
TRANSFORMAÇÕES RÁPIDAS, CONVOLUÇÕES E APLICAÇÕES -
ANÁLISE DOS ALGORITMOS E IMPLEMENTAÇÃO.

JOSE LUIZ NETO

TRANSFORMAÇÕES RÁPIDAS, CONVOLUÇÕES E APLICAÇÕES -
ANÁLISE DOS ALGORITMOS E IMPLEMENTAÇÃO.

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO CURSO DE PÓS-GRADUAÇÃO
EM SISTEMAS E COMPUTAÇÃO DA UNIVERSIDADE FEDERAL DA PARAIBA COMO
PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE
MESTRE EM CIÊNCIAS (M.Sc.).

Mário Toyotaro Hattori
- Orientador -

Campina Grande - Paraíba - Brasil

Novembro de 1987



L952t

Luiz Neto, Jose

Transformacoes rapidas, convolucoes e aplicacoes :
analise dos algoritmos e implementacao / Jose Luiz Neto. -
Campina Grande, 1987.
99 p.

Dissertacao (Mestrado em Computacao) - Universidade
Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Processamento Digital de Sinais 2. Transformacoes de
Fourier 3. Dissertacao I. Hattori, Mario Toyotaro, Dr. II.
Universidade Federal da Paraiba - Campina Grande (PB) III.
Titulo

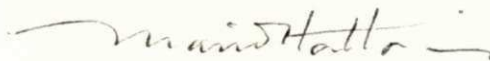
CDU 004.383.3(043)

TRANSFORMAÇÕES RÁPIDAS, CONVOLUÇÕES E APLICAÇÕES -
ANÁLISE DOS ALGORITMOS E IMPLEMENTAÇÃO

JOSÉ LUIZ NETO

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO CURSO DE PÓS-GRADUAÇÃO
EM SISTEMAS E COMPUTAÇÃO DA UNIVERSIDADE FEDERAL DA PARAÍBA COMO
PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE
MESTRE EM CIÊNCIAS (M.Sc.).

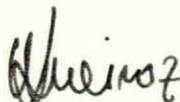
Aprovado por:



MÁRIO TOYOTARO HATTORI - M.Sc.
- Presidente -



JOÃO MARQUES DE CARVALHO - Ph.D.
- Examinador -



BRUNO CORREIA DA N. QUEIROZ - M.Sc.
- Examinador -

A minha esposa Alba, pelo seu
constante incentivo.

Aos Meus filhos Saulo e Simone, como
um incentivo para que se disponham a
grandes êxitos em seus estudos.

AGRADECIMENTOS

Ao Professor Mário Toyotaro Hattori pela sua atenção, estímulo e valiosa orientação.

Ao Professor João Marques de Carvalho pela sua atenção e colaboração.

Ao colega Babatunde Ayodele Oresotu que sempre se dispôs a estudar comigo, durante a integralização dos créditos.

A todos os membros do DSC, a todos os membros do DME e a todos os funcionários do NPD que de alguma forma cooperaram nesta realização.

A minha irmã Francinete e a minha cunhada Albenize pela atenção dispensada aos meus filhos, durante a realização deste trabalho.

SUMARIO

	PAGINA
Lista de Figuras	vii
Lista de Tabelas	ix
RESUMO	x
ABSTRACT	xii

CAPITULO I

INTRODUÇÃO	1
------------------	---

CAPITULO II

ARITMETICA RESIDUAL	5
2.1 - Divisibilidade, Máximo divisor Comum e Algoritmo de Euclides	5
2.2 - Congruências e Resíduos	6

2.3 - Polinômios Ciclotômicos	8
2.4 - A função ϕ de Euler e o Teorema de Euler ...	8
2.5 - O Teorema do Resto Chinês	9
2.5.1 - Aplicações	11
2.6 - Número de Mersenne e Números de Fermat	12
2.7 - Álgebra Polinomial	12
2.7.1 - A convolução	13
2.8 - O Teorema do Resto Chinês	14

CAPITULO III

TRANSFORMAÇÕES DE FOURIER E CONVOLUÇÕES	16
3.1 - Transformações de Fourier	16
3.1.1 - A integral de Fourier	17
3.1.2 - A transformada discreta de Fourier	19
3.1.3 - As transformadas continua e discreta de Fourier	23
3.2 - Convoluções	23
3.2.1 - Definição e aplicações	24
3.2.2 - Impulso unitário	24
3.2.3 - A convolução integral	25
3.2.4 - A convolução periódica discreta	26
3.2.5 - A convolução linear e a convolução circular	27
3.2.6 - A convolução continua e a convolução discreta	29

CAPITULO V

A CONVOLUÇÃO RAPIDA	61
5.1 - Computação da Convolução através do Teorema do Resto Chinês	61
5.2 - Computação da Convolução através de Algoritmos FFT	66
5.2.1 - Sequências reais quaisquer	66
5.2.2 - Sequências reais pares	67
5.3 - Computação da Convolução através de Transformadas de Mersenne	68
5.3.1 - Computação da transformada de Mersenne \bar{X}_k de x_m	70
5.3.2 - Computação da transformada de Mersenne \bar{H}_k de h_n	72
5.3.3 - Computação da transformada inversa de Mersenne do produto $\bar{X}_k \bar{H}_k$	72
5.4 - Diferença entre a Convolução Linear e a Convolução Circular	73
5.4.1 - Computação da convolução linear y_r de x_m com h_n	74
5.4.2 - Computação da convolução circular y_r de x_m com h_n	75
5.5 - Produto de Polinômios	75

CAPITULO VI

IMPLEMENTAÇÕES	76
6.1 - O Algoritmo FFT de Cooley-Tukey	77
6.1.1 - Obtenção dos elementos de $X(n)$ - Equação (4.4)	78
6.1.2 - Reordenamento dos elementos da DFT $X(n)$	80
6.1.3 - Obtenção dos elementos da DFT inversa, quando for o caso	80
6.1.4 - Função IBR	80
6.2 - O Algoritmo de Bruun	81
6.2.1 - Obtenção do vetor de raízes (VR) ...	81
6.2.2 - Obtenção da matriz de apontadores (para VR)	83
6.2.3 - Caminha, empilha, obtém valores finais e desempilha	84
6.2.3.1 - Caminha	84
6.2.3.2 - Empilha	87
6.2.3.3 - Obtém valores finais	87
6.2.3.4 - Desempilha	87
6.2.4 - Rotina rápida de divisão	87
6.2.5 - Obtenção da DFT \bar{X}_k de x_m	90

CAPITULO VII

RESULTADOS NUMERICOS, CONCLUSOES E SUGESTOES	93
7.1 - Resultados	93
7.2 - Conclusões	96
7.3 - Sugestões	97
REFERENCIAS BIBLIOGRAFICAS	98

LISTA DE FIGURAS

	PAGINA
4.1 - Comparação do número de multiplicações necessárias pelo cálculo direto e o algoritmo de Cooley-Tukey (o FFT).....	37
4.2 - Representação gráfica, $N = 4$, para o algoritmo de Cooley-Tukey.....	40
4.3 - Computação da DFT de 8 pontos pelo algoritmo de Bruun.....	51
4.4 - Simetria das funções trigonométricas seno e cosseno para $N = 16$	55
4.5 - Computação da DFT de 16 pontos pelo algoritmo de Bruun	56
5.1 - Computação da convolução circular de comprimento N através do Teorema do Resto Chinês. N Primo ...	64

5.2 - Computação da convolução real em 3 passos de FFT.	67
5.3 - Computação da convolução circular de comprimento p módulo (2^D-1) por transformadas de Mersenne ...	69
5.4 - Ilustração da computação da convolução linear ...	74
5.5 - Ilustração da computação da convolução circular .	75
6.1 - Situação do vetor de raízes (VR), para $N = 16$ - Algoritmo de Bruun	82
6.2 - Situação inicial da matriz de apontadores IAP, para $N = 32$ - Algoritmo de Bruun	83
6.3 - Diagrama do algoritmo de Bruun para $N = 8$	86
6.4 - Obtenção dos índices dos elementos de uma DFT de N pontos a partir dos índices dos elementos de uma DFT de $N/2$ pontos - Algoritmo de Bruun	92

LISTA DE TABELAS

	PAGINA
4.1 - Número de operações reais não triviais para os algoritmos FFTs Base-2 e Base-4	44
4.2 - Número de multiplicações reais não triviais, para computar DFTs complexos pelo método Rader-Brenner	46
7.1 - Tempo de execução, em segundos, dos algoritmos de Bruun e Cooley-Tukey (média de 4 execuções para cada N)	94
7.2 - Resultados da regressão linear do tempo em função de $N \log_2 N$	95

RESUMO

As transformações de Fourier são usadas com frequência em Análise Numérica e no processamento digital de sinais é uma ferramenta indispensável de análise e síntese de dispositivos como filtros digitais. A avaliação de transformadas discretas de Fourier usando diretamente a sua definição é um processo computacionalmente dispendioso; felizmente existem algoritmos que permitem tornar o processo mais eficiente, conhecidos como algoritmos de transformação rápida de Fourier, aos quais nos referimos pelas suas iniciais em inglês FFT, de Fast Fourier Transform.

Motivados pelos fatos acima e também pela utilização de FFT na computação eficiente da convolução, neste trabalho

- a) fizemos um estudo dos fundamentos matemáticos das transformações rápidas;
- b) analisamos as diversas formulações de algoritmos de transformação rápida, em particular, FFT, transformada de Mersenne e o teorema do resto chinês;

- c) estudamos e realizamos a implementação de alguns algoritmos de transformação rápida para avaliar as diferenças entre os algoritmos definidos matematicamente e as suas implementações como um software robusto, eficiente e portátil; e
- d) estudamos e realizamos a implementação de alguns algoritmos de computação rápida de convoluções.

ABSTRACT

The Fourier transforms are often used in Numerical Analysis and in Digital Signal Processing they are fundamental tools for analysing and synthesizing devices as digital filters. Evaluation of discrete Fourier transforms using its definition directly is a computationally expensive process; fortunately, there exist algorithms that allow the process to become more efficient, known as fast Fourier transforms, to which we refer as FFT.

Motivated by the above facts and also by the use of FFT in efficiently computing convolutions, in this work:

- a) we have studied the mathematical foundations of fast transforms;
- b) we have analyzed different formulations of fast transform algorithms, in particular, FFT, Mersenne transform and the Chinese Remainder Theorem;

- c) we have studied and implemented some fast transform algorithms in order to evaluate the differences between algorithms defined mathematically and their implementations as robust, efficient, and portable software; and
- d) we have studied and implemented some fast convolution algorithms.

CAPITULO I

INTRODUÇÃO

Qualquer solução computacional dada a um problema matemático deve levar em consideração dois fatores importantes: o tempo de computação e o espaço de memória necessário. Por isso às vezes, resolver um simples problema de matemática pode significar resolver um complexo problema de computação. A razão disso é que nem sempre a linguagem de programação escolhida facilita a implementação desejada. Por exemplo, a linguagem de programação FORTRAN não possui recursividade, muito embora frequentemente precisemos dela, como veremos na implementação do algoritmo de Bruun (cap. 6). A necessidade de algoritmos rápidos pode ser justificada observando-se que se computássemos uma Transformada Discreta de Fourier (DFT), \bar{X}_k de N termos, diretamente pela definição (cap. 3), o tempo de computação seria proporcional a N^2

enquanto que os algoritmos de transformação rápida (FFT - Fast Fourier Transform), computam a mesma DFT \bar{X}_k num tempo proporcional a $N \log_2 N$, que significa uma redução drástica do tempo quando comparamos N^2 com $N \log_2 N$. Para $N = 1024$ essa redução é quase de 200 para 1. Em 1965, Cooley & Tukey [4] publicaram um algoritmo FFT para computação da transformada discreta de Fourier, aplicável quando N é um número composto, isto é, N é o produto de dois ou mais inteiros.

A DFT constitui uma ferramenta matemática poderosa para resolver diversos problemas da física que envolvem relações entre o domínio do tempo e o domínio da frequência. Como na representação de um sinal discreto. Além disso, uma das principais aplicações da DFT é a computação de convoluções discretas e essas convoluções por sua vez, constituem uma classe de ferramentas muito utilizada por engenheiros de comunicações, por duas razões [3]:

I) a convolução é um modelo dos processos físicos que se tornam ativos em um sistema linear.

II) a convolução auxilia no entendimento das relações entre o domínio do tempo e o domínio da frequência.

O principal objetivo deste trabalho é a computação rápida e eficiente da transformada de Fourier e convoluções, haja vista a grande importância de suas aplicações práticas. Por isso, selecionamos e implementamos alguns algoritmos para computação da

DFT e convoluções com algumas restrições. Alguns requisitos foram obedecidos a fim de que tal implementação viesse a constituir-se em um software científico transportável. Em virtude dos vários assuntos envolvidos e para uma melhor compreensão do trabalho, adotamos o seguinte desenvolvimento:

- 1 - Uma revisão da aritmética modular (aritmética residual) onde abordamos o teorema do resto chinês e suas aplicações práticas, bem como os números de Mersenne e os números de Fermat.
- 2 - Definimos as transformadas contínua e discreta de Fourier onde procuramos fazer um paralelo entre ambas. Definimos também as convoluções contínua (integral) e discreta. Mostramos a diferença entre a convolução linear e a convolução circular.
- 3 - Fizemos um estudo de alguns algoritmos de transformações rápidas [2] e [13], detalhando completamente o algoritmo FFT de Cooley-Tukey e o algoritmo de Bruun. Relacionamos alguns algoritmos de transformações multidimensionais.
- 4 - Fizemos um estudo detalhado de três algoritmos que computam convoluções rápidas:

- I) computação de convolução através do teorema do resto chinês,

II) computação da convolução através de transformadas de Mersenne e

III) computação da convolução através de algoritmos Fast Fourier Transform - FFT.

5 - Apesar dos programas de computador, implementados em linguagem FORTRAN, não constarem aqui, damos uma visão geral do procedimento computacional desses algoritmos (*).

6 - Por fim, apresentamos resultados numéricos, as conclusões e sugestões.

(*) - os programas a que nós nos referimos, encontram-se a disposição dos interessados, na biblioteca de subrotinas FORTRAN no NPD - UFPb - CAMPUS II, que tem como responsável o Prof. Mário Toyotaro Hattori.

CAPITULO II

ARITMETICA RESIDUAL

2.1) - Divisibilidade, máximo divisor comum e algoritmo de Euclides

Sejam a e b dois inteiros, com b positivo. A divisão de a por b é definida por $a = bq + r$, $0 \leq r < b$, onde q é chamado o quociente e r é chamado o resto. Quando $r = 0$, b e q são fatores ou divisores de a , e neste caso, dizemos que b divide a e denotamos por b/a . Quando a não possui outros divisores além de ± 1 e $\pm a$, dizemos que a é primo, caso contrário dizemos que a é composto.

Quando a é composto, podemos sempre fatorar a em um produto de potências de números primos $p_i^{c_i}$, onde c_i é um inteiro positivo, com

$$a = \prod_i p_i^{c_i}$$

O teorema fundamental da aritmética, garante que esta fatoração é única [13].

O maior inteiro positivo d que divide dois inteiros a e b é chamado o máximo divisor comum entre a e b e é denotado por

$$d = (a,b)$$

Quando $d = (a,b) = 1$, dizemos que a e b são mutuamente primos ou primos entre si.

De acordo com o algoritmo de Euclides [12], dados a, b inteiros não nulos, existe um único par (q,r) de números inteiros tal que:

$$a = bq + r \quad \text{com} \quad 0 \leq |r| < |b| .$$

2.2) - Congruências e resíduos

Seja E um conjunto não vazio e seja R uma relação de equivalência sobre E [12]. Para todo elemento a de E o conjunto

$$\bar{a} = \{x \in E / x \equiv a \pmod{R}\}$$

é denominado classe de equivalência módulo R determinada pelo

elemento a e este elemento, por sua vez, é chamado representante da classe de equivalência a . Assim, para todo número inteiro a o conjunto

$$\bar{a} = \{r \in \mathbb{Z} / r \equiv a \pmod{m}\}$$

onde \mathbb{Z} é o conjunto dos números inteiros [11], é a classe de equivalência determinada por a segundo a relação de congruência módulo m ; diremos neste caso, que a é a classe de equivalência módulo m determinada pelo inteiro a , ou, que a é a classe de restos módulo m determinada pelo inteiro a [12].

Dois inteiros a e b pertencentes a mesma classe de equivalência são ditos congruentes módulo m e tal equivalência é denotada por

$$a \equiv b \pmod{m} .$$

Assim, dois inteiros a e b são congruentes módulo m se $m \mid (a - b)$ [10].

Para nós, a relação $r \equiv a \pmod{b}$ significará que r é o resto da divisão de a por b [13].

2.3) - Polinômios ciclotônicos

Definição 1: Uma raiz da unidade no conjunto dos números complexos é um número w tal que $w^n = 1$ para algum inteiro n .

Proposição 1: $X^n - 1$ tem n zeros distintos em \mathbb{C} , o conjunto dos números complexos. Os zeros de $X^n - 1$ são as n -ésimas raízes da unidade [5].

Definição 2: Um polinômio cujos zeros são precisamente as raízes n -ésimas da unidade é chamado o n -ésimo polinômio ciclotômico.

2.4) - A função ϕ de Euler e o teorema de Euler

Definição 3: a função ϕ de Euler $\phi(m)$ [9] é definida como segue:

$$\phi: \mathbb{N} \rightarrow \mathbb{N}$$

$$m \mapsto \phi(m) = s,$$

onde \mathbb{N} é o conjunto dos números naturais e s é o número de inteiros positivos menores ou iguais a m e relativamente primos com m .

Ex.: $\phi(10) = 4$ pois, os inteiros menores ou iguais a 10 são 1,2,3,4,5,6,7,8,9 e 10. E os menores ou iguais a 10 e relativamente primos com 10 são 1,3,7 e 9. Logo $\phi(10) = 4$.

O teorema de Euler [6]:

Sejam a e m números inteiros e positivos. Se $(a,m) = 1$, então

$$a^{\phi(m)} \equiv 1 \pmod{m} .$$

2.5) O teorema do resto chinês

Sejam m_i e k inteiros positivos maiores que 1 com $(m_i, m_j) = 1$ para $i \neq j$. O conjunto de congruências lineares $x \equiv r_i \pmod{m_i}$ ($i = 1, \dots, k$) tem uma única solução módulo M com

$$M = \prod_{i=1}^k m_i$$

onde, k é o número de congruências lineares.

Prova: a prova do teorema é estabelecida usando-se as relações

$$x \equiv \sum_{i=1}^k (M/m_i) r_i T_i \pmod{M} \tag{2.1}$$

$$(M/m_i) T_i \equiv 1 \pmod{m_i} . \tag{2.2}$$

A equação (2.2) define k congruências lineares. Como os m_i são relativamente primos, $(m_i, (M/m_i)) = 1$ e cada uma dessas congruências tem uma única solução T_i que pode ser computada pelo algoritmo de Euclides ou pelo teorema de Euler. Vamos agora reduzir x em (2.1) módulo m_u exceto para M/m_u , todas as expressões M/m_i contêm m_u como um fator e portanto, são iguais a zero módulo m_u . Conseqüentemente, (2.1) reduz-se a

$$x \equiv (M/m_u) r_u T_u \pmod{m_u} \quad (2.3)$$

e, já que (2.2) implica que $(M/m_u) T_u \equiv 1 \pmod{m_u}$ (2.3) torna-se

$x \equiv r \pmod{m_u}$, para todo u . Portanto, a equação (2.1) é a solução do teorema.

Exemplo 2.1

Como ilustração do teorema, vamos encontrar a solução do conjunto de congruências lineares

$$x \equiv 2 \pmod{3}$$

$$x \equiv 1 \pmod{4}$$

$$x \equiv 3 \pmod{5}$$

Solução: temos que $r_1 = 2$, $r_2 = 1$, $r_3 = 3$, $m_1 = 3$, $m_2 = 4$, $m_3 = 5$,
 $M = 3.4.5 = 60$, $M/m_1 = 20$, $M/m_2 = 15$ e $M/m_3 = 12$. As
congruências $1 \equiv 20T_1 \pmod{3}$, $1 \equiv 15T_2 \pmod{4}$ e $1 \equiv 12T_3 \pmod{5}$
são resolvidas respectivamente por $T_1 = 2$, $T_2 = 3$
e $T_3 = 3$. Consequentemente,

$$x \equiv (20.2.2 + 15.1.3 + 12.3.3) \pmod{60}$$

o que implica $x = 53$.

2.5.1) - Aplicações

O teorema do resto chinês também é frequentemente usado para
transformar uma sequência de dados x_m de M pontos unidimensionais
para um conjunto de dados k -dimensional (também com M pontos).
Isto é dado observando-se que se n é definido módulo M , com $n =$
 $0, \dots, M-1$, podemos redefinir n pelo teorema do resto chinês como

$$n \equiv \sum_{i=1}^k (M/m_i) n_i T_i \pmod{M} \quad (2.4)$$

onde n_i e i assumem os valores $0, \dots, m_i-1$. Este mapeamento só é
possível quando M for um produto de fatores m_i relativamente
primos ($M = \prod_{i=1}^k m_i$). Este mapeamento é muito importante para a
computação da transformada discreta de Fourier e convoluções
[13].

Exemplo 2.2

Supondo que $M = 6$, vamos fazer uma aplicação da equação (2.4). Neste caso, $m_1 = 2$ e $m_2 = 3$. Logo $n_1 = 0,1$ e $n_2 = 0,1,2$. A sequência n é dada por $\{0,1,2,3,4,5\}$. Por (2.2), concluímos que $T_1 = 1$ e $T_2 = 2$. Por (2.4), temos que

$$n \equiv (3n_1 + 4n_2) \pmod{6},$$

de modo que, quando o par n_1, n_2 assume sucessivamente os valores $\{(0,0), (1,0), (0,1), (1,1), (0,2), (1,2)\}$, a sequência n torna-se $\{0,3,4,1,2,5\}$.

2.6) - Números de Mersenne e números de Fermat

Os números de Mersenne são definidos por $M = 2^p - 1$, com $p > 2$ primo. Enquanto que os números de Fermat são definidos por $F_t = 2^{2^t} + 1$, onde t é qualquer inteiro positivo [13].

2.7) - Álgebra polinomial

A álgebra polinomial fornece uma importante regra no processamento digital de sinais, pois a convolução e outras extensões e transformadas discretas de Fourier, podem ser expressas em termos de operações sobre os polinômios [13].

2.7.1) - A convolução

Consideremos a convolução y_r de duas seqüências x_m e h_n de N termos

$$y_r = \sum_{n=0}^{N-1} h_n x_{r-n}, \quad r = 0, \dots, 2N-2 \quad (2.5)$$

Suponhamos que os N elementos de x_m e h_n representem os coeficientes dos polinômios $X(z)$ e $H(z)$ de grau $N-1$ na variável z . Temos

$$X(z) = \sum_{m=0}^{N-1} x_m z^m \quad \text{e} \quad H(z) = \sum_{n=0}^{N-1} h_n z^n .$$

Se nós multiplicarmos $H(z)$ por $X(z)$, o resultado será um polinômio $Y(z)$ de grau $2N-2$, já que $H(z)$ e $X(z)$ têm grau $N-1$. Assim,

$$Y(z) = H(z)X(z) = \sum_{r=0}^{2N-2} a_r z^r .$$

Na multiplicação polinomial cada coeficiente a_r de z^r é obtido somando-se todos os produtos $h_n x_m$ tais que $n + m = r$. Consequentemente, $m = r-n$ e

$$a_r = y_r = \sum_{n=0}^{N-1} h_n x_{r-n} \quad \text{e, portanto,}$$

$$Y(z) = \sum_{r=0}^{2N-2} y_r z^r . \quad (2.6)$$

A equação (2.6) mostra que a convolução de duas sequências pode ser tratada como a multiplicação de dois polinômios. Além disso, se a convolução definida por (2.5) for circular, os índices r , m e n são definidos módulo N . Assim, em uma convolução circular nós temos $N \equiv 0$ (o resto da divisão de N por N é zero). Isto implica que $z^N = 1$ e portanto, a convolução circular pode ser vista como o produto de dois polinômios módulo o polinômio $z^N - 1$ [13], isto é:

$$Y(z) \equiv H(z)X(z) \pmod{(z^N - 1)}.$$

2.8) O Teorema do Resto Chinês

Seja $H(z) = a_0 + a_1 z + a_2 z^2 + \dots + a_{N-1} z^{N-1}$ e suponhamos que $P(z)$ é o produto de d polinômios $P_i(z)$ não tendo fatores comuns, isto é:

$$P(z) = \prod_{i=1}^d P_i(z) \text{ com } P_i \text{ e } P_u, \quad i \neq u,$$

relativamente primos, então [13]

$$H(z) \equiv \sum_{i=1}^d S_i(z) H_i(z) \pmod{P(z)}$$

onde $H_i(z) \equiv H(z) \pmod{P_i(z)}$ e, para cada valor u de i tem-se:

$$S_u(z) \equiv 0 \pmod{P_i(z)}, \quad i \neq u,$$

$$S_u(z) \equiv 1 \pmod{P_u(z)}$$

e

$$S_u(z) \equiv T(z) \prod_{\substack{i=1 \\ i \neq u}}^d P_i(z) \pmod{P(z)}$$

Com $T_u(z)$ definido por

$$T_u(z) \prod_{\substack{i=1 \\ i \neq u}}^d P_i(z) \equiv 1 \pmod{P_u(z)} .$$

CAPITULO III

TRANSFORMAÇÕES DE FOURIER E CONVOLUÇÕES

3.1) - Transformações de Fourier

As transformações de Fourier proporcionam uma representação alternativa, no domínio da frequência, para sinais e sistemas normalmente descritos no domínio do tempo, facilitando assim a análise de vários fenômenos físicos (ex. tratamento e transmissão de sinais elétricos) estudados em engenharia. Entram na análise de estabilidade de muitos métodos numéricos, especialmente no método das diferenças finitas, aplicado aos problemas de valor inicial em equações diferenciais parciais [15], e também usado na solução de problemas de valores inicial e de fronteira (PVIF) em equações diferenciais parciais [7].

3.1.1) - A integral de Fourier

Definição: a integral de Fourier é definida pela expressão

$$H(f) = \int_{-\infty}^{\infty} h(t) e^{-j2\pi ft} dt, \quad j = \sqrt{-1}. \quad (3.1)$$

Se a integral em (3.1) existir para cada valor do parâmetro f então a equação (3.1) define $H(f)$, a transformada de Fourier de $h(t)$ [2]. Nas aplicações práticas, $h(t)$ é considerada uma função da variável tempo (t) e, $H(f)$ é considerada uma função da variável frequência (f).

Em geral, a transformada de Fourier é um número complexo da forma

$$H(f) = R(f) + jI(f)$$

onde, $R(f)$ é a parte real da transformada de Fourier e $I(f)$ é a parte imaginária da transformada de Fourier.

Definição: A transformada inversa de Fourier [2] é definida como

$$h(t) = \int_{-\infty}^{\infty} H(f) e^{j2\pi ft} df, \quad j = \sqrt{-1}. \quad (3.2)$$

Se as funções $h(t)$ e $H(f)$ são relacionadas pelas equações (3.1) e (3.2), as duas funções são consideradas um par de transformadas de Fourier e, nós indicamos esta relação com a notação

$$h(t) \iff H(f).$$

Algumas propriedades da transformada de Fourier

i) Linearidade

Se $x(t)$ e $h(t)$ tem transformadas de Fourier $X(f)$ e $H(f)$ respectivamente, então a soma $x(t) + h(t)$ tem transformada de Fourier $X(f) + H(f)$. Consequência imediata da definição.

$$\text{Notação: } x(t) + h(t) \iff X(f) + H(f).$$

ii) Simetria

Se $h(t)$ e $H(f)$ são um par de transformadas então $H(t) \iff h(-f)$. Consequência imediata da definição.

iii) Escalonamento do tempo

Suponhamos $k \in \mathbb{R}$, o conjunto dos números reais, $k > 0$. Se a transformada de Fourier de $h(t)$ é $H(f)$ então

$$h(kt) \Leftrightarrow \frac{1}{k} H(f/k)$$

Prova: Fazendo a substituição $T = kt$, tem-se

$$\frac{dT}{dt} = k \text{ e portanto } \frac{dT}{k} = dt .$$

$$\int_{-\infty}^{\infty} h(kt) e^{-j2\pi ft} dt = \int_{-\infty}^{\infty} h(T) e^{-j2\pi f \frac{T}{k}} \frac{dT}{k} = \frac{1}{k} H(f/k) .$$

obs.: se $k < 0$, temos

$$h(kt) \Leftrightarrow \frac{1}{|k|} H(f/k)$$

3.1.2) - A transformada discreta de Fourier

Definição: Dada a sequência x_m de N termos de números reais ou complexos, a transformada discreta de Fourier (DFT) \bar{X}_k da sequência x_m é definida por

$$X_k = \sum_{m=0}^{N-1} x_m W^{mk} ; k = 0, \dots, N-1;$$

$$W = e^{-j2\pi/N} \text{ e } j = \sqrt{-1} . \quad (3.3)$$

Uma das propriedades mais importantes da DFT é que x_m e \bar{X}_k estão inequivocamente relacionadas, constituindo o que se chama um par de transformadas discretas de Fourier (ou par discreto de Fourier), com a transformada direta definida por (3.3) e a transformada inversa definida por

$$x_m = \frac{1}{N} \sum_{k=0}^{N-1} X_k W^{-mk}, \quad m = 0, \dots, N-1.$$

A relação entre a sequência x_m e sua transformada \bar{X}_k é denotada por

$$\{x_m\} \Leftrightarrow \{\bar{X}_k\}.$$

Propriedades

Em virtude do grande número de propriedades, enunciaremos aqui apenas as principais. As demonstrações das propriedades aqui relacionadas e as demais propriedades e suas respectivas demonstrações podem ser encontradas em [13].

i) A DFT de uma sequência permutada

$$\{x_{pm}\} \Leftrightarrow \{\bar{X}_{qk}\}$$

Nós assumimos que a sequência x_m é permutada, com m substituído por pm módulo N , onde p é um inteiro relativamente primo com N [13].

ii) A correlação de sequências reais

$$\left\{ \sum_{n=0}^{N-1} h_n x_{l+n} \right\} \Leftrightarrow \{ \bar{H}_k^* \bar{X}_k \}, \text{ onde } H_k^* \text{ é o complexo conjugado de } \bar{H}_k .$$

iii) - O teorema de Parseval

$$\left\{ \sum_{m=0}^{N-1} x_m^2 \right\} \Leftrightarrow \left\{ \frac{1}{N} \sum_{k=0}^{N-1} |\bar{X}_k|^2 \right\}$$

DFTs de Sequências Reais

Na maioria das aplicações práticas, a sequência de entrada x_m é uma sequência real, isto é, os elementos de x_m são números reais. Como $w^{mk} = \cos \left(\frac{2\pi mk}{N} \right) - j \sin \left(\frac{2\pi mk}{N} \right)$, a DFT \bar{X}_k de x_m torna-se

$$\bar{X}_k = \sum_{m=0}^{N-1} x_m w^{mk} = \sum_{m=0}^{N-1} x_m \left(\cos \left(\frac{2\pi km}{N} \right) - j \sin \left(\frac{2\pi km}{N} \right) \right) =$$

$$= \sum_{m=0}^{N-1} x_m \cos \left(\frac{2\pi km}{N} \right) - j \sum_{m=0}^{N-1} x_m \sin \left(\frac{2\pi km}{N} \right) . \quad (3.4)$$

Como x_m é real, (3.4) implica que a parte real $\text{Re}\{\bar{X}_k\}$ de \bar{X}_k é par e a parte imaginária $\text{Im}\{\bar{X}_k\}$ de \bar{X}_k é ímpar e portanto, temos

- i) $\text{Re}\{\bar{X}_k\} = \text{Re}\{\bar{X}_{-k}\}$
- ii) $\text{Im}\{\bar{X}_k\} = -\text{Im}\{\bar{X}_{-k}\}$
- iii) $|\bar{X}_k| = |\bar{X}_{-k}|$

De modo análogo, se x_m for uma sequência de números imaginários puros, então

$$\text{iv) } \operatorname{Re}\{\bar{X}_k\} = -\operatorname{Re}\{\bar{X}_{-k}\}$$

$$\text{v) } \operatorname{Im}\{\bar{X}_k\} = \operatorname{Im}\{\bar{X}_{-k}\}$$

$$\text{vi) } |\bar{X}_k| = |\bar{X}_{-k}|.$$

As propriedades i) a vi) podem ser usadas para computar simultaneamente as transformadas \bar{X}_k e \bar{H}_k de duas sequências reais de N pontos x_m e h_n , com um simples DFT complexo [13], isto é, seja $y_m = x_m + jh_n$ e conseqüentemente $\bar{Y}_k = \bar{X}_k + j\bar{H}_k$.

DFTs de sequências pares e sequências ímpares

Se x_m for uma sequência real e par, então $x_m = x_{-m}$. Conseqüentemente,

$$\sum_{m=0}^{N-1} x_m \operatorname{sen}\left(\frac{2\pi mk}{N}\right) = \sum_{m=0}^{N/2-1} (x_m - x_{-m}) \operatorname{sen}\left(\frac{2\pi mk}{N}\right) = 0.$$

Logo, a DFT \bar{X}_k de x_m é uma sequência real e também par, isto é,

$$X_k = \sum_{m=0}^{N-1} x_m \cos\left(\frac{2\pi mk}{N}\right).$$

Se x_m for uma sequência real ímpar, então $x_m = -x_{-m}$. É fácil ver que a DFT \bar{X}_k de x_m é uma sequência de números imaginários

puros e impar, isto é,

$$\bar{X}_k = -j \sum_{m=0}^{N-1} x_m \operatorname{sen} \left(\frac{2\pi mk}{N} \right) .$$

3.1.3) - As transformadas continua e discreta de Fourier

Na definição (3.3), a sequência x_m representa, por exemplo, N amostras consecutivas $x(mT)$ de um certo sinal contínuo $x(t)$ no domínio do tempo, enquanto que a sequência \bar{X}_k representa N amostras consecutivas $\bar{X}(kf)$ no domínio da frequência, onde T é o espaçamento entre as amostras $x(mT)$ e $\bar{X}(f)$ é a transformada de Fourier de x_m . Assim, sob certas condições, a DFT fornece uma aproximação da transformada contínua de Fourier de uma função [13].

3.2) - Convoluções

A convolução de duas funções é um conceito físico significativo em diversos campos científicos. Em particular, o teorema da convolução proporciona um tipo de relação de transformada de Fourier de grande importância na análise de sinais e sistemas.

3.2.1) - Definição e aplicações

Definição: consideremos as funções contínuas $x(t)$ e $h(t)$, a convolução $y(t)$ de $x(t)$ com $h(t)$ é definida por

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau. \quad (3.5)$$

Aplicações

Uma das nossas principais preocupações com funções do tempo (t) ou sinais, é estudar como elas são afetadas ao passarem por sistemas físicos tais como filtros elétricos. A palavra "sistema" significará para nós qualquer dispositivo que excitado pela "função" de entrada $x(t)$ apresenta na sua saída a "função" do tempo $y(t)$. Em particular, para a classe de sistemas lineares estacionários, $y(t)$ é obtido através da convolução de $x(t)$ com a resposta ao impulso $h(t)$. Por isso, as principais aplicações da convolução estão no estudo do comportamento de sistemas lineares estacionários no domínio do tempo.

3.2.2) - Impulso unitário

A equação (3.5) mostra que a convolução de duas funções é uma nova função, no entanto, quando uma das funções é a função delta de Dirac $\delta(t)$ ou impulso unitário, fazendo a convolução de uma função, por exemplo $h(t)$, com um impulso, simplesmente reproduz esta função, isto é

$$h(t) * \delta(t) = h(t) .$$

Realmente, $\delta(t)$ não é uma função do ponto de vista estritamente matemático. Contudo, $\delta(t)$ pertence à classe especial conhecida como funções generalizadas ou distribuições [2].

3.2.3) - A convolução integral

A fórmula (3.5) define a convolução integral $y(t)$ das funções $x(t)$ e $h(t)$.

O teorema da convolução: Sejam $x(t)$ e $h(t)$ duas funções cujas transformadas de Fourier são dadas por $X(f)$ e $H(f)$ respectivamente. A convolução $x(t) * h(t)$ tem a transformada de Fourier $X(f) H(f)$ [2].

$$\text{Notação: } x(t) * h(t) \Leftrightarrow X(f)H(f)$$

O teorema da convolução da frequência: consideremos $x(t)$, $h(t)$, $X(f)$ e $H(f)$ nas condições do teorema acima. A transformada de Fourier do produto $x(t)h(t)$ é igual a convolução $X(f) * H(f)$ [2].

$$\text{Notação: } x(t)h(t) \Leftrightarrow X(f) * H(f)$$

3.2.4) - A convolução periódica discreta

Sejam $\bar{x}(m)$ e $\bar{h}(n)$ seqüências periódicas com período N ,

$$\bar{x}(m) = \bar{x}(m + kN)$$

$$k \in \mathbf{Z}$$

$$\bar{h}(n) = \bar{h}(n + kN)$$

definimos a convolução periódica discreta de $\bar{x}(m)$ e $\bar{h}(n)$ por

$$\bar{y}(n) = \sum_{i=0}^{N-1} \bar{x}(i) \bar{h}(n-i)$$

ou

$$\bar{y}(n) = \bar{x}(m) * \bar{h}(n) .$$

E interessante observar que o produto das duas seqüências também é periódico, e que a somatória é efetuada apenas em um período, de duração N .

3.2.5) - A convolução linear e convolução circular

Consideremos duas sequências de dados x_m e h_n , ambas com N pontos.

Definição: A convolução linear y_r de x_m com h_n é definida por

$$y_r = \sum_{n=0}^{N-1} h_n x_{r-n}, \quad r = 0, \dots, 2N-2 \quad (3.7)$$

Definição: A convolução circular y_r de x_m com h_n é definida por

$$y_r = \sum_{n=0}^{N-1} h_n x_{r-n}, \quad r = 0, \dots, N-1 \quad (3.8)$$

onde $(r - n)$ é definido módulo N .

Pode-se observar que se x_m e h_n correspondem a um período das sequências periódicas $\bar{x}(m)$ e $\bar{h}(n)$ respectivamente, então a convolução circular y_r corresponde a um período da convolução periódica $\bar{y}(n)$.

De (3.7) concluímos que a convolução linear de duas sequências de comprimento N é uma sequência de comprimento $2N - 1$ e, de (3.8) concluímos que a convolução circular de duas sequências de comprimento N é uma sequência de comprimento N .

Em geral, a convolução linear é possível mesmo que as sequências x_m e h_n tenham comprimentos diferentes, por exemplo M e N e, neste caso, a convolução linear de x_m com h_n é uma sequência de comprimento $M + N - 1$. Por outro lado, a convolução circular só será possível, se ambas as sequências h_n e x_m tiverem o mesmo comprimento.

A convolução linear pode ser obtida da convolução circular, bastando para isso, apenas completar com zeros os elementos da sequência que possuir menor comprimento. Como os algoritmos que computam a convolução rápida (cap. 4) exigem que o comprimento N de cada sequência x_m e h_n seja potência de 2, às vezes, é necessário completar com zeros os elementos de ambas as sequências, até que se tenha um valor de N conveniente, sem que isto cause nenhum prejuízo tanto do ponto de vista matemático ou computacional, como também, do ponto de vista de aplicação.

3.2.6) - A convolução contínua e a convolução discreta

Se considerarmos somente funções periódicas representadas por funções impulso igualmente espaçadas, a convolução contínua e a convolução discreta relacionam-se identicamente e equivalentemente. No entanto, é possível aproximar a convolução contínua de uma forma de onda geral através da convolução discreta [2].

CAPITULO IV

ALGORITMOS DE TRANSFORMAÇÃO RAPIDA DE FOURIER

Vamos calcular a DFT \bar{X}_k de uma sequência x_m de N termos com $N = N_1 N_2 = 2^\delta$, $\delta = 1, 2, 3, \dots$, e,

$$\bar{X}_k = \sum_{m=0}^{N-1} x_m W^{mk}; \quad k = 0, \dots, N-1; \quad W = e^{-j2\pi/N}; \quad j = \sqrt{-1}. \quad (4.1)$$

4.1) - O Algoritmo de Cooley-Tukey

Para justificar intuitivamente o algoritmo, usaremos fatoração de matrizes. As matrizes fatoradas, são representadas alternadamente por um sinal gráfico. Para esses gráficos, nós construiremos a lógica de um programa de computador.

4.1.1) - Notação matricial

Por conveniência, usaremos a DFT (4.1) com a formulação

$$X(n) = \sum_{k=0}^{N-1} x_0(k) W^{nk}, \quad n = 0, \dots, N-1, \quad (4.1')$$

onde, $W = e^{-j2\pi/N}$.

É fácil ver que a equação (4.1') pode ser escrita na forma de N equações lineares. Faremos o desenvolvimento para $N = 4$. A equação (4.1'), pode ser escrita na forma:

$$\begin{aligned} X(0) &= x_0(0)W^0 + x_0(1)W^0 + x_0(2)W^0 + x_0(3)W^0 \\ X(1) &= x_0(0)W^0 + x_0(1)W^1 + x_0(2)W^2 + x_0(3)W^3 \\ X(2) &= x_0(0)W^0 + x_0(1)W^2 + x_0(2)W^4 + x_0(3)W^6 \\ X(3) &= x_0(0)W^0 + x_0(1)W^3 + x_0(2)W^6 + x_0(3)W^9 \end{aligned}$$

cuja representação matricial é:

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} x_0(0) \\ x_0(1) \\ x_0(2) \\ x_0(3) \end{bmatrix} \quad (4.2)$$

ou mais compactadamente,

$$\underline{X}(n) = W^{nk} \underline{x}_0(k).$$

Já que W é complexo e possivelmente x_0 , examinando (4.2) vemos que são necessárias N^2 multiplicações e $N(N-1)$ adições complexas para computarmos o X desejado. Contudo, pode-se reduzir o número de multiplicações e adições necessárias à computação de (4.2) [2].

4.1.2) - Fatoração de matriz

Para ilustrarmos o algoritmo, é conveniente escolher um número de pontos de amostras de $x_0(k)$ de acordo com a relação $N = 2^\delta$, $\delta = 1, 2, \dots$. Para a equação (4.2) nós temos $N = 4 = 2^\delta = 2^2$. Observamos que $W^{nk} = W^{nk \bmod N}$ e então podemos reescrever (4.2) na forma:

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^0 & W^2 \\ 1 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x_0(0) \\ x_0(1) \\ x_0(2) \\ x_0(3) \end{bmatrix} \quad (4.3)$$

Podemos fatorar a matriz quadrada de (4.3) de modo que

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} x_0(0) \\ x_0(1) \\ x_0(2) \\ x_0(3) \end{bmatrix} \quad (4.4)$$

Este método de fatoração é baseado no desenvolvimento teórico do algoritmo [2].

Se nós multiplicarmos as matrizes quadradas em (4.4) encontraremos a matriz quadrada de (4.3), observando-se apenas que as linhas 1 e 2 aparecem permutadas entre si (as colunas são permutadas de 0 a N-1). Esta permuta também fez com que as linhas 1 e 2 da matriz do lado esquerdo de (4.4) sejam permutadas entre si. Consideremos agora um novo vetor

$$Y(n) = \begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} \quad (4.5)$$

É fácil ver que a equação (4.4) reproduz a equação (4.3) a menos das linhas 1 e 2 que aparecem permutadas entre si. Esta fatoração, é a chave da eficiência do algoritmo FFT.

Passamos a examinar o número de multiplicações necessárias para computar (4.4). Em primeiro lugar, consideremos $x_1(k)$ como sendo o produto das duas matrizes que estão mais à direita em (4.4), isto é:

$$\begin{bmatrix} x_1(0) \\ x_1(1) \\ x_1(2) \\ x_1(3) \end{bmatrix} = \begin{bmatrix} 1 & 0 & w^0 & 0 \\ 0 & 1 & 0 & w^0 \\ 1 & 0 & w^2 & 0 \\ 0 & 1 & 0 & w^2 \end{bmatrix} \begin{bmatrix} x_0(0) \\ x_0(1) \\ x_0(2) \\ x_0(3) \end{bmatrix} \quad (4.6)$$

O elemento $x_1(0)$ é computado com uma multiplicação complexa e uma adição complexa (W^0 não deve ser reduzido à unidade no caso de desenvolvimento generalizado), então

$$x_1(0) = x_0(0) + W^0 x_0(2). \quad (4.7)$$

O elemento $x_1(1)$ também será computado com uma multiplicação complexa e uma adição complexa. Somente uma adição complexa é necessária para computar $x_1(2)$. Isto segue do fato de que $W^0 = -W^2$, conseqüentemente,

$$\begin{aligned} x_1(2) &= x_0(0) + W^2 x_0(2) \\ &= x_0(0) - W^0 x_0(2), \end{aligned}$$

onde a multiplicação complexa $W^0 x_0(2)$ já havia sido computada na determinação de $x_1(0)$, equação (4.7). Pela mesma razão $x_1(3)$ é computada por uma única adição complexa. Logo, o vetor intermediário $x_1(k)$ é então determinado por 4 adições complexas e duas multiplicações complexas.

Continuando a computação de (4.4) tem-se:

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} = \begin{bmatrix} x_2(0) \\ x_2(1) \\ x_2(2) \\ x_2(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} x_1(0) \\ x_1(1) \\ x_1(2) \\ x_1(3) \end{bmatrix} \quad (4.8)$$

O termo $x_2(0)$ é determinado por uma adição complexa e uma multiplicação complexa

$$x_2(0) = x_1(0) + W^0 x_1(1).$$

O termo $x_2(1)$ é computado com uma única adição complexa em virtude de $W^0 = -W^2$. De modo análogo ao que foi explicado acima, tem-se que: $x_2(2)$ é determinado por uma multiplicação complexa e uma adição complexa e $x_2(3)$ é determinado com apenas uma adição complexa.

A computação de $Y(n)$ por meio da equação (4.4) requer um total de 4 multiplicações complexas e 8 adições complexas. Enquanto que a computação de $X(n)$ por meio da equação (4.2) requer 16 multiplicações complexas e 12 adições complexas. Notemos que o processo de fatoração da matriz, introduz a matriz fatorada e, como resultado reduz o número de multiplicações. Para $N = 4$, o processo de fatoração da matriz reduz o número de multiplicações por um fator de 2. Como o tempo de computação é proporcional ao número de multiplicações envolvidas, esta é a razão da eficiência desse algoritmo.

Para qualquer $N = 2^\delta$, $\delta = 1, 2, \dots$, o procedimento do algoritmo é sempre o mesmo, observando-se apenas que a fatoração da matriz $N \times N$ é feita sempre em δ matrizes $N \times N$, tal que cada matriz fatorada tem a propriedade especial de minimizar o número de multiplicações complexas e o número de adições complexas. Para $N = 2^\delta = 2^2 = 4$, o algoritmo requer $N\delta/2 = 4$ multiplicações

complexas e $N\delta = 4.2 = 8$ adições complexas. Enquanto que o método direto pela equação (4.3), requer N^2 multiplicações complexas e $N(N-1)$ adições complexas. Se nós assumirmos que o tempo de computação é proporcional ao número de multiplicações, então a razão aproximadamente direta do tempo de computação do algoritmo será dada por [2]

$$\frac{N^2}{N\delta/2} = \frac{2N}{\delta} .$$

A figura (4.1) ilustra a relação entre o número de multiplicações utilizada neste algoritmo pelo o número de multiplicações utilizadas no método direto.

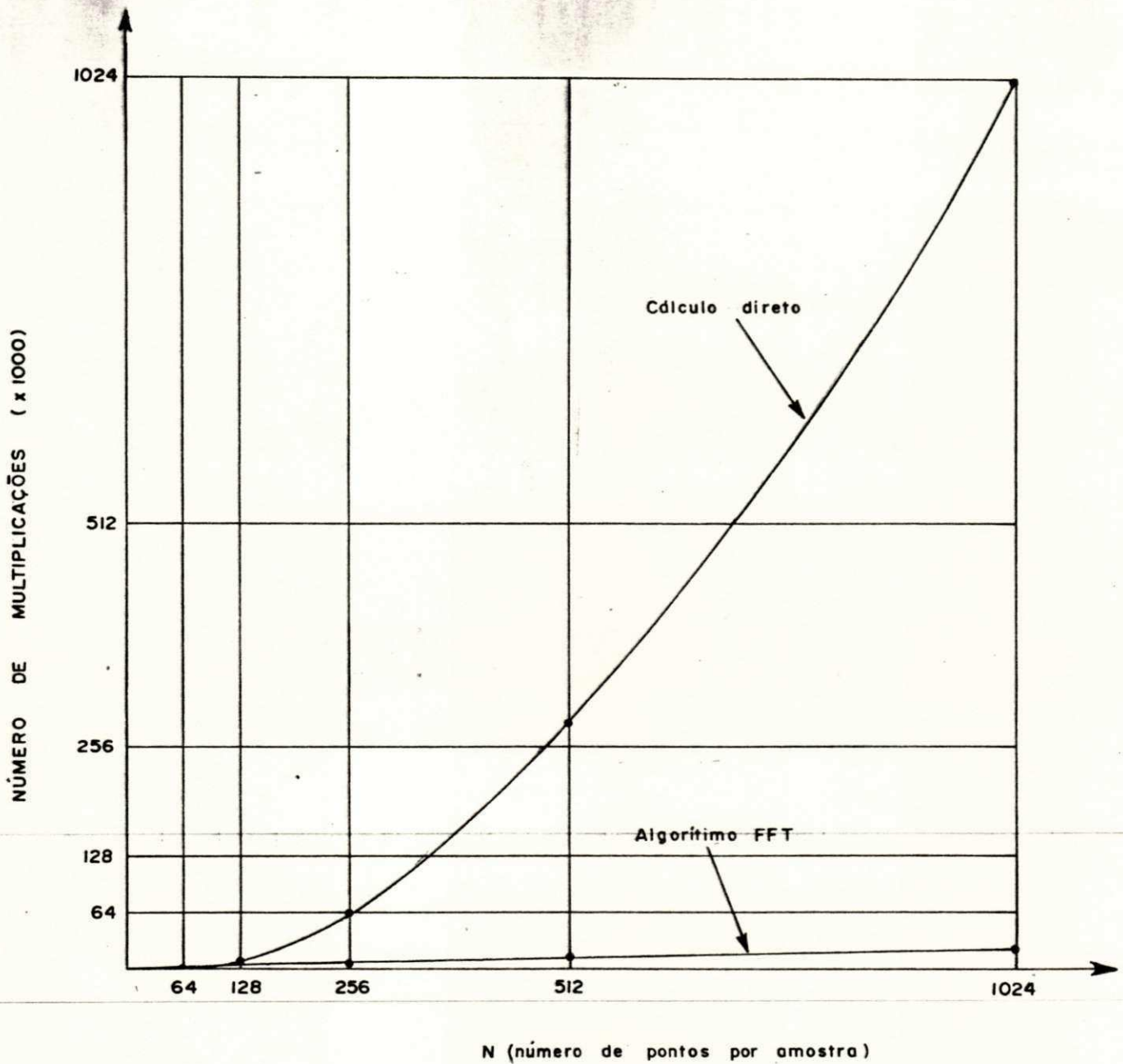


Figura 4.1 - Comparação do número de multiplicações necessárias pelo cálculo direto e o algoritmo de Cooley-Tukey (o FFT).

4.1.3) - Implicações da fatoração de matriz

O procedimento da fatoração de matriz produz um pequeno problema na computação de (4.4) pois, ele produz $Y(n)$ em lugar de $X(n)$, isto é,

$$Y(n) = \begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} \text{ em lugar de } X(n) = \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix}$$

porém esse reordenamento é inerente ao processo de fatoração da matriz e não constitui um problema, pois, é possível a generalização de uma técnica para obtenção de $Y(n)$ e, conseqüentemente, obteremos $X(n)$ de $Y(n)$.

4.1.3.1) - Obtenção de $X(n)$ a partir de $Y(n)$

Vamos escrever os argumentos de $Y(n)$ em binário com $\delta = 2$ bits, isto é,

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} \text{ torna-se } \begin{bmatrix} X(00) \\ X(10) \\ X(01) \\ X(11) \end{bmatrix} \quad (4.9)$$

Se revertermos a posição dos bits em (4.9) ou seja: 00 torna-se 01, 10 torna-se 01, 01 torna-se 10 e 11 torna-se 11, então

$$Y(n) = \begin{bmatrix} X(00) \\ X(10) \\ X(01) \\ X(11) \end{bmatrix} \text{ será mudado para } \begin{bmatrix} X(00) \\ X(01) \\ X(10) \\ X(11) \end{bmatrix} = X(n). \quad (4.10)$$

A equação mais a direita em (4.10) mostra que o desenvolvimento geral direto, resulta de uma mudança do FFT [2].

4.1.4) - Representação gráfica e sua interpretação

Para N maior que 4 torna-se muito trabalhoso descrever o processo de fatoração da matriz, análogo ao da equação (4.4). Por essa razão vamos interpretar (4.4) através de um gráfico. Usando essa formulação gráfica, podemos descrever com generalidade suficiente para o desenvolvimento gráfico de um programa de computador.

Vamos converter a equação (4.4) para um gráfico, como ilustra a figura (4.2). Como observamos, o vetor de dados ou conjunto $x_0(k)$ é representado por uma coluna vertical de nodos a esquerda do gráfico. O segundo conjunto vertical de nodos é o vetor $x_1(k)$ computado na equação (4.6), e o próximo conjunto vertical corresponde ao vetor $x_2(k) = Y(n)$, equação (4.8). Em

geral, existirão δ conjuntos computacionais onde, $N = 2^\delta$.

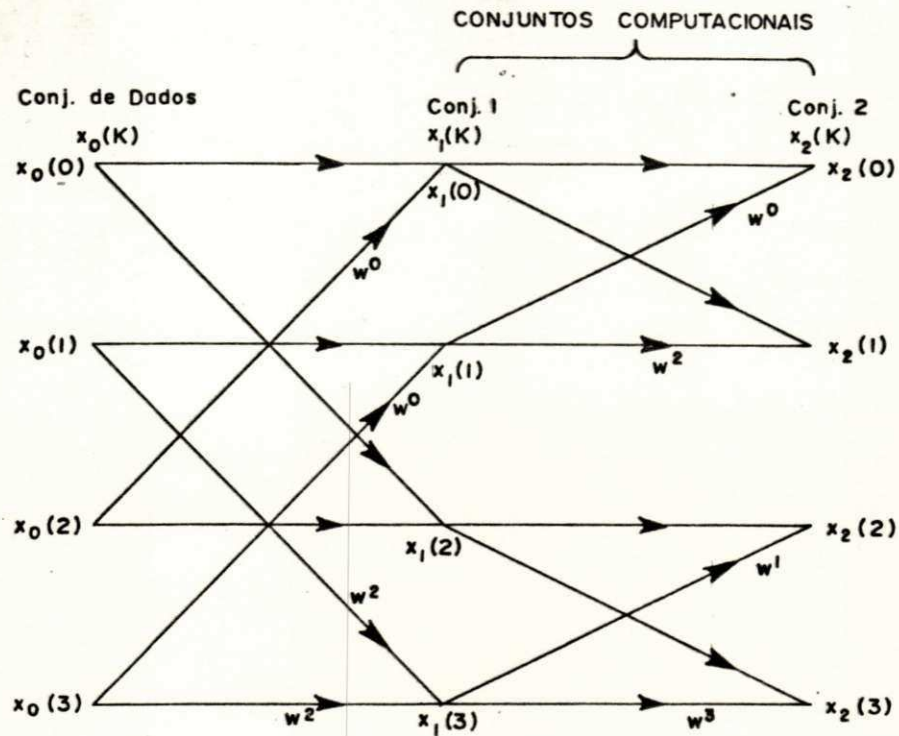


Figura 4.2 - Representação gráfica, $N = 4$, para o algoritmo de Cooley-Tukey.

Chamemos de nodos os pontos $x_1(0), \dots, x_1(N), x_2(0), \dots, x_2(N)$. Todos os nodos são interpretados da mesma forma. Por exemplo, da equação (4.7) temos que

$$x_1(0) = x_0(0) + w^0 x_0(2)$$

e isto, é o que mostra a Figura (4.2).

Outro exemplo: $x_1(2) = x_0(0) + W^2 x_0(2)$.

4.2) - O Algoritmo FFT BASE-2

Vamos escolher $N_1 = 2$ e $N_2 = 2^{\delta} - 1$. Isto é equivalente a dividir a sequência de entrada x_m de N pontos em duas sequências de $N/2$ pontos x_{2m} e x_{2m+1} , correspondendo respectivamente às amostras pares e ímpares de x_m . Nestas condições a equação (4.1) torna-se

$$\bar{X}_k = \sum_{m=0}^{N/2-1} x_{2m} W^{2mk} + W^k \sum_{m=0}^{N/2-1} x_{2m+1} W^{2mk}$$

e, já que $W^{N/2} = -1$,

$$\bar{X}_{k+N/2} = \sum_{m=0}^{N/2-1} x_{2m} W^{2mk} - W^k \sum_{m=0}^{N/2-1} x_{2m+1} W^{2mk}$$

$k = 0, \dots, N/2-1$.

Essa abordagem é chamada decimação no tempo [14], e neste caso, a DFT de N - pontos é representada por 2 DFTs de comprimentos $N/2$ mais N adições e $N/2$ multiplicações por W^k . Com o mesmo procedimento, podemos representar as 2 DFTs de comprimentos $N/2$ por 4 DFTs de comprimentos $N/4$ mais N adições e $N/2$ multiplicações. A aplicação sistemática deste método, computa a DFT de comprimento 2^{δ} em $\delta = \log_2 N$ estágios onde cada estágio converte 2^i DFTs de comprimentos $2^{\delta-i}$ para 2^{i+1} DFTs de comprimentos $2^{\delta-i-1}$ mais N adições e $N/2$ multiplicações. Consequentemente, o número de multiplicações complexas M e o

número de adições complexas A para computar a DFT de comprimento N pelo algoritmo Base-2 é [13].

$$M = \frac{N\delta}{2} \text{ e } A = N\delta.$$

A segunda forma desse algoritmo pode ser obtida por uma simples divisão da sequência de entrada x_m de N pontos, em duas sequências de $N/2$ pontos x_m e $x_{m+N/2}$ correspondendo respectivamente às $N/2$ primeiras amostras e as $N/2$ amostras seguintes de x_m . Essa abordagem é chamada decimação na frequência [14] e, neste caso, a equação (4.1) torna-se

$$\bar{X}_k = \sum_{m=0}^{N/2-1} (x_m + W^{Nk/2} x_{m+N/2}) W^{mk}$$

Vamos computar separadamente o número de amostras pares e ímpares, isto é:

$$\bar{X}_{2k} = \sum_{m=0}^{N/2-1} (x_m + x_{m+N/2}) W^{2mk}, \quad k \text{ par}, \quad k = 0, \dots, N/2-1 \quad (4.11)$$

e

$$\bar{X}_{2k+1} = \sum_{m=0}^{N/2-1} (x_m - x_{m+N/2}) W^m W^{2mk}, \quad k \text{ ímpar}, \quad k = 0, \dots, N/2-1. \quad (4.12)$$

Assim, \bar{X}_k é computada por (4.11) e (4.12) em termos de 2 DFTs de comprimento $N/2$, porém, com uma premultiplicação por W^m da sequência de entrada x_m em (4.12). Logo, a computação da DFT de N termos é representada por 2 DFTs de $N/2$ termos mais N adições complexas e $N/2$ multiplicações complexas [13].

Podemos ver que os algoritmos de decimação no tempo e na frequência têm a mesma estrutura, diferindo apenas nos coeficientes [13].

4.3) - O Algoritmo FFT BASE-4

Vamos escolher $N_1 = 4$ e $N_2 = 2^{\delta-2}$. Isto é equivalente a dividir os N pontos da sequência de entrada x_m em 4 sequências de $N/4$ pontos correspondendo a x_{4m} , x_{4m+1} , x_{4m+2} e x_{4m+3} para $m = 0, \dots, N/4-1$. Com esta partição a equação (4.1) torna-se:

$$\bar{X}_k = \sum_{r=0}^3 W^{rk} \sum_{m=0}^{N/4-1} x_{4m+r} W^{4mk}$$

e, já que $W^{N/4} = -j$

$$\bar{X}_{k+N/4} = \sum_{r=0}^3 (-j)^r W^{rk} \sum_{m=0}^{N/4-1} x_{4m+r} W^{4mk}$$

$$\bar{X}_{k+N/2} = \sum_{r=0}^3 (-1)^r W^{rk} \sum_{m=0}^{N/4-1} x_{4m+r} W^{4mk}$$

$$\bar{X}_{k+3N/4} = \sum_{r=0}^3 j^r W^{rk} \sum_{m=0}^{N/4-1} x_{4m+r} W^{4mk}, \quad k = 0, \dots, N/4-1.$$

Conseqüentemente, o algoritmo Base-4, transforma uma DFT de N pontos em 4 DFTs de N/4 pontos [13]. O mesmo procedimento pode ser aplicado recursivamente em $\delta/2$ estágios, reduzindo as dimensões das DFTs por um fator de 4. Assim a DFT é computada pelo algoritmo Base-4 com $N\delta/2$ multiplicações complexas e $3N \delta/2$ adições complexas. Esse número é mais elevado do que aquele do Base-2. No entanto, a complexidade computacional pode ser drasticamente reduzida, levando-se em consideração as simetrias das funções seno e cosseno [13].

A tabela 4.1 mostra que o algoritmo Base-4 reduz o número de multiplicações em até 25% em comparação com o algoritmo Base-2, enquanto que o número de adições é aproximadamente o mesmo. O número de multiplicações pode ser diminuído ainda mais se usarmos os algoritmos Base-8 ou Base-16 [13].

Tabela 4.1 - Número de operações reais não triviais para os algoritmos FFTs Base-2 e Base-4.

N	FFT Base-2		FFT Base-4	
	número de multiplicações	número de adições	número de multiplicações	número de adições
4	0	16	0	16
16	24	152	20	148
64	264	1032	208	976
256	1800	5896	1392	5488
1024	10248	30728	7856	28336
4096	53256	151560	40624	138928

4.4) - O Algoritmo FFT de Rader-Brenner

A avaliação de DFTs pelos algoritmos FFT anteriormente mencionados requer multiplicações complexas. Nós veremos agora que uma simples modificação do algoritmo FFT substitui essas multiplicações complexas por multiplicações de um número complexo por um número real ou um número imaginário puro [13]. Isto é feito computando-se a DFT definida em (4.1) através do algoritmo de decimação na frequência Base-2, fórmulas (4.11) e (4.12).

Os cálculos da DFT definida em (4.12) serão simplificados se nós definirmos uma sequência auxiliar a_m de $N/2$ pontos por

$$a_m = (x_m - x_{m+N/2}) / [2 \cos(2\pi m/N)], \quad m \neq 0, N/4$$

$$a_0 = 0 \quad \text{e} \quad a_{N/4} = 0$$

e computarmos sua DFT \bar{A}_k de $N/2$ pontos [13].

A Tabela 4.2, quando comparada à Tabela 4.1, mostra que o algoritmo de Rader-Brenner reduz o número de multiplicações dos algoritmos Base-2 e Base-4, enquanto requer 10% a mais no número de adições [13].

Tabela 4.2 - Número de multiplicações reais não triviais, para computar DFTs complexos pelo método Rader-Brenner.

N	número de multiplicações reais	número de adições reais
8	4	52
16	20	148
32	68	424
64	196	1104
128	516	2720
256	1284	6464
512	3076	14976
1024	7172	34048
2048	16388	76288

4.5) - O Algoritmo de Bruun

Este algoritmo possui uma grande importância e significação prática, visto que, dada uma sequência x_m de dados reais com N pontos, $N = 2^\delta$, $\delta = 1, 2, 3, \dots$ podemos computar sua DFT \bar{X}_k quase que inteiramente com aritmética real, simplificando assim a implementação de DFTs de sequências de dados reais. Aqui, será apresentada uma versão do algoritmo original, a qual permite computar a DFT \bar{X}_k de x_m através de divisão polinomial.

A fim de desenvolvermos o algoritmo, usaremos (4.1) com uma representação polinomial da DFT definida pelas duas equações seguintes:

$$X(z) \equiv \sum_{m=0}^{N-1} x_m z^m \quad \text{mod} \quad (z^N - 1) \quad (4.13)$$

e

$$\bar{X}_k(z) \equiv X(z) \quad \text{mod} \quad (z - W^k). \quad (4.14)$$

As equações (4.13) e (4.14) são equivalentes à equação (4.1). A definição de (4.14) módulo $(z - W^k)$ significa que podemos representar z por W^k em (4.13). Nestas condições a definição de (4.13) módulo $(z^N - 1)$ não é necessária, contudo, esta definição é válida porque $z^N = W^{kN} = 1$ ($W^{kN} = \cos(2\pi k) - j\sin(2\pi k) = 1$). É fácil ver que as N raízes complexas de $z^N - 1$ são dadas por W^k , $k = 0, \dots, N-1$, com

$$z^N - 1 = \prod_{k=0}^{N-1} (z - W^k).$$

Além disso, como $N = 2^\delta$, podemos expressar $z^N - 1$ como um produto de dois polinômios de $N/2$ termos em z , isto é:

$$z^N - 1 = (z^{N/2} - 1)(z^{N/2} + 1)$$

com

$$z^{N/2} - 1 = \prod_{k_1=0}^{N/2-1} (z - W^{2k_1}), \quad k_1 = 0, \dots, N/2-1$$

e

$$z^{N/2} + 1 = \prod_{k_1=0}^{N/2-1} (z - W^{2k_1+1}), \quad k_1 = 0, \dots, N/2-1.$$

Consequentemente, para k par, todos os valores de W^k correspondem ao polinômio $z^{N/2} - 1$ e, podemos representar (4.13) e (4.14) por

$$X_1(z) \equiv \sum_{m=0}^{N/2-1} (x_m + x_{m+N/2}) z^m \equiv X(z) \pmod{(z^{N/2} - 1)} \quad (4.15)$$

e

$$\bar{X}_k(z) \equiv X_1(z) \pmod{(z - W^k)}, \quad k \text{ par} . \quad (4.16)$$

De modo análogo, para k ímpar, todos os valores de W^k , correspondem ao polinômio $z^{N/2} + 1$ e, podemos representar (4.13) e (4.14) por

$$X_2(z) \equiv \sum_{m=0}^{N/2-1} (x_m - x_{m+N/2}) z^m \equiv X(z) \pmod{(z^{N/2} + 1)} \quad (4.17)$$

e

$$\bar{X}_k(z) \equiv X_2(z) \pmod{(z - W^k)}, \quad k \text{ ímpar} . \quad (4.18)$$

As fórmulas (4.15) e (4.16) representam uma DFT par de $N/2$ termos enquanto que (4.17) e (4.18) representam uma DFT ímpar de $N/2$ termos. De modo que as fórmulas (4.15) a (4.18) podem ser vistas como sendo o primeiro estágio da decomposição do FFT de decimação na frequência. Nos próximos estágios, nós usaremos o fato de que qualquer polinômio da forma $z^{4q} + az^{2q} + 1$, onde a é um número real qualquer, fatora-se em dois polinômios reais,

$$z^{4q} + az^{2q} + 1 = (z^{2q} + \sqrt{2-a} z^q + 1)(z^{2q} - \sqrt{2-a} z^q + 1)$$

portanto, isto implica que:

$$z^{N/2} + 1 = (z^{N/4} + \sqrt{2} z^{N/8} + 1)(z^{N/4} - \sqrt{2} z^{N/8} + 1),$$

com

$$z^{N/4} + \sqrt{2} z^{N/8} + 1 = \prod_{k_1 \in B_1} (z - W^{2k_1+1})$$

e

$$z^{N/4} - \sqrt{2} z^{N/8} + 1 = \prod_{k_1 \in B_2} (z - W^{2k_1+1}).$$

onde B_1 é o conjunto de $N/4$ valores de k_1 tais que W^{2k_1+1} é uma raiz de $z^{N/4} + \sqrt{2} z^{N/8} + 1$ e, B_2 é o conjunto de $N/4$ outros valores de k_1 . Nestas condições, a DFT impar representada por (4.17) e (4.18) pode ser substituída por

$$X_3(z) \equiv X_2(z) \pmod{(z^{N/4} + \sqrt{2} z^{N/8} + 1)},$$

$$\bar{X}_{2k_1+1}(z) \equiv X_3(z) \pmod{(z - W^{2k_1+1})}, \quad k_1 \in B_1,$$

e

$$X_4(z) \equiv X_2(z) \pmod{(z^{N/4} - \sqrt{2} z^{N/8} + 1)},$$

$$\bar{X}_{2k_1+1}(z) \equiv X_4(z) \pmod{(z - W^{2k_1+1})}.$$

O mesmo processo de decomposição pode ser representado sistematicamente para expressar um polinômio da forma $z^{4q} + az^{2q} + 1$ como um produto de dois polinômios reais de grau $2q$, até que os polinômios sejam reduzidos para grau 2. Os termos de saída das duas DFTs são obtidos por cada polinômio de grau 2, representando z com as duas raízes complexas do polinômio. Esse processo é resumido na Figura 4.3 para uma DFT de 8 termos. Neste diagrama, cada retângulo representa uma redução módulo o polinômio nele indicado.

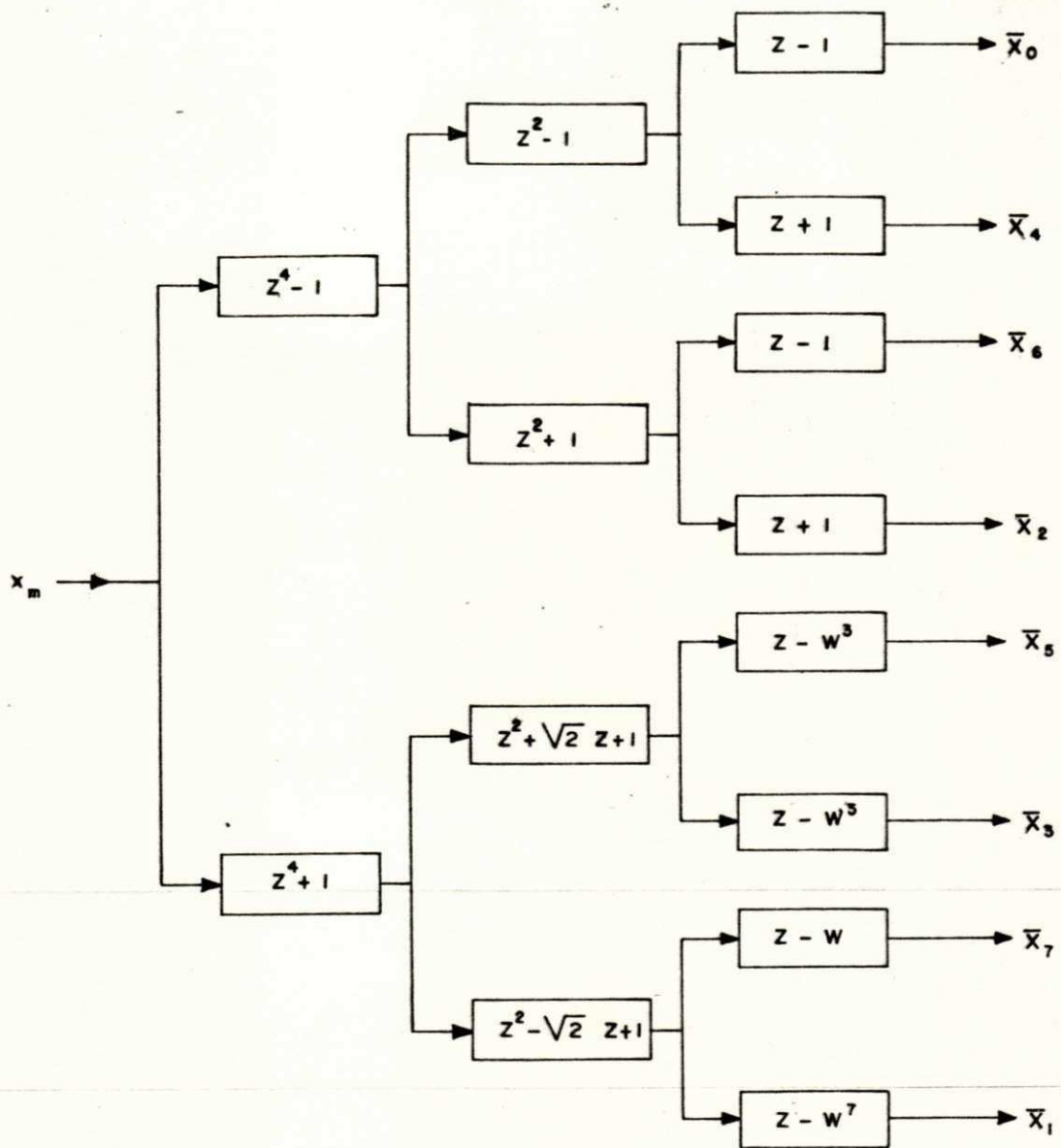


Figura 4.3 - Computação da DFT de 8 pontos pelo algoritmo de Bruun.

4.5.1) - Desenvolvimento completo do algoritmo para $N = 16$

Da trigonometria, sabemos que

$$\cos \frac{x}{2} = \sqrt{\frac{1 + \cos x}{2}} \quad (4.19)$$

e

$$\sin \frac{x}{2} = \sqrt{\frac{1 - \cos x}{2}} . \quad (4.20)$$

Como $W = e^{-j2\pi/N}$ e $N = 16$, temos que $W = e^{-\pi j/8}$ o que implica que $W^k = e^{-jk\pi/8}$, ou seja:

$$W^k = \cos(k\pi/8) - j\sin(k\pi/8), \quad k = 0, \dots, 15 . \quad (4.21)$$

Como mostra a Figura 4.3, a saída da DFT satisfaz a uma determinada ordem. Para determinarmos essa ordem, para $N = 16$, vamos fatorar todos os polinômios que aparecem no penúltimo estágio (polinômios de grau 2).

4.5.1.1) - Fatoração dos polinômios de grau 2

$$z^2 - 1 = (z-1)(z+1) = (z-W^0)(z-W^8)$$

$$z^2 + 1 = (z-j)(z+j) = (z-W^{12})(z-W^4)$$

$$z^2 + \sqrt{2}z + 1 = (z - (-\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}))(z - (-\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2})) = (z-W^{10})(z-W^6)$$

$$z^2 - \sqrt{2}z + 1 = (z - (\frac{\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}))(z - (\frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2})) = (z-W^{14})(z-W^2)$$

$$z^2 + \sqrt{2-\sqrt{2}}z + 1 = \left(z - \left(-\frac{\sqrt{2-\sqrt{2}}}{2} + j\frac{\sqrt{2+\sqrt{2}}}{2}\right)\right) \left(z - \left(-\frac{\sqrt{2-\sqrt{2}}}{2} - j\frac{\sqrt{2+\sqrt{2}}}{2}\right)\right) = (z - W^{11})(z - W^5)$$

$$z^2 - \sqrt{2-\sqrt{2}}z + 1 = \left(z - \left(\frac{\sqrt{2-\sqrt{2}}}{2} + j\frac{\sqrt{2+\sqrt{2}}}{2}\right)\right) \left(z - \left(\frac{\sqrt{2-\sqrt{2}}}{2} - j\frac{\sqrt{2+\sqrt{2}}}{2}\right)\right) = (z - W^{13})(z - W^3)$$

$$z^2 + \sqrt{2+\sqrt{2}}z + 1 = \left(z - \left(-\frac{\sqrt{2+\sqrt{2}}}{2} + j\frac{\sqrt{2-\sqrt{2}}}{2}\right)\right) \left(z - \left(-\frac{\sqrt{2+\sqrt{2}}}{2} - j\frac{\sqrt{2-\sqrt{2}}}{2}\right)\right) = (z - W^9)(z - W^7)$$

$$z^2 - \sqrt{2+\sqrt{2}}z + 1 = \left(z - \left(\frac{\sqrt{2+\sqrt{2}}}{2} + j\frac{\sqrt{2-\sqrt{2}}}{2}\right)\right) \left(z - \left(\frac{\sqrt{2+\sqrt{2}}}{2} - j\frac{\sqrt{2-\sqrt{2}}}{2}\right)\right) = (z - W^{15})(z - W^1)$$

4.5.1.2) Cálculo dos W^k

Esses cálculos são efetuados utilizando-se as fórmulas (4.19) a (4.21) e, levando-se em consideração a simetria das funções seno e cosseno, como ilustra a Figura 4.4.

Fazendo $\theta = k\pi/8$ em (4.21) então, basta calcular os valores de $\text{sen}\theta$ e $\text{cos}\theta$ para $k = 0, 1, 2, 3$.

i) Se $k = 0$, então $W^0 = \cos 0 - j\text{sen} 0 = 1 - 0j = 1$

ii) Se $k = 1$, então $W^1 = \cos(\pi/8) - j\text{sen}(\pi/8) = \frac{\sqrt{2+\sqrt{2}}}{2} - j\frac{\sqrt{2-\sqrt{2}}}{2}$,

pois, por (4.19), temos que:

$$\cos(\pi/8) = \sqrt{\frac{1+\cos(\pi/4)}{2}} = \sqrt{\frac{2+\sqrt{2}}{4}} = \frac{\sqrt{2+\sqrt{2}}}{2},$$

e, por (4.20) temos que:

$$\text{sen}(\pi/8) = \sqrt{\frac{1-\cos(\pi/4)}{2}} = \sqrt{\frac{2-\sqrt{2}}{4}} = \frac{\sqrt{2-\sqrt{2}}}{2}.$$

iii) Se $k = 2$, então $W^2 = \cos(\pi/4) - j\text{sen}(\pi/4) = \frac{\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}$.

Por (4.19) temos que:

$$\cos(3\pi/8) = \sqrt{\frac{1+\cos(3\pi/8)}{2}} = \sqrt{\frac{1-\cos(\pi/4)}{2}} = \frac{\sqrt{2-\sqrt{2}}}{2},$$

e por (4.20) temos que:

$$\text{sen}(3\pi/8) = \sqrt{\frac{1-\cos(3\pi/8)}{2}} = \sqrt{\frac{1+\cos(\pi/4)}{2}} = \frac{\sqrt{2+\sqrt{2}}}{2}, \text{ logo}$$

iv) Se $k = 3$, então $W^3 = \cos(3\pi/8) - j\text{sen}(3\pi/8) = \frac{\sqrt{2-\sqrt{2}}}{2} - j\frac{\sqrt{2+\sqrt{2}}}{2}$.

Os valores de W^k , $k = 4, 5, \dots, 15$ são obtidos por simetria das funções seno e cosseno sobre o círculo trigonométrico, como ilustra a Figura 4.4.

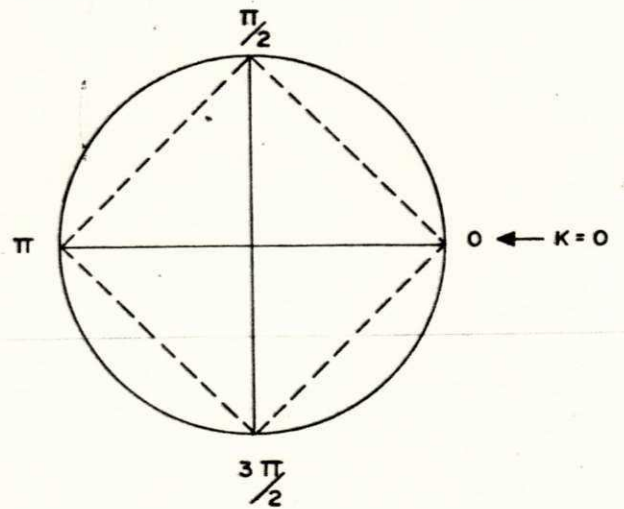
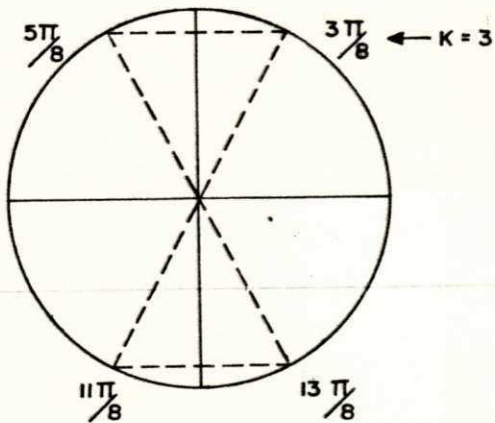
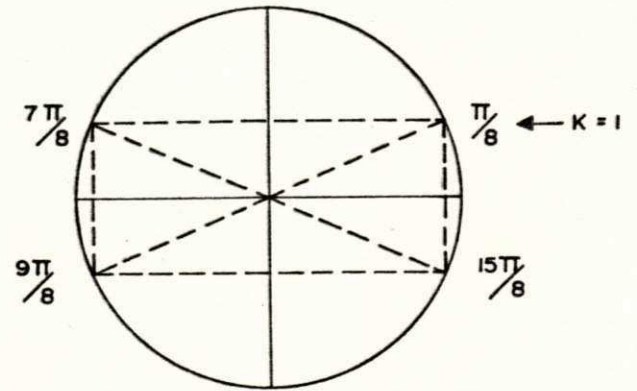
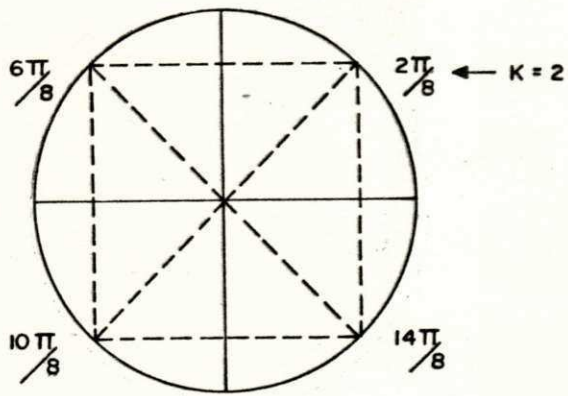


Figura 4.4 - Simetria das funções trigonométricas seno e cosseno, para $N = 16$.

5.1.3) - Diagrama completo para $N = 16$

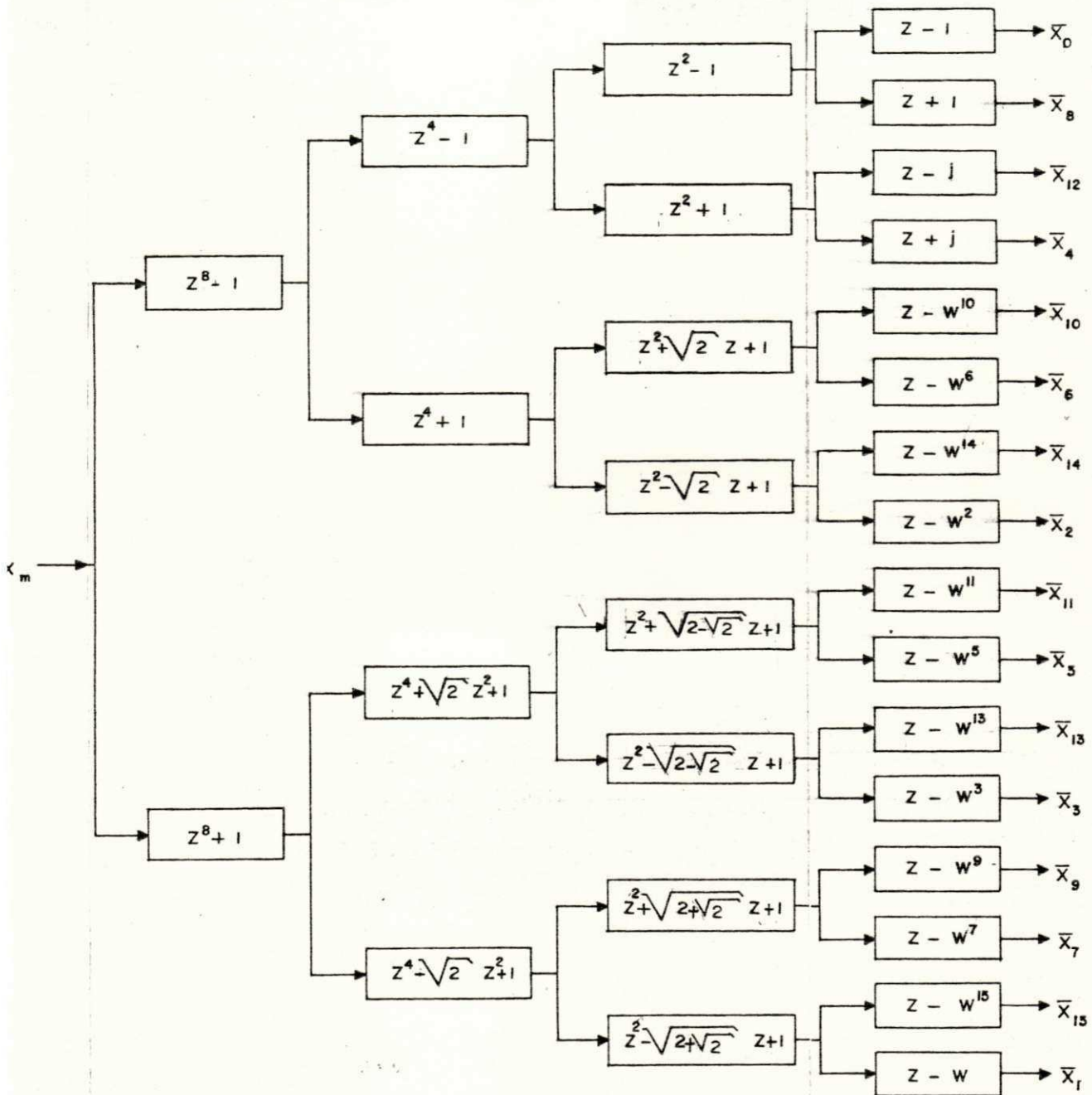


Figura 4.5 - Computação da DFT de 16 pontos pelo algoritmo de Bruun.

4.6) - DFTs Multidimensionais

A transformada de Fourier multidimensional é uma extensão direta da transformada de Fourier unidimensional [6].

Relacionaremos a seguir, alguns algoritmos ou métodos [13] de computação de DFT multidimensionais e, também alguns FFT unidimensionais.

Consideremos primeiro, uma DFT bidimensional de tamanho $N_1 \times N_2$, com

$$\bar{X}_{k_1, k_2} = \sum_{m_1=0}^{N_1-1} \sum_{m_2=0}^{N_2-1} x_{m_1, m_2} W_1^{m_1 k_1} W_2^{m_2 k_2} ,$$

$$W_1 = e^{-j2\pi/N_1}, W_2 = e^{-j2\pi/N_2}, k_1 = 0, \dots, N_1-1$$

$$\text{e } k_2 = 0, \dots, N_2-1 . \quad (4.22)$$

A fim de avaliarmos essa DFT, primeiro reescrevemos (4.22)

como

$$\bar{X}_{k_1, k_2} = \sum_{m_1=0}^{N_1-1} W_1^{m_1 k_1} \sum_{m_2=0}^{N_2-1} x_{m_1, m_2} W_2^{m_2 k_2} .$$

Inicialmente, nós avaliamos os N_1 DFTs \bar{Y}_{m_1, k_2} de comprimento N_2 que corresponde a N_1 valores distintos de m_1

$$\bar{Y}_{m_1, k_2} = \sum_{m_2=0}^{N_2-1} x_{m_1, m_2} W_2^{m_2 k_2} .$$

\bar{X}_{k_1, k_2} é então obtido, calculando-se N_2 DFTs de comprimento N_1 das N_2 sequências \bar{Y}_{m_1, k_2} correspondendo a N_2 valores distintos de k_2

$$\bar{X}_{k_1, k_2} = \sum_{m_1=0}^{N_1-1} \bar{Y}_{m_1, k_2} W_1^{m_1 k_2} .$$

Essa abordagem é chamada método linha-coluna porque é possível organizarmos os dados de entrada em linhas e colunas em um conjunto de tamanho $N_1 \times N_2$, computando-se primeiro a sequência de DFTs das colunas e depois a sequência de DFTs das linhas. Com essa técnica, uma DFT bidimensional é mapeada por N_1 DFTs de N_2 termos mais N_2 DFTs de N_1 termos, respectivamente. Se N_1 e N_2 são potências de 2, as DFTs unidimensionais de comprimento N_1 e N_2 podem ser avaliadas por qualquer um dos FFTs já mencionados. O mapeamento de DFTs multidimensionais para DFTs unidimensionais obtido por esse método é bom. No entanto, o algoritmo DFT reduzido, usado para computar DFTs multidimensionais por transformações polinomiais é melhor que este.

O método linha-coluna também se aplica a mais de duas dimensões, de modo que uma DFT d-dimensional de tamanho $N \times N \times N \dots \times N$ é calculada com dN^{d-1} DFTs de comprimento N .

Se $N = p$ ou $N = p^2$ com p primo, $p \neq 2$, podemos computar uma DFT de N termos através dos algoritmos de Rader.

Se N é o produto de d fatores mutuamente primos, isto é, $N = N_1 N_2 \dots N_d$ com $(N_i, N_j) = 1$ para $i \neq j$, podemos utilizar a técnica do FFT Fator Primo, proposta por Good [13], que tem como objetivo computar grandes DFTs de tamanho N por combinações dos vários DFTs de tamanho N_1, \dots, N_d , obtidos através do teorema do resto chinês. O algoritmo de Fatores Primos Separados, mostra que uma DFT de tamanho N , pode ser mapeada para uma DFT multidimensional de tamanho $N_1 \times N_2 \times \dots \times N_d$, enquanto que, o algoritmo Nesting de Winograd, além de computar DFTs de tamanho $N_1 \times N_2 \times \dots \times N_d$, pode também computar qualquer DFT multidimensional de tamanho $N_1 \times N_2 \times \dots \times N_d$, onde os fatores N_i são mutuamente primos entre si.

4.7) - Resumo

O algoritmo de Cooley-Tukey é computado, utilizando-se o processo de fatoração de matrizes. Os algoritmos Base-2, Base-4 e Rader-Brenner, utilizam na sua computação, o processo de partição da sequência de entrada, enquanto que, o algoritmo de Bruun, é computado utilizando-se restos de divisão de polinômios.

Apesar dos diferentes procedimentos computacionais, todos os FFTs aqui mencionados, podem ser implementados com aritmética real, com um tempo proporcional a $N \log_2 N$, em vez de N^2 pela definição (4.1), computando assim, eficientemente, uma DFT de N pontos e, produzindo o mesmo resultado. Além disso, esses algoritmos unidimensionais, são utilizados pelos métodos e ou

algoritmos de computação da transformada discreta de Fourier multidimensional.

CAPITULO V

A CONVOLUÇÃO RAPIDA

5.1) - Computação da Convolução Através do Teorema do Resto Chinês

Nosso objetivo é minimizar o número de multiplicações na computação da convolução circular, o que é possível, utilizando-se o Teorema do Resto Chinês. Consideremos a convolução circular y_r de N termos

$$y_r = \sum_{n=0}^{N-1} h_n x_{r-n}, \quad r = 0, \dots, N-1, \text{ com } (r-n) \text{ definido módulo } N,$$

onde x_m e h_n são as duas sequências de entrada de comprimento N. y_r pode ser vista como um produto polinomial módulo $(z^N - 1)$, isto é, se

$$X(z) = \sum_{m=0}^{N-1} x_m z^m,$$

$$H(z) = \sum_{n=0}^{N-1} h_n z^n ,$$

$$Y(z) \equiv X(z)H(z) \pmod{(z^N-1)},$$

$$\text{e portanto, } Y(z) = \sum_{r=1}^{N-1} Y_r z^r .$$

Para coeficientes no corpo dos números racionais z^N-1 é o produto de d polinômios ciclotômicos $P_i(z)$, onde d é o número de divisores de N , incluindo 1 e N , ou seja:

$$z^N-1 = \prod_{i=1}^d P_i(z) .$$

$Y(z)$ é computado reduzindo-se primeiro os polinômios de entrada $X(z)$ e $H(z)$ módulo $P_i(z)$, isto é

$$X_i(z) \equiv X(z) \pmod{P_i(z)},$$

$$H_i(z) \equiv H(z) \pmod{P_i(z)} .$$

$Y(z)$ será obtido computando-se os d produtos polinomiais $X_i(z) H_i(z) \pmod{P_i(z)}$ e usando-se o teorema do resto chinês, como mostra a Figura 5.1, para N primo. A reconstrução $Y(z)$ dos produtos módulo $P_i(z)$ é feita por

$$Y(z) \equiv \sum_{i=1}^d S_i(z) X_i(z) H_i(z) \pmod{(z^N-1)},$$

onde, para cada valor u de i,

$$S_u(z) \equiv 0 \pmod{P_i(z)}, \quad i \neq u,$$

$$S_u(z) \equiv 1 \pmod{P_u(z)}$$

e

$$S_u(z) \equiv T_u(z)/R_u(z) \pmod{(z^N-1)}$$

$$\equiv \left(\prod_{\substack{i=1 \\ i \neq u}}^d P_i(z) \right) / \left(\prod_{\substack{i=1 \\ i \neq u}}^d P_i(z) \pmod{P_u(z)} \right) \pmod{(z^N-1)}. \quad (5.1)$$

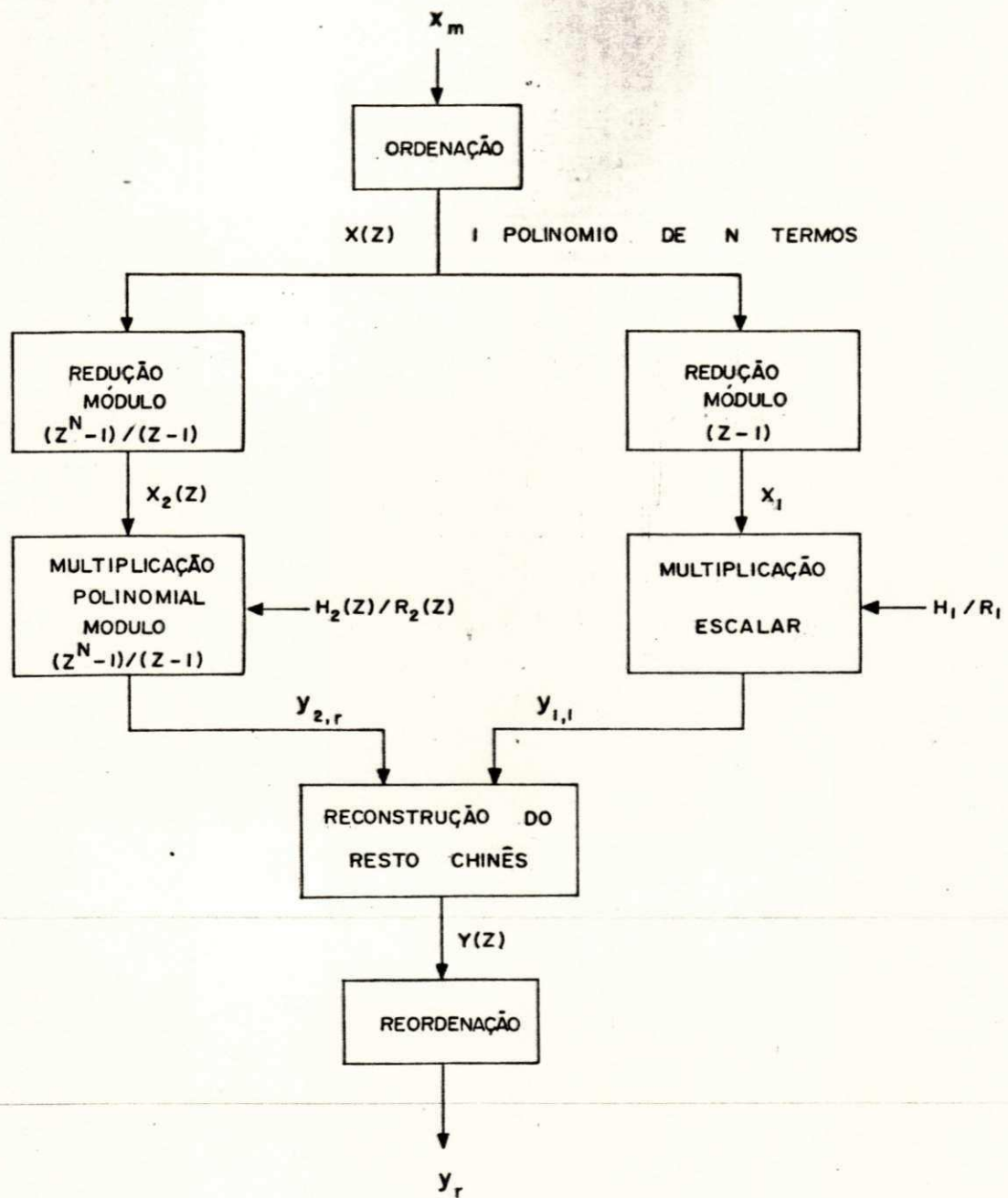


Figura 5.1 - Computação da convolução circular de comprimento N através do teorema do resto chinês. N primo.

Para N primo, d é igual a 2 e $z^N - 1$ é o produto de dois polinômios ciclotômicos $P_1(z) = z - 1$ e $P_2(z) = z^{N-1} + z^{N-2} + \dots + z + 1$. Isto significa que as reduções módulo $P_i(z)$ e a reconstrução de $Y(z)$ pelo teorema do resto chinês são obtidos sem multiplicações e com um número limitado de adições. X_1 e $X_2(z)$ são computados com $N - 1$ adições por

$$X_1 = \sum_{m=0}^{N-1} x_m \quad \text{e} \quad X_2(z) = \sum_{m=0}^{N-1} (x_m - x_{N-1}) z^m$$

De modo análogo, computamos H_1 e $H_2(z)$.

Da equação (5.1), podemos ver que $S_u(z) = T_u(z)/R_u(z)$, onde $1/R_u(z)$ é definido módulo $P_u(z)$. Assim a multiplicação por $1/R_u(z)$ pode ser combinada com a multiplicação $X_u(z)H_u(z)$ módulo $P_u(z)$. Além disso, em muitos casos práticos, uma das sequências de entrada h_n , é fixa. Esse algoritmo só funciona para essa situação, e, neste caso, $H_u(z)/R_u(z)$ pode ser precomputada, isto é, $0 \equiv H_u(z) \pmod{R_u(z)}$. Para N primo, $X_1 H_1 / R_1$ é um escalar $y_{1,1}$, enquanto que $X_2(z)H_2(z)/R_2(z) \pmod{P_2(z)}$ é um polinômio de $N - 1$ termos com os coeficientes de z^r dados por $y_{2,r}$. Assim, a reconstrução pelo resto chinês é obtida neste caso com $2(N-1)$ adições por

$$Y(z) = (y_{1,1} + y_{2,N-2}) z^{N-1} + y_{1,1} - y_{2,0} + \sum_{r=1}^{N-2} (y_{1,1} - y_{2,r} + y_{2,r-1}) z^r$$

É fácil de ver que o número de adições necessárias em cada redução módulo os vários polinômios ciclotômicos é o mesmo. Esse

resultado é geral e se aplica a qualquer convolução circular, quando uma das sequências de entrada for fixa [13].

5.2) - Computação da convolução através de algoritmos FFT

Sejam x_m e h_n as sequências de entrada, de N termos, com $N = 2^\delta$, $\delta = 1, 2, 3, \dots$ e sejam \bar{X}_k e \bar{H}_k as transformadas discretas de Fourier de x_m e h_n respectivamente.

A convolução linear, também pode ser obtida através da DFT, conforme subsecção 3.2.5.

5.2.1) - Sequências reais quaisquer

A convolução circular y_r das sequências x_m com h_n é dada pela transformada inversa discreta de Fourier do produto das transformadas discretas de Fourier de cada sequência [13]. Isto é, a convolução circular y_r de x_m com h_n é dada por

$$y_r = \text{DFT}^{-1} [\bar{X}_k \bar{H}_k]$$

onde o produto $\bar{X}_k \bar{H}_k$ é calculado termo a termo. Neste caso, a computação da convolução é feita com 3 aplicações de FFT, como mostra a Figura 5.2.

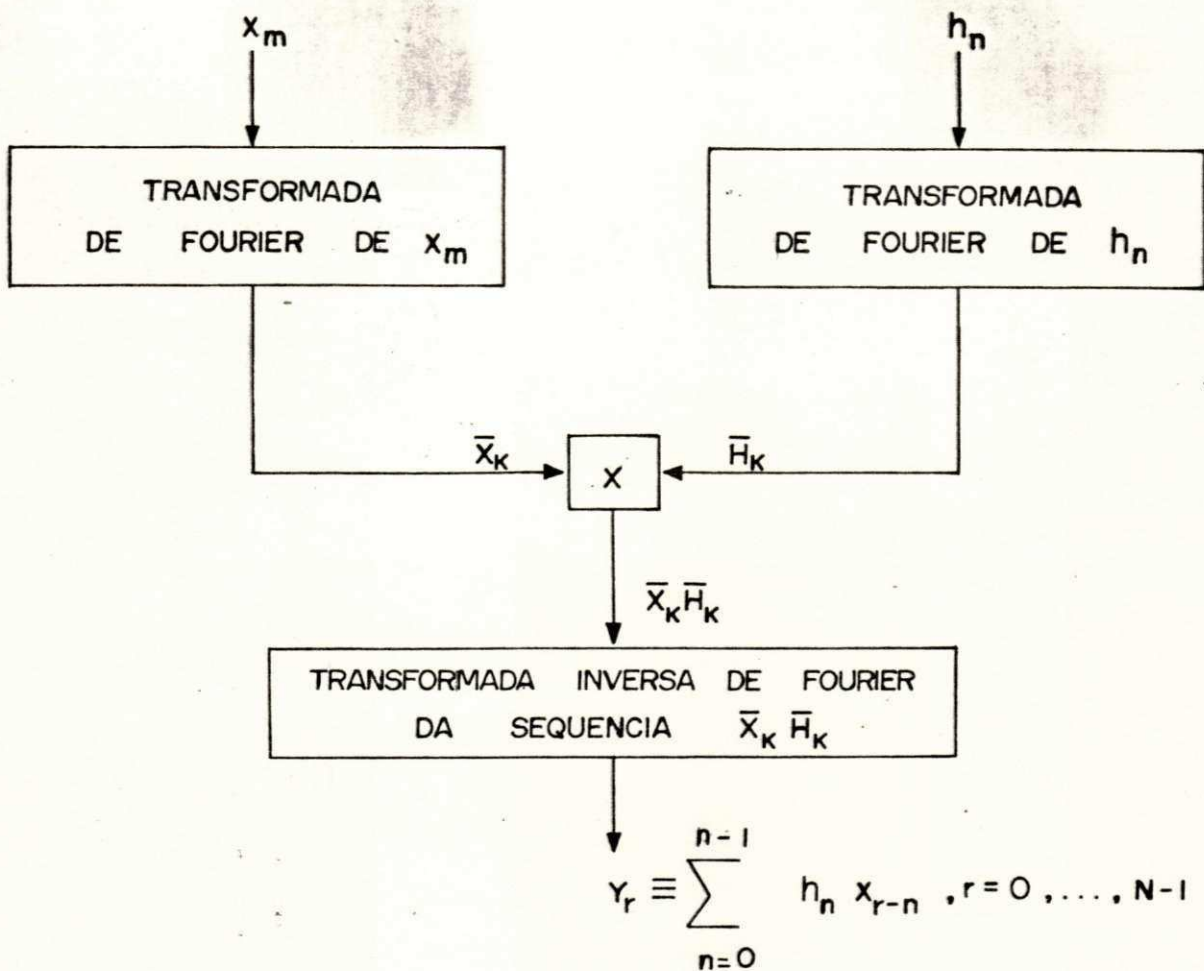


Figura 5.2 - Computação da convolução real em 3 passos de FFT.

5.2.2) - Sequências reais pares

Como vimos na secção 3.1.2, a DFT de uma sequência real par, é uma sequência real par. Portanto, para computarmos a convolução circular de duas sequências reais pares só necessitamos de duas aplicações de FFT. Para isso, formemos a sequência auxiliar

complexa $y_n = x_m + jh_n$. A DFT de y_n é dada por $\bar{Y}_k = \bar{X}_k + j\bar{H}_k$ onde \bar{X}_k é a DFT de x_m e \bar{H}_k é a DFT de h_n [13]. Para obtermos a convolução circular y_r de x_m com h_n , basta calcularmos a DFT inversa do produto $\bar{X}_k \bar{H}_k$.

5.3) - Computação da convolução através de transformadas de Mersenne

A computação da convolução circular y_r das sequências x_m e h_n de p pontos, utilizando transformadas de Mersenne de p pontos definida módulo $(2^P - 1)$, p primo, será obtida com 3 passos de transformadas de Mersenne como ilustra a Figura 5.3 .

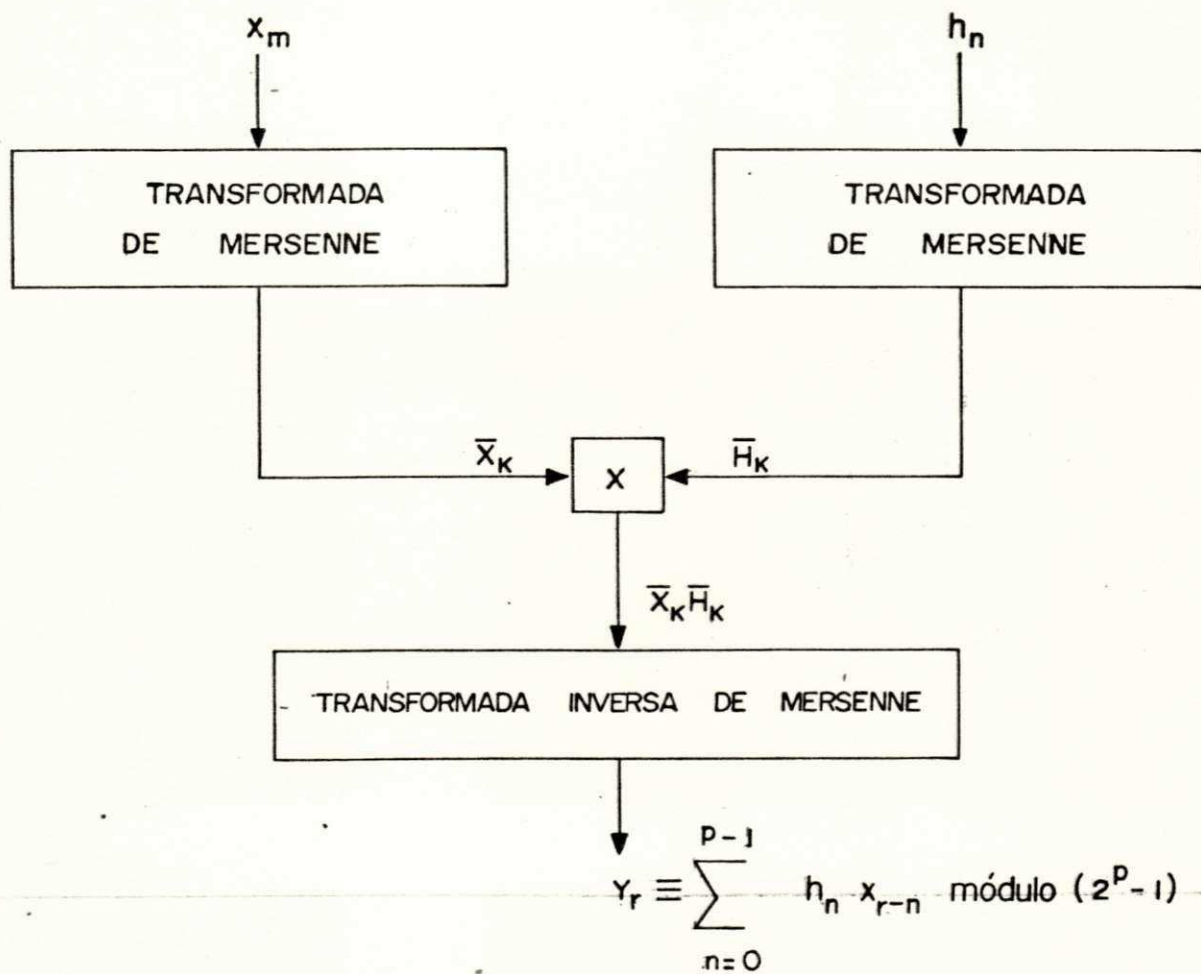


Figura 5.3 - Computação da convolução circular de comprimento p módulo $(2^p - 1)$ por transformadas de Mersenne.

Os números de Mersenne 2^p-1 , crescem muito rapidamente. Por exemplo, no IBM-4341 só é possível trabalharmos com $p = 31$. Para resolvermos esse problema, precisamos desenvolver uma aritmética específica para podermos operar com esses números. Mesmo sem desenvolvermos essa aritmética, apresentaremos o procedimento, a seguir, da computação da convolução circular através de transformadas de Mersenne.

5.3.1) - Computação da transformada de Mersenne \bar{x}_k de x_m .

$$\text{Definição: } \bar{x}_k \equiv \sum_{m=0}^{p-1} x_m 2^{mk} \pmod{(2^p-1)},$$

$$k = 0, 1, \dots, p-1. \quad (5.2)$$

A equação (5.2) pode ser reescrita na forma matricial

$$\begin{bmatrix} \bar{x}_0 \\ \bar{x}_1 \\ \bar{x}_2 \\ \bar{x}_3 \\ \vdots \\ \bar{x}_{p-1} \end{bmatrix} = \begin{bmatrix} 2^0 & 2^0 & 2^0 & \dots & 2^0 \\ 2^0 & 2 & 2^2 & \dots & 2^{p-1} \\ 2^0 & 2^2 & 2^4 & \dots & 2^{2(p-1)} \\ 2^0 & 2^3 & 2^6 & \dots & 2^{3(p-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 2^0 & 2^k & 2^{2k} & \dots & 2^{(p-1)^2} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{p-1} \end{bmatrix} \quad (5.3)$$

onde cada \bar{x}_i é definido módulo (2^p-1) .

Qualquer número do tipo 2^{mk} é definido módulo (2^p-1) se e somente se mk for definido módulo p [13]. Logo, (5.3) pode ser reescrita na forma

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{p-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ & 1 & & & \vdots \\ & & 1 & & \vdots \\ & & & 1 & \vdots \\ & & & & \vdots \\ & & & & 2^{mk \bmod p} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{p-1} \end{bmatrix} \quad (5.4)$$

Se chamarmos de A a primeira matriz do lado direito de (5.4), então na implementação, A não precisa ser construída explicitamente conforme o trecho de programa seguinte:

```

for I = 1 to p
  begin
    k:=I-1
    for J = 1 to p
      begin
        m:=J-1
        A(I,J):=2mk mod p
      end
    end
  end
end

```

(5.5)

Matematicamente a transformada de Mersenne \bar{x}_k de x_m será obtida resolvendo-se a equação matricial (5.4) e computacionalmente levando-se em consideração (5.5).

5.3.2) - Computação da transformada de Mersenne \bar{H}_k de h_n .

\bar{H}_k é obtida de modo análogo a \bar{X}_k .

5.3.3) - Computação da transformada inversa de Mersenne do produto $\bar{X}_k \bar{H}_k$

$$\text{Definição: } y_r \equiv p^{-1} \sum_{k=0}^{p-1} \bar{X}_k \bar{H}_k 2^{-rk} \pmod{(2^p-1)},$$

$$r = 0, \dots, p-1 \quad (5.6)$$

onde: $p^{-1} = 2^{p-1} - (2^p-2)/p \pmod{(2^p-1)} = \text{INV}P,$

$$2^{-1} = 2^{p-1} \pmod{(2^p-1)} = \text{INV}2 \text{ e}$$

y_r é a convolução circular de x_m com h_n [13].

Portanto, podemos reescrever (5.6) da seguinte forma:

$$y_r \equiv \sum_{k=0}^{p-1} (\text{INV}P) \bar{X}_k \bar{H}_k (\text{INV}2)^{rk} \pmod{(2^p-1)},$$

$r = 0, \dots, p-1$; ou ainda:

$$y_r \equiv \sum_{k=0}^{p-1} t_k (\text{INV}2)^{rk} \pmod{(2^p-1)},$$

$r = 0, \dots, N-1, \quad (5.7)$

onde $t_k = (\text{INV}P)\bar{X}_k\bar{H}_k$ e, $\bar{X}_k\bar{H}_k$ é um produto termo a termo.

A computação de (5.7) é semelhante à computação de (5.2). Assim, a convolução circular de comprimento p é computada com três transformadas de Mersenne mais p multiplicações conforme a Figura 5.3. Porém, quando uma das seqüências de entrada é fixa, por exemplo h_n , então sua transformada \bar{H}_k pode ser precalculada e combinada com multiplicações por p^{-1} correspondente a transformada inversa. Neste caso, apenas duas transformadas de Mersenne precisam ser avaliadas.

5.4) - Diferença entre a convolução linear e a convolução circular

Consideremos duas seqüências x_m e h_n de N termos com h_n fixa e, suponhamos $N = 4$. A figura 5.4 mostra como são obtidos os elementos da convolução linear y_r de x_m com h_n . Os elementos diferentes de zero de y_r são obtidos fazendo coincidir inicialmente x_0 com h_0 e a seguir, desloca-se a seqüência x_m sobre a seqüência h_n , de um em um termo, até coincidir x_{m-1} com h_{n-1} . Por outro lado, para se obter os elementos diferentes de zero da convolução circular y_r de x_m com h_n , constroi-se inicialmente a seqüência $a_m = (x_{m-1}, \dots, x_0, x_{m-1}, \dots, x_1)$ e a seguir, a obtenção dos elementos da convolução circular é idêntica à obtenção dos elementos da convolução linear, conforme a figura 5.5.

5.4.1) - Computação da convolução linear y_r de x_m com h_n .

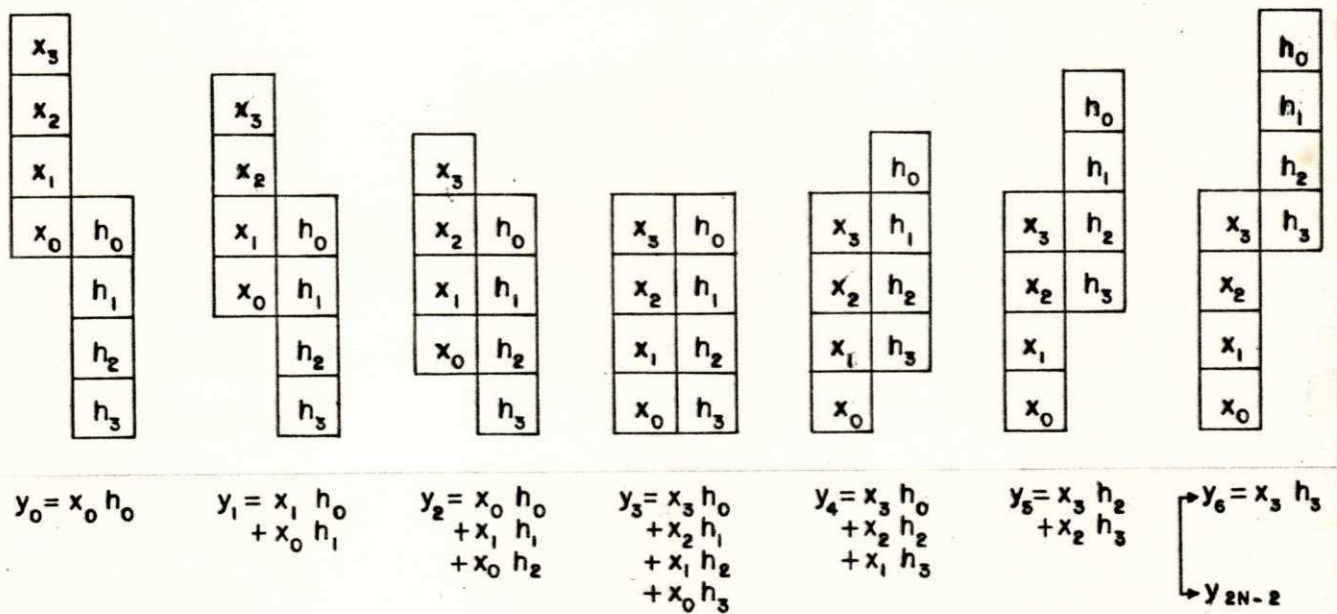


Figura 5.4 - Ilustração da computação da convolução linear.

5.4.2) - Computação da convolução circular y_l de x_m com h_n .

x_3	
x_2	
x_1	
x_0	h_0
x_3	h_1
x_2	h_2
x_1	h_3

$$y_0 = x_0 h_0 + x_3 h_1 + x_2 h_2 + x_1 h_3$$

x_3	
x_2	
x_1	h_0
x_0	h_1
x_3	h_2
x_2	h_3
x_1	

$$y_1 = x_1 h_0 + x_0 h_1 + x_3 h_2 + x_2 h_3$$

x_3	
x_2	h_0
x_1	h_1
x_0	h_2
x_3	h_3
x_2	
x_1	

$$y_2 = x_2 h_0 + x_1 h_1 + x_0 h_2 + x_3 h_3$$

x_3	h_0
x_2	h_1
x_1	h_2
x_0	h_3
x_3	
x_2	
x_1	

$$y_3 = x_3 h_0 + x_2 h_1 + x_1 h_2 + x_0 h_3$$

$\rightarrow y_{N-1}$

Figura 5.5 - Ilustração da computação da convolução circular.

5.5) - Produtos de Polinômios

O problema de computar o produto de dois polinômios de uma variável, é exatamente o mesmo de computar a convolução linear de duas seqüências [1]. Isto é,

$$\left(\sum_{n=0}^{N-1} a_n x^n \right) \left(\sum_{m=0}^{N-1} b_m x^m \right) = \sum_{k=0}^{2N-2} c_k x^k ,$$

onde $c_k = \sum_{m=0}^{N-1} a_m b_{k-m}$.

CAPITULO VI

IMPLEMENTAÇÕES

Nesta secção são relacionados os algoritmos implementados e serão descritos alguns detalhes de implementação de dois algoritmos em linguagem FORTRAN. Uma visão macro desses algoritmos é dada pelas figuras inseridas nos capitulos 4 e 5. No entanto, procuramos mostrar alguns detalhes de implementação dos algoritmos FFT de Cooley-Tukey e Bruun.

No trabalho foram implementados:

- 1) Computação da convolução circular de N termos, N primo, usando o teorema do resto chinês.
- 2) Computação da convolução circular de N termos, N primo, usando as transformadas de Mersenne.
- 3) O algoritmo FFT de Cooley-Tukey.

4) Computação da convolução circular usando algoritmos FFT.

4.1) - Com 3 passos de FFT - sequências reais quaisquer.

4.2) - Com 2 passos de FFT - sequências reais pares.

5) O algoritmo de Bruun.

A seguir daremos alguns detalhes de implementação dos algoritmos FFT de Cooley-Tukey e Bruun.

Implementação

6.1) O Algoritmo FFT de Cooley-Tukey.

Seja $x_m = XR + jXI$ uma sequência complexa de N termos e seja $X(n) = XR + jXI$ a transformada discreta de Fourier de x_m onde $N = 2^\delta$, $\delta = 1, 2, 3, 4, \dots$

Algoritmo: Computa transformada discreta de Fourier $X(n)$ de N pontos

$$X(n) = \sum_{k=0}^{N-1} x_0(k) W^{nk} ; n = 0, \dots, N-1;$$

$$k = 0, \dots, N-1; W = e^{-j2\pi/N} \text{ e } j = \sqrt{-1} , \quad (6.1)$$

de uma sequência x_m de números reais ou complexos, utilizando fatoração de matriz conforme secção 4.1.

6.1.1) - Obtenção dos elementos de $X(n)$ - Equação (4.4)

A fatoração da matriz $N \times N$ em δ matrizes $N \times N$ e consequentemente, a solução das δ equações matriciais, é feita conforme o trecho do programa abaixo:

```

 $\delta 1 := \delta - 1$ 
 $N2 := N/2$ 
 $K := 0$ 
for L = 1 to  $\delta$ 
  begin
    while(K < N) do
      begin
        for I = 1 to N2
          begin
            M:=Valor inteiro de  $(K/2^{\delta 1})$ 
            P:=IBR(M)
            T:=W x(K + N2)
            x(K + N2):= x(K) - T
            x(K):=x(K) + T
            K:=K + 1
          end
          K:=K + N2
        end
        K:=0
         $\delta 1 := \delta 1 - 1$ 
        N2:=N2/2
      end
    end
  end

```

e, observando-se que,

$$x_r(k) = x_{r-1}(k) + W^p x_{r-1}(k + N/2^r) \quad (6.2)$$

e

$$x_r(k + N/2^r) = x_{r-1}(k) - W^p x_{r-1}(k + N/2^r),$$

pois $W^p = W^{p+N/2}$.

O sistema de equações:

$$x_1(0) = x_0(0) + W^0 x_0(2)$$

$$x_1(1) = x_0(1) + W^0 x_0(3)$$

$$x_1(2) = x_0(0) + W^2 x_0(2)$$

$$x_1(3) = x_0(1) + W^2 x_0(3)$$

é o sistema de equações (6.2) com $N = 4$, $r = 1$, $k = 0$ ou $k = 1$, $N/2 = 2$ e $N/2^r = 2$. Enquanto que, o sistema de equações:

$$x_2(0) = x_1(0) + W^0 x_1(1)$$

$$x_2(1) = x_1(0) + W^2 x_1(1)$$

$$x_2(2) = x_1(2) + W^1 x_1(3)$$

$$x_2(3) = x_1(2) + W^3 x_1(3)$$

é o sistema de equações (6.2) com $N = 4$, $r = 2$, $k = 0$ ou $k = 2$, $N/2 = 2$ e $N/2^r = 1$.

6.1.2) - Reordenamento dos elementos da DFT $X(n)$

E feito de acordo com o trecho de programa seguinte:

```
for K = 1 to N
  begin
    if (IBR(K)) > K then
      begin
        R:=x(K)
        x(K):=x(IBR(K))
        x(IBR(K)):=R
      end
    end
  end
```

6.1.3) - Obtenção dos elementos da DFT inversa, quando for o caso

Basta chamar a rotina com $X(n)$ no lugar de $x_0(k)$ e $-k$ no lugar de k na equação (6.1) e, dividir cada elemento por N .

6.1.4) - Função IBR

A função IBR dada pelo trecho de programa seguinte:

```
for I = 1 to
  begin
    J:=M/2
    IBR:=2*IBR+(M-2*J)
    M:=J
  end
```

é utilizada para determinar o valor de p na equação (6.2) e também é utilizada no reordenamento da DFT em 6.1.2, onde $Y(n)$ é mudado para $X(n)$ conforme a equação (4.10).

6.2) - O Algoritmo de Bruun

Consideremos uma sequência real x de N termos, $N = 2^\delta$, $\delta = 4, 5, 6, 7, \dots$, e seja $\bar{X}_k = XR + jXI$ a transformada discreta de Fourier de x_m .

Algoritmo: Computa transformada discreta de Fourier \bar{X}_k de N pontos

$$\bar{X}_k = \sum_{m=0}^{N-1} x_m W^{mk}, \quad k = 0, \dots, N-1;$$

$$W = e^{-j2\pi/N} \quad e \quad j = \sqrt{-1}, \quad (6.3)$$

de uma sequência x_m de números reais, utilizando restos de divisão polinomial conforme secção 4.5.

6.2.1) - Obtenção do vetor de raízes (VR)

Na figura 6.3, cada retângulo contém um polinômio da forma $z^{4q} + az^{2q} + 1$, onde a é um número real. VR conterá os valores de a de todos os retângulos, onde cada segmento i de VR está associado ao nível i da árvore gerada pelo algoritmo. O primeiro e o segundo segmentos de VR são obtidos em separado, conforme figura 6.1. O terceiro segmento de VR é obtido do segundo segmento de VR da seguinte forma: a primeira metade é constituída pelos elementos do segundo segmento e a segunda metade do terceiro segmento é obtida da segunda metade do segundo segmento,

observando-se a indicação das setas conforme a figura 6.1, ou seja: cada elemento a da segunda metade do segundo segmento, dá origem a dois elementos b e $-b$ da segunda metade do terceiro segmento onde b é obtido como segue, $b = \sqrt{2-a}$. De modo análogo obtêm-se o quarto segmento de VR a partir do terceiro segmento e assim sucessivamente. A implementação de VR foi feita de acordo com as explicações acima.

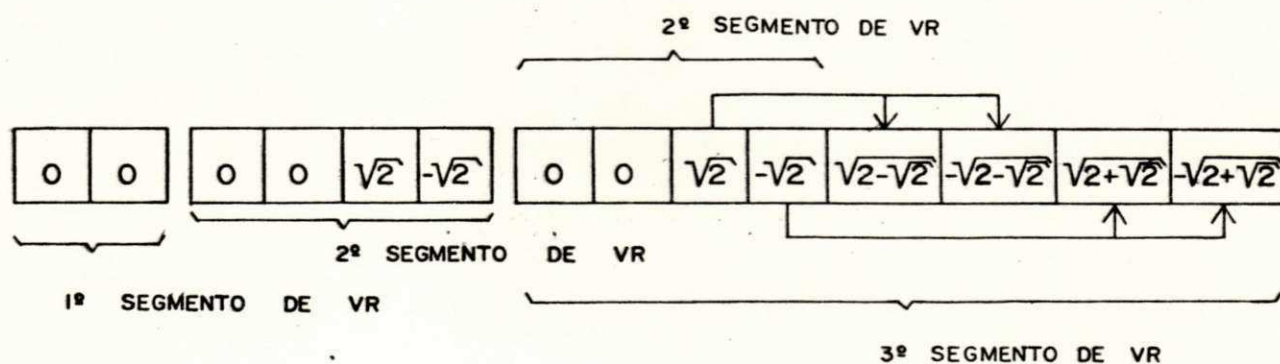


Figura 6.1 - Situação do vetor de raízes (VR), para $N = 16$ - Algoritmo de Bruun.

E importante observar o processo de dicotomia existente no algoritmo de Bruun.

Não foi possível localizar nenhum tipo de referência bibliográfica, que pudesse ajudar nessa implementação e por isso, foi necessário muitas horas de trabalho e testes com programas de computador para que pudessemos concluir integralmente a implementação do algoritmo de Bruun.

6.2.2) - Obtenção da matriz de apontadores (para VR)

A figura 6.2 ilustra a situação inicial para o caso em que $N = 32$, da matriz de apontadores IAP, que tem como objetivo localizar os elementos de VR. A primeira coluna de IAP, contém o início de cada segmento de VR, em ordem decrescente, enquanto que, a segunda coluna de IAP, fornece a quantidade de posições utilizadas em cada segmento de VR. Assim as posições de VR, em cada segmento a serem utilizadas, são obtidas somando-se os conteúdos das colunas 1 e 2 da matriz de apontadores IAP.

15	0
7	0
3	0
1	0

Figura 6.2 - Situação inicial da matriz de apontadores IAP, para $N = 32$ - Algoritmo de Bruun.

6.2.3) - Caminha, empilha, obtém valores finais e desempilha

6.2.3.1) - Caminha: Como o algoritmo de Bruun possui uma estrutura de árvore binária completa, esse trecho do algoritmo nos permite caminhar conveniente e eficientemente na árvore gerada pelo algoritmo. Isto significa calcular os coeficientes dos restos intermediários obtidos nas divisões polinomiais e guardá-los em um vetor até que se possa obter os elementos correspondentes da DFT. Para ilustrar essa situação, escolhemos o caso em que $N = 8$. As dificuldades aumentam a medida que N cresce. Para se obter a generalização do procedimento, foi necessário desenvolver o algoritmo para $N = 4, 8, 16, 32, 64$ e 128 . Além disso, notou-se que era necessário fazer o caminhamento em duas etapas. Na primeira, obtém-se todos os elementos pares da DFT e na segunda, obtém-se todos os elementos ímpares da DFT. O controle do desempilhamento dos restos intermediários é feito através do parâmetro "m" conforme a subrotina "DPILHA" do programa de computador "ZEBRUUN", que se encontra na biblioteca de subprogramas FORTRAN do NPD-UFPb.

A figura 6.3 ilustra o procedimento de como obter os restos intermediários e conseqüentemente os valores correspondentes dos elementos da DFT. O procedimento é o seguinte:

- 1) obtém-se e guardam-se R_1 e R_{11} .
- 2) com R_{11} obtém-se \bar{X}_0 e \bar{X}_4 ,
- 3) volta-se a R_1 e obtém-se R_{12} e, conseqüentemente \bar{X}_6 e \bar{X}_2 ,
- 4) volta-se ao início x_m . Obtém-se e guardam-se R_2 e R_{21} ,
- 5) com R_{21} obtém-se \bar{X}_5 e \bar{X}_3 ,
- 6) volta-se a R_2 e obtém-se R_{22} e, conseqüentemente \bar{X}_7 e \bar{X}_1 .

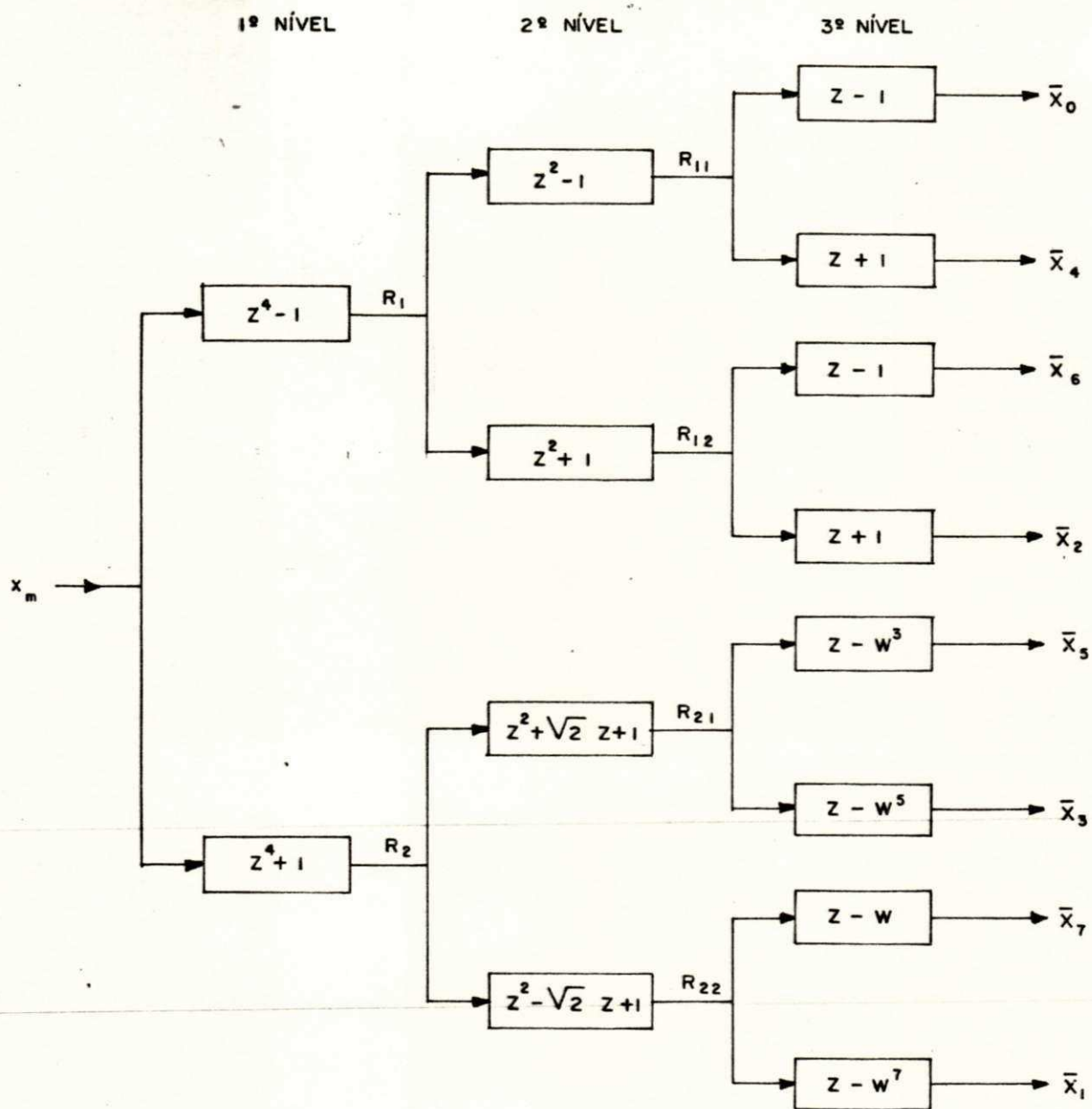


Figura 6.3 - Diagrama do algoritmo de Bruun para $N = 8$.

6.2.3.2) - Empilha: Obtém restos intermediários e armazena-os em uma pilha (vetor de trabalho).

6.2.3.3) - Obtém valores finais: Obtém os elementos da transformada discreta de Fourier \bar{X}_k .

6.2.3.4) - Desempilha: Quando obtemos um par de valores finais, um determinado resto intermediário deve ser eliminado da pilha (vetor de trabalho). Aqui, selecionamos o próximo resto, para reiniciar o caminharmento na árvore.

6.2.4) - Rotina rápida de divisão: Restos de divisão de polinômios, no algoritmo de Bruun.

Consideremos a equação polinomial

$$D(x) = d(x)Q(x) + r(x). \quad (6.4)$$

Os coeficientes de $r(x)$ em (6.4) podem ser obtidos rapidamente explorando a forma especial de $d(x)$ no caso do algoritmo de Bruun. Para $N = 8$ tem-se:

$$D(x) = d_7x^7 + d_6x^6 + d_5x^5 + d_4x^4 + d_3x^3 + d_2x^2 + d_1x + d_0 .$$

$$d(x) = x^4 + ax^2 + 1, \text{ onde } a \text{ é um dos elementos de VR.}$$

$$Q(x) = q_3x^3 + q_2x^2 + q_1x + q_0 .$$

$$r(x) = r_3x^3 + r_2x^2 + r_1x + r_0 .$$

Nosso objetivo é determinar os coeficientes de $r(x)$ no caso em que $a \neq 0$ pois, se $a = 0$ existe uma maneira de calcular rapidamente os coeficientes de $r(x)$ [13].

Resolvendo a equação polinomial (6.2) obtêm-se:

$$r_3 = (d_3 - ad_5) + (a^2 - 1)d_7$$

$$r_2 = (d_2 - ad_4) + (a^2 - 1)d_6$$

$$r_1 = (d_1 - d_5) + ad_7$$

$$r_0 = (d_0 - d_4) + ad_6 .$$

(6.5)

A generalização da rotina com $a \neq 0$, só foi possível mediante os resultados obtidos para $N = 16$ e para $N = 32$ e, isso levou muito tempo em virtude dos graus dos polinômios envolvidos tornarem-se muito elevados. A partir daí, observou-se então que os coeficientes de $r(x)$ seriam obtidos em duas etapas, conforme (6.5). Na primeira etapa, obtêm-se a primeira metade dos coeficientes de $r(x)$ e na segunda etapa, obtêm-se a segunda metade dos coeficientes de $r(x)$. É bom lembrar que os coeficientes de $r(x)$ estão em ordem decrescente das potências de x . Tem-se, assim:

$$r(x) = \sum_{i=0}^{N/4-1} (d_i - d_{i+N/2} + ad_{i+N/2+N/4})x^i + \sum_{k=N/4}^{N/4-1} (d_k - ad_{k+N/4} + (a^2-1)d_{N/2+N/4})x^k.$$

Se considerarmos os polinômios $D(x)$, $d(x)$ e $Q(x)$ escritos em potências decrescentes de x , os coeficientes de $r(x)$ serão modificados. Essa modificação foi necessária na implementação do algoritmo, haja vista os casos em que $d(x) = x^{N/2} - 1$ ou $d(x) = x^{N/2} + 1$ [13].

Observe na figura 6.3 que $d(x)$ é sempre o polinômio contido em cada retângulo.

6.2.5) - Obtenção da DFT \bar{X}_k de x_m

Na subsecção 6.2.3, nós determinamos os valores dos elementos da sequência \bar{X}_k , só que esses valores aparecem desordenados. Para reordenar esses valores na ordem correta, foi necessário gerar os índices dos elementos da sequência \bar{X}_k , de acordo com o diagrama apresentado na figura 6.3. Esses índices foram gerados inicialmente para $N = 4, 8, 16$ e 32 . A seguir, foi feita a generalização para qualquer N . A figura 6.4, ilustra o procedimento para obtermos os índices dos elementos da sequência \bar{X}_k de N termos, a partir dos índices dos elementos da sequência \bar{X}_k de $N/2$ termos.

A generalização da obtenção desses índices só foi possível após desenvolvermos na íntegra o algoritmo, para as quatro situações acima. A partir daí, observou-se que os índices da DFT de 8 pontos poderiam ser obtidos dos índices da DFT de 4 pontos. De modo análogo, os índices da DFT de 16 pontos poderiam ser obtidos dos índices da DFT de 8 pontos e também, os índices da DFT de 32 pontos poderiam ser obtidos dos índices da DFT de 16 pontos. Com isso, verificou-se que se tivermos os índices de uma DFT de $N/2$ pontos, podemos obter os índices da DFT de N pontos da seguinte forma: o primeiro índice é sempre 0 e o segundo é sempre $N/2$, o último índice é sempre 1 e o penúltimo é sempre $N-1$. A partir daí, os demais índices da DFT de N pontos são obtidos conforme a figura 6.4, observando-se que exceto o primeiro e o último índices da DFT de $N/2$ pontos, cada um dos demais índices i da DFT de $N/2$ pontos gera dois índices i e k na DFT de N pontos.

Um desses índices i é igual ao índice da DFT de $N/2$ pontos e outro índice k satisfaz a relação $k = N-i$. Como vemos, os índices da DFT de N pontos são sempre gerados de dois em dois. Como não havia dúvidas de como obter os índices dos elementos das DFTs de 8, 16 e 32 pontos a partir dos índices dos elementos da DFT de 4 pontos, utilizamos a partir daí o computador, onde foram feitas centenas de testes com um programa computando a DFT diretamente pela definição (6.3) e outro o programa, computando a mesma DFT pelo algoritmo de Bruun. Muito tempo foi necessário para comparar tais resultados.

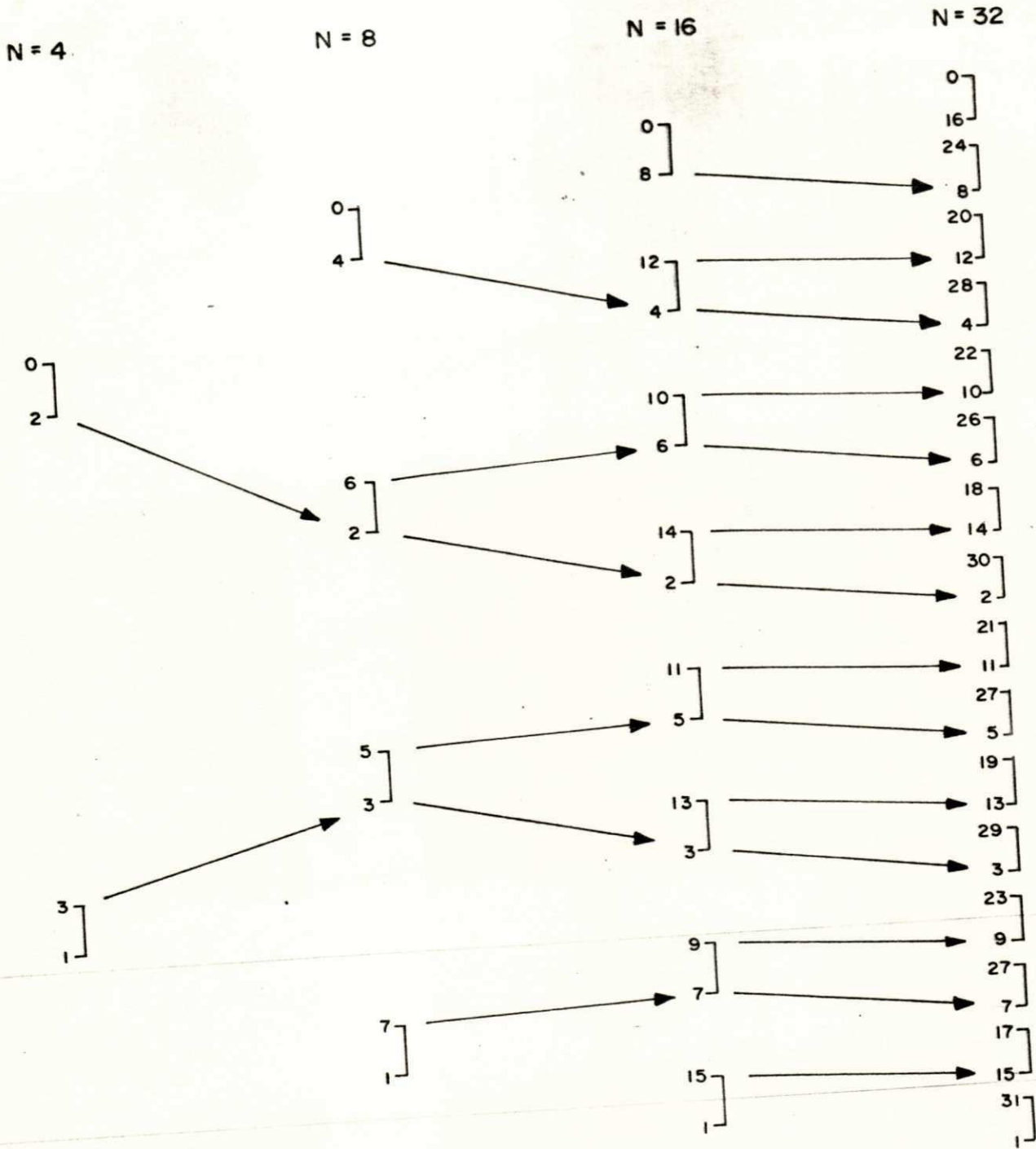


Figura 6.4 - Obtenção dos índices dos elementos de uma DFT de N pontos a partir dos índices dos elementos de uma DFT de $N/2$ pontos - Algoritmo de Bruun.

CAPITULO VII

RESULTADOS NUMERICOS, CONCLUSOES E SUGESTOES

7.1) - Resultados

Além da implementação do algoritmo de Cooley-Tukey [2], mencionada na secção 6.1, utilizamos o mesmo algoritmo com outra implementação [8], bem diferente desta. De modo que foi possível comparar o tempo de execução do algoritmo de Bruun com as duas versões do algoritmo de Cooley-Tukey. A tabela 7.1 relaciona o tempo de execução, em segundos, dos três algoritmos, executados em WATFIV num computador IBM-4341 com o sistema operacional OS/VS1 num ambiente VM para valores de N entre 16 e 1024. Para cada valor de N os programas foram executados quatro vezes e em seguida, feita a média aritmética desses quatro tempos. Para se obter tais tempos, eliminaram-se a entrada e a saída de dados. Os dados de entrada foram gerados aleatoriamente e colocados em uma declaração DATA e por isso, infelizmente, só foi possível

executar os programas em WATFIV até $N = 1024$. Para $N = 2048$ usou-se o mesmo procedimento acima, porém, os programas foram compilados e executados em FORTRAN VS, também no IBM-4341 e, verificou-se que o tempo de execução neste caso é insignificante (da ordem de centésimos de segundos) para qualquer um dos algoritmos. Adotando a seguinte convenção:

TBRUUN - Tempo do algoritmo de Bruun,

TCTUKEY1 - Tempo do algoritmo de Cooley-Tukey [2],

TCTUKEY2 - Tempo do algoritmo de Cooley-Tukey [8],

a tabela 7.1 exibe os tempos médios obtidos em quatro execuções para todos os algoritmos.

Tabela 7.1 - Tempo de execução, em segundos, dos algoritmos de Bruun e Cooley-Tukey (média de 4 execuções para cada N , com sequências reais).

N	TBRUUN	TCTUKEY1	TCTUKEY2
16	0.175	0.205	0.222
32	0.267	0.272	0.292
64	0.467	0.552	0.455
128	0.810	1.142	0.837
256	1.677	2.507	1.707
512	3.480	5.707	3.522
1024	7.060	12.882	7.690

Para verificar se de fato o tempo de computação (execução) em qualquer um dos algoritmos FFT implementados é realmente

proporcional a $N \log_2 N$, foram usados os tempos obtidos em todas as execuções (quatro vezes para cada N) numa regressão linear do tempo T em função de $N \log_2 N$, isto é: $T = \alpha N \log_2 N + \beta + \text{erro}$, para cada um dos algoritmos. Constatou-se que em qualquer uma das situações, o coeficiente de determinação R^2 é maior que 80% e, isto garante, de acordo com a metodologia utilizada na regressão linear, que a proporcionalidade é boa.

A tabela 7.2 mostra a estatística referente aos algoritmos de Bruun (BRUUN), de Cooley-Tukey (CTUKEY1) primeira versão [2] e (CTUKEY2) segunda versão [8].

Tabela 7.2 - Resultados da regressão linear do tempo em função de $N \log_2 N$.

ALGORITMO	α	β	R^2
BRUUN	0.0017	0.0012	99,8%
CTUKEY1	0.0031	-0.3316	99,5%
CTUKEY2	0.0018	-0.0373	99,6%

Comentários sobre os resultados relacionados na tabela 7.2.

BRUUN : Só aritmética real mas tem overhead das divisões polinomiais (cálculo do resto).

CTUKEY1 : Opera com parte real e imaginária mas não tem cálculo de resto de divisão de polinômios. A implementação deve ter sido muito direta. Embora obedeça $T = \alpha N \log_2 N + \beta$ + erro os tempos são substancialmente maiores conforme tabela 7.1.

CTUKEY2 : Opera com parte real e imaginária mas não tem cálculo de restos de divisão de polinômios, o que pode explicar a taxa de crescimento do tempo de execução com $N \log_2 N$ quase igual ao do algoritmo de Bruun.

7.2) - Conclusões

Os algoritmos relacionados nas secções 6.1 e 6.2 mostram que existem pelo menos duas maneiras diferentes de computar a convolução circular de N termos, com N primo, produzindo o mesmo resultado. Além disso, o algoritmo mencionado na subsecção 6.4.1 mostra que a convolução circular de N termos com $N = 2^\delta$, $\delta = 1, 2, 3, \dots$ pode ser computada utilizando-se qualquer um dos FFTs mencionados nas secções 4.1 a 4.5. Isso demonstra que diferentes métodos computacionais podem resolver eficientemente o mesmo problema matemático. Por outro lado, as aplicações do teorema do resto chinês e do algoritmo de Bruun mostram a grande importância da álgebra (aritmética residual), na resolução de problemas ligados à Engenharia Elétrica.

Com relação a parte de implementação, podemos observar que nem tudo que matematicamente é simples será simples de implementar. Um exemplo disso, é a implementação do algoritmo de Bruun. Por outro lado, a maneira de implementar um mesmo algoritmo tem influência na eficiência da rotina computacional (v. comparação das taxas de variação do tempo com $N \log_2 N$ nas duas implementações do algoritmo de Cooley-Tukey).

7.3) - Sugestões

Sugerimos como assunto para trabalhos adicionais,

- i) O estudo completo, aplicações e implementações dos algoritmos mencionados na secção 4.6 .
- ii) Implementar a convolução circular de p pontos, p primo, através de transformadas de Mersenne, para qualquer p .
- iii) Implementar a convolução circular de N termos através do teorema do resto chinês para $N = N_1 \times N_2$, com N_1 e N_2 mutuamente primos.
- iv) Estudar uma implementação eficiente da transformada de Mersenne (a nossa foi uma implementação direta, com número de multiplicações proporcional a p^2).

REFERENCIAS BIBLIOGRAFICAS

- [1] AHO, A.V., HOPCROFT, J.E. and ULLMAN, J.D.; The Design and Analysis of Computer Algorithms, Second Edition, Addison-Wesley Publishing Company, Massachusetts, 1975.
- [2] BRIGHAM, E.O.; The Fast Fourier Transform, Prentice-Hall, INC, Englewood Cliffs, New Jersey, 1974
- [3] CARLSON, A.B.; Sistemas de Comunicação, Editora da Universidade de São Paulo/Editora McGraw-Hill do Brasil, São Paulo, 1981.
- [4] COOLEY, J.W. and TUKEY, J.W.; An Algorithm for Machine Computation of Complex Fourier Series, Math. Comp. 19(1965), pp. 297-301.
- [5] DEAN, R.A.; Elementos de Álgebra Abstrata, Livros Técnicos e Científicos S.A., Rio de Janeiro, 1974.
- [6] ELLIOTT, D.F. and RAO, K.R.; Fast Transforms Algorithms, Analyses and Applications, Academic Press, INC, New York, 1982.
- [7] FIGUEIREDO, D.J.; Análise de Fourier e Equações Diferenciais Parciais, Editora Edgard Blucher Ltda, São Paulo, 1977.
- [8] FORMAN, M.L.; Fast Fourier Transform Technique and Application to Fourier Spectroscopy, J. Optical Soc. of America 56(1966), pp. 972-979.
- [9] FRALEIGH, J.B.; A First Course in Abstract Algebra, Addison-Wesley Publishing Company, Massachusetts, 1968.
- [10] HERNSTEIN, I.N.; Tópicos de Álgebra, Editora Universidade de São Paulo/Editora Polígono, São Paulo, 1970.

- [11] LIPSON, J.D.; Elements of Algebra and Algebraic Computing, Addison-Wesley Publishing Company, Massachusetts, 1981.
- [12] MONTEIRO, L.H.J.; Elementos de Algebra, Livros Técnicos e Científicos Editora S.A., São Paulo, 1978.
- [13] NUSSBAUMER, H.W.; Fast Fourier Transform and Convolution Algorithms, Second Edition, Springer Verlag, Berlin, 1982.
- [14] OPPENHEIM, A.V. and SCHAFER, R.W.; Digital Signal Processing, Prentice-Hall, INC, New Jersey, 1975.
- [15] RICHTMYER, R.D. and MORTON, K.W.; Difference Methods for Initial Value Problems, Wiley Interscience, New York, 1967.