$, no domínio do tempo:

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{j\frac{2\pi k}{N}n}, \quad (n = 0, 1, \cdots, N-1). \tag{1.5}$$

Teremos, então, o seguinte resultado para cada passo:

Cálculo das Transformadas de Fourier Discretas

Uma vez que o sinal em questão é definido para um número de amostras que é potência de 2, utilizamos o algoritmo de Transformada Rápida de Fourier (FFT) conhecido como *Radix-2 Decimation-in-Time*. Esse algoritmo implementa uma solução de *divisão e conquista* ao dividir o cálculo da transformada da sequência inteira, de tamanho N, em subproblemas de tamanhos $N/2^i$, para $i = 1, 2, ..., log_2(N)$, e recursivamente combinar as suas soluções individuais. Aplicando o algoritmo individualmente para $x_1[n]$ e $x_2[n]$, obtemos os resultados exposto na Figura 1.2.

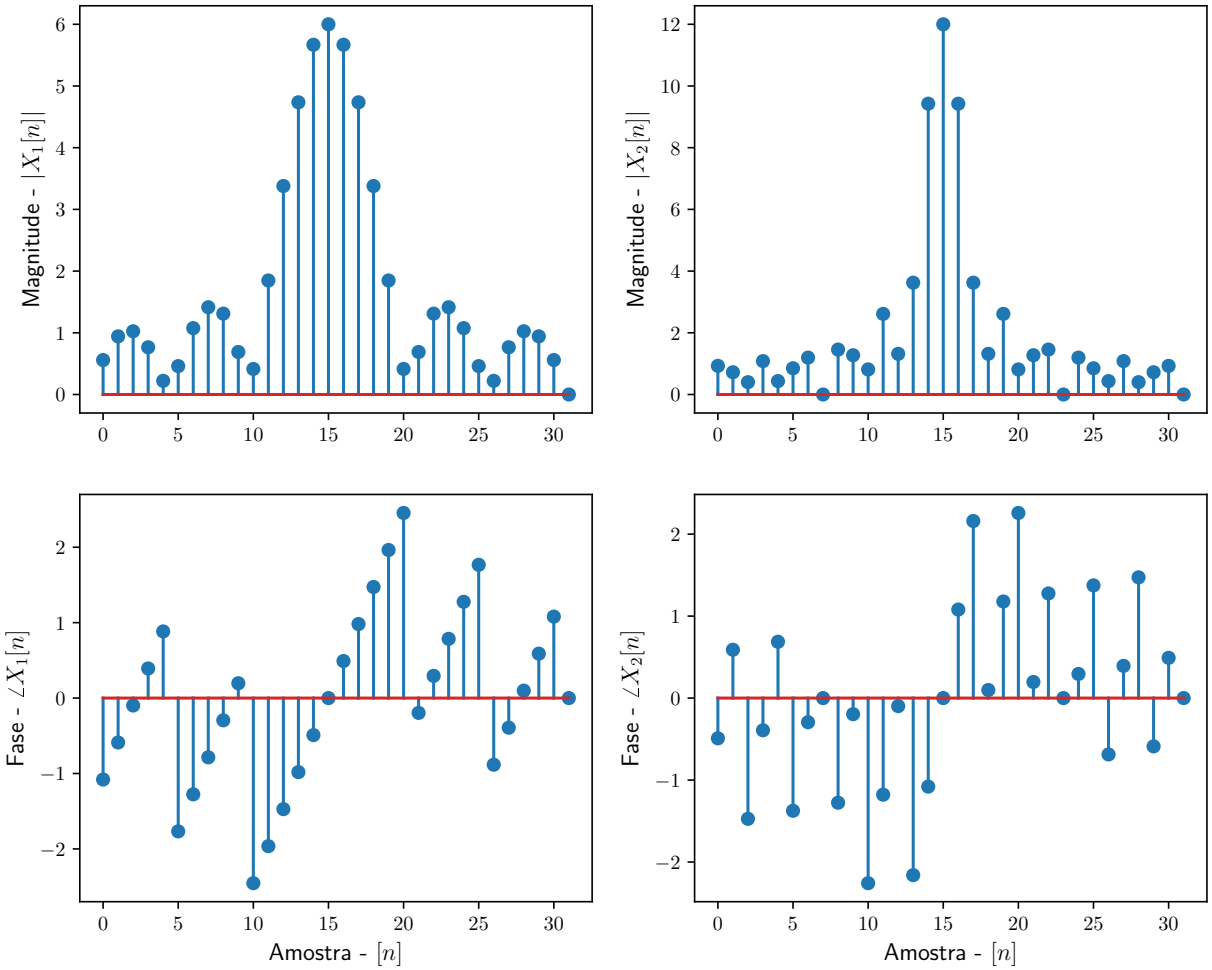


Figure 1.2: Visualização das sequências $X_1[k]$ (esquerda) e $X_2[k]$ (direita).

Nessa visualização, os valores das sequências $X_1[k]$ e $X_2[k]$, que são números complexos, são exibidos através da visualização de magnitude e de fase de cada componente. O formato da magnitude dos sinais exibe a forma da conhecida função `sinc` $= sin(x)/x$, o que é esperado uma vez que os sinais no domínio do tempo consistem em janelas retangulares.

Multiplicação no Domínio da Frequência

Após o cálculo das transformadas individuais, podemos calcular a transformada resultante ao multiplicar cada sequência *ponto-a-ponto*. O procedimento é bastante simples, e o resultado é exibido na Figura 1.3 abaixo.

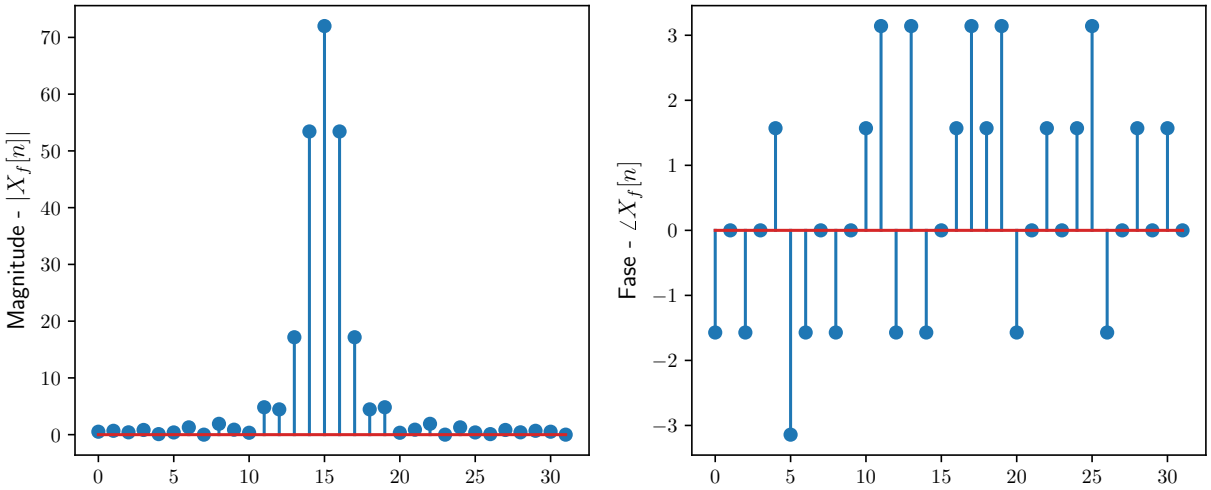


Figure 1.3: Visualização da sequências $X_f[k] = X_1[k]X_2[k]$.

Transformada Inversa de Fourier do Sinal Resultante

Subsequentemente, é possível analisar o resultado da convolução no domínio do tempo ao realizar a IDFT diretamente à sequência $X_f[k]$. O resultado é demonstrado na Figura 1.4 abaixo.

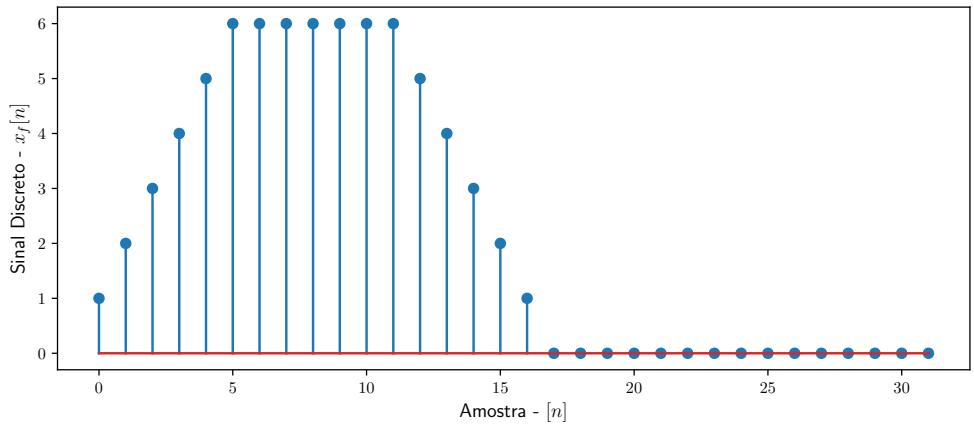


Figure 1.4: Visualização da sequência $x_f[n] = x_1[n] * x_2[n]$.

O sinal resultante exibe alguns resultados interessantes. Primeiramente, nota-se que o sinal resultante da convolução, no domínio do tempo, possui exatamente 16 amostras não-nulas, o mesmo resultado da soma $N_1 + N_2 - 2$, onde N_1 e N_2 são os tamanhos das respectivas sequências originais. Ademais, o sinal resultante possui um formato trapezoidal cujo topo possui $N_1 + 1$ amostras de valor N_1. Um sinal trapezoidal é um resultado característico da convolução entre

dois sinais retangulares, e o mesmo se reduz a um sinal triangular quando os sinais possuem a mesma largura (pois haveria apenas uma superposição completa entre esses sinais).

Finalmente, comparamos o resultado da convolução no tempo através da multiplicação na frequência com o resultado de diretamente computar a convolução discreta no domínio do tempo. As sequências resultantes obtidas são comparadas na Figura 1.5. A fim de enfatizar possíveis diferenças, a visualização é realizada utilizando um gráfico de linhas, apesar de que os dados em questão ainda sejam discretos. Podemos perceber, pela visualização, que os dois métodos obtiveram praticamente os mesmos resultados, dado a perfeita superposição entre as linhas no gráfico. Portanto, demonstramos a equivalência entre a convolução no tempo com a multiplicação na frequência, além de demonstrar a vantagem computacional dessa abordagem, uma vez que multiplicações são menos custosas que convoluções.

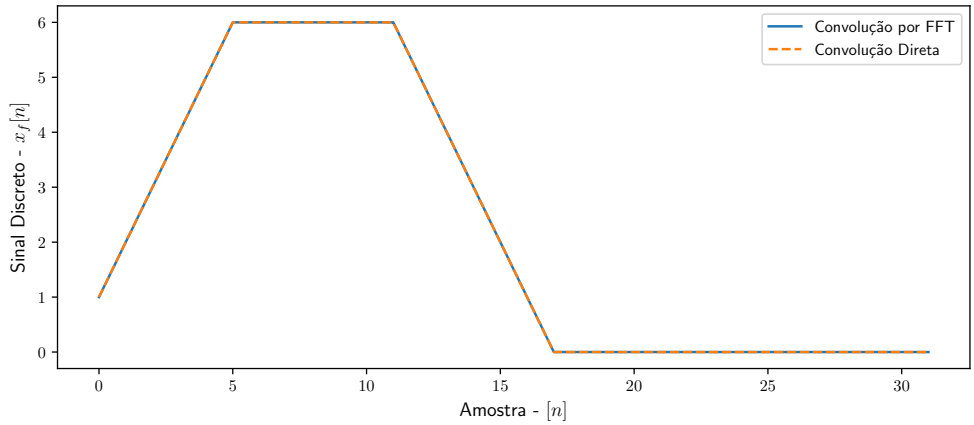


Figure 1.5: Comparação entre o resultado da proposta convolução para os dois métodos citados.

A - Explicação da FFT e Detalhes de Implementação

Nessa secção explicamos a formulação matemática do algoritmo de Transformada Rápida de Fourier e apresentamos uma implementação desse algoritmo utilizando a linguagem de programação Python. Tratamos, aqui, exclusivamente do método *Radix-2 Decimation-In-Time* para implementação da transformada rápida. A implementação apresentada fora utilizada nos exercícios anteriores nesse documento.

Considere a definição da Transformada Discreta de Fourier (DFT), dada como:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad k = 0, 1, \cdots, N-1 \tag{1.6}$$

de onde $W_N = e^{-j2\pi/N}$. Podemos, então, particionar o somatório em dois somatórios independentes, sendo o primeiro uma iteração sobre os elementos de índice ímpar e o segundo uma iteração sobre os elementos de índice par da sequência. Ou seja,

$$\begin{aligned} X[k] &= \sum_{neven} x[n] W_N^{kn} + \sum_{nodd} x[n] W_N^{kn} \\ &= \sum_{m=0}^{(N/2)-1} x[2m] W_N^{k(2m)} + \sum_{m=0}^{(N/2)-1} x[2m+1] W_N^{k(2m+1)} \\ &= \underbrace{\sum_{m=0}^{(N/2)-1} x[2m] W_{N/2}^{km}}_{X_e[k]} + W_N^k \underbrace{\sum_{m=0}^{(N/2)-1} x[2m+1] W_{N/2}^{km}}_{X_o[k]} \end{aligned} \tag{1.7}$$

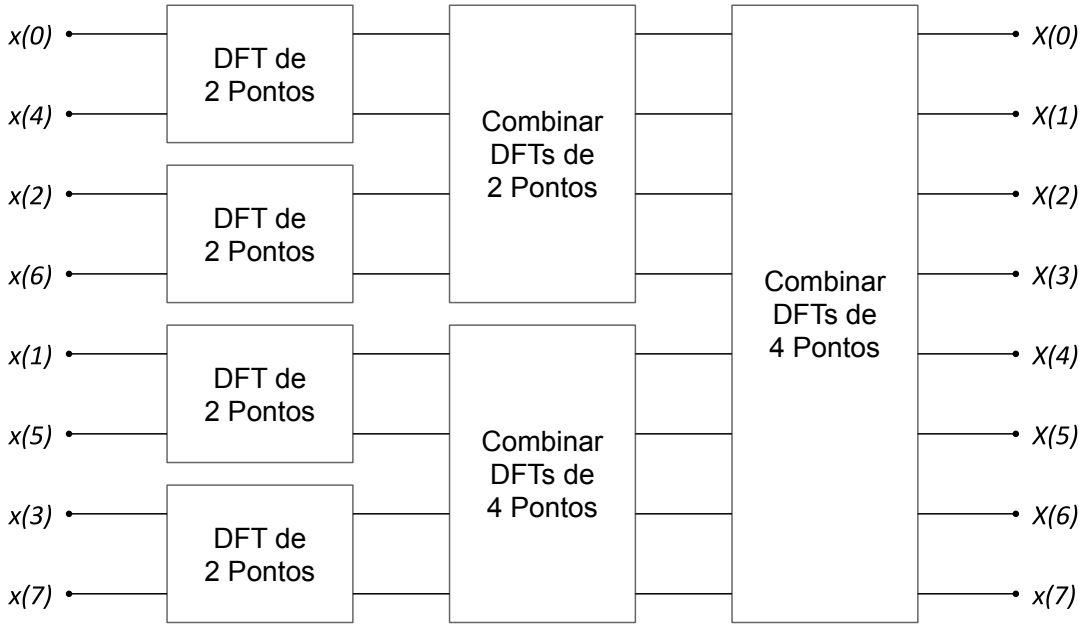
onde no último passo fora utilizado $W_N^{2km} = e^{(-j2\pi/N)2km} = e^{(-j2\pi/(N/2))km} = W_{N/2}^{km}$. Agora, sabemos que $X_e[k]$ e $X_o[k]$ ambos são periódicos com período $(N/2)$, o que implica em $X_e[k + N/2] = X_e[k]$ e $X_o[k + N/2] = X_o[k]$. Ademais, temos a propriedade de que $W_N^{k+N/2} = -W_N^k$. Portanto, a equação acima pode ser representada pelo par de equações:

$$X[k] = X_e[k] + W_N^k X_o[k], \quad k = 0, 1, 2, \cdots, \frac{N}{2} - 1 \tag{1.8}$$

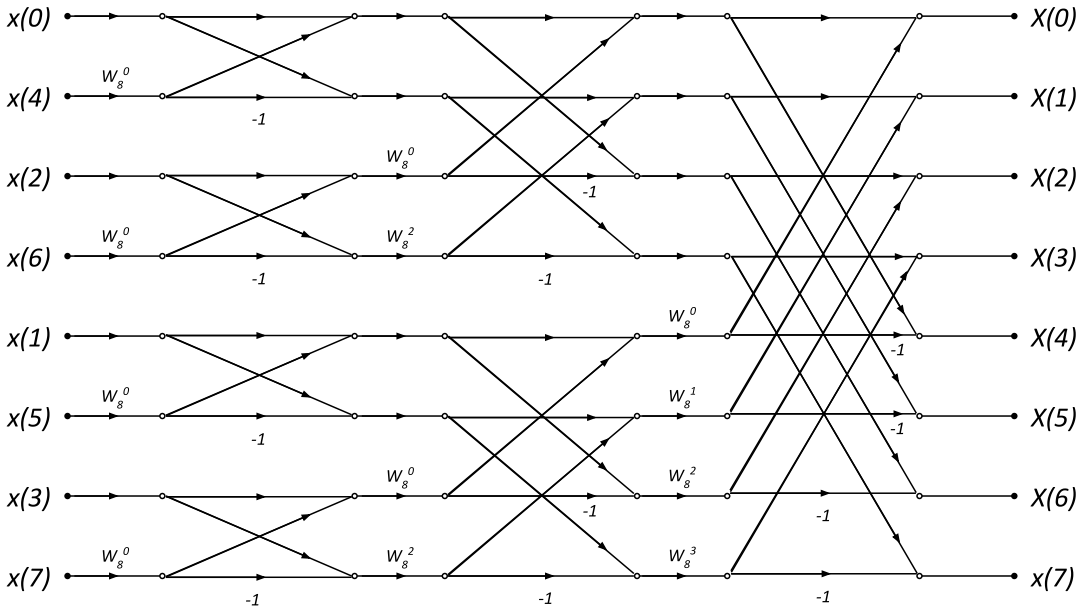
$$X\left[k + \frac{N}{2}\right] = X_e[k] - W_N^k X_o[k], \quad k = 0, 1, 2, \cdots, \frac{N}{2} - 1 \tag{1.9}$$

Portanto, é possível calcular a transformada $X[n]$ utilizando os resultados já computados em $X_e[k]$ e $X_o[k]$. Claramente, também é possível realizar o mesmo particionamento feito em (1.7) diretamente em $X_e[k]$ e $X_o[k]$, o que implica em que o cálculo da transformada final pode ser realizado de maneira recursiva. Essa abordagem, conhecida como *dividir e conquistar*, reduz o número de computações necessárias ao reaproveitar a solução de subproblemas menores.

No caso do algoritmo de FFT em discussão, é possível associar as computações à diagramas de fluxos, como exemplificado na Figura 1.6 para uma sequência de tamanho $N = 8$. Na Figura 1.6a temos a estrutura geral da recursividade na solução da FFT, enquanto que a Figura 1.6b é o diagrama de fluxo que explicita as operações entre os sinais. Nessa estrutura, cada cruzamento entre pares $X_e[k]$ e $X_o[k]$ é denominado de *borboleta*. Por fim, note que o procedimento de recursivamente dividir a sequência acabou causando um "embaralhamento" da ordem dos índices na sequência original, apesar de que as transformadas são calculadas na ordem correta. No caso do algoritmo em questão, no entanto, esse "embaralhamento" pode ser facilmente calculado ao notar-se que ele correponde à inversão dos bits do índice na sua representação binária.



(a)



(b)

Figure 1.6: Esquemático da TRF pelo algoritmo *Radix-2 Decimal-in-Time*.

Complexidade Computacional

Primeiramente, discutimos a complexidade computacional do cálculo da Transformada de Fourier Discreta diretamente pela aplicação da fórmula em (1.6). Note que por essa fórmula temos um total de $N-1$ adições complexas e N multiplicações complexas (uma multiplicação para cada iteração do somatório), sendo essas operações repetidas para cada diferente valor de k. Uma vez que $k = 0, 1, \cdots, N-1$, então teremos um total de N repetições dessas operações, o que resulta em $N^2 - N$ adições complexas e N^2 multiplicações complexas para o cálculo da transformada

discreta. Dessa forma, a complexidade computacional dessa abordagem é dada por

$$\mathcal{O}\left(N^2 - N + N^2\right) = \mathcal{O}\left(N^2\right) \tag{1.10}$$

Agora, consideramos a complexidade do algoritmo de Transformada Rápida pelo método *Radix-2 Decimation-in-Time*. Note que, evidenciado pelo diagrama na Figura 1.6a, esse algoritmo divide o cálculo da transformada num total de $log_2(N)$ estágios, onde N é o número total de pontos. Para cada estágio, pares periódicos de pontos são combinados de acordo com as equações (1.8) e (1.9), o que consiste na operação denominada de *"borboleta"*. Se pré-calcularmos $z_k = W_N^k X_o[k]$, cada *borboleta* consiste em exatamente 1 multiplicação complexa e 2 adições complexas. Uma vez que consistem em cruzamento de pares, cada estágio tem exatamente $N/2$ *borboletas*, o que resulta no total de $N/2$ multiplicações complexas e N adições complexas. Por fim, teremos um total de $(N/2)log_2(N)$ multiplicações complexas e $Nlog_2(N)$ adições complexas, resultando na complexidade computacional de

$$\mathcal{O}\left(\frac{N}{2}log_2(N) + Nlog_2(N)\right) = \mathcal{O}\left(Nlog_2(N)\right) \tag{1.11}$$

Conclui-se que o algoritmo de Transformada Rápida de Fourier apresenta uma melhora significativa em relação ao cálculo direto, uma vez que $\mathcal{O}\left(Nlog_2(N)\right)$ cresce muito mais lentamente que $\mathcal{O}\left(N^2\right)$ para valores muito altos de N.

Implementação

Para este trabalho, o algoritmo de FFT *Radix-2 Decimation-in-Time* fora implementado utilizando a linguagem de programação *Python*. A função de transformação (tanto direta como inversa) é apresentada abaixo:

```
def fft(x_n, inverse=False):
  N = len(x_n)
  W_N = np.exp((-1)**(not inverse) * 2j * np.pi / N)

  x_n = bitrevorder(x_n)

  size = 2
  while size <= N:
    step = size // 2
    for i in range(0, N, size):
      k = 0
      for j in range(i, i + step):
        z_k = (W_N**k) * x_n[j + step]

        x_n[j + step] = x_n[j] - z_k
        x_n[j]        = x_n[j] + z_k

        k += N // size
    size *= 2

  return (1/N)**inverse * x_n
```

Essa implementação é bastante prática. Inicia-se ao realizar o reordenamento dos índices da sequência original, utilizando a função `bitreorder(.)`. Logo após essa operação, inicia-se um *while-loop* para iterar sobre todos os possíveis níveis de recursividade. Dentro desse laço, dois *for-loops* são implementados: o primeiro irá iterar sobre os *"pontos de início"* das *borboletas*,

enquanto o laço seguinte irá iterar sobre todos os pares de borboletas a partir daquele ponto. Para cada iteração dos laços aninhados, a transformada é calculada utilizando exatamente a fórmula exposta em (1.8) e (1.9), sendo $z[k] = W_N^k x[n + S/2]$ uma variável auxiliar.

O cálculo da Transformada Inversa de Fourier segue exatamente o mesmo formato de uma Transformada Direta, porém temos que $W_N = e^{j2\pi/N}$. A implementação de tal método, utilizando a função já apresentada, consistiu apenas em adicionar uma variável booleana (a variável `inverse`) que *"ativa"* os fatores de fase adequado e a normalização do sinal calculado para obter o resultado da transformação inversa. Por fim, enfatiza-se que outras funções auxiliares utilizadas nos exercícios, e que não foram explicitadas nesse relatório, também se encontram implementadas no arquivo de código `FFT.py` que o acompanha.