APS
Implementação Computacionalmente Eficiente da DFT

Welliton Jhonathan Leal Babinski

Universidade Tecnológica Federal do Paraná - Campus Pato Branco
Programa de Pós-Graduação em Engenharia Elétrica
Processamento de Sinais
Prof. Dr. Rafael Cardoso

27 de Agosto de 2020

Resumo

Este trabalho se trata de uma pesquisa sobre a técnica para a implementação computacional eficiente da DFT baseada na decimação no tempo, bem como suas formalizações matemáticas, complexidade dos algoritmos e suas implementações em código via *software* Matlab.

1 Introdução

A transformada de Fourier discreta (DFT) desempenha um papel importante na análise, projeto e implementação de algoritmos e sistemas de processamento de sinal em tempo discreto, porque as propriedades básicas da transformada de Fourier de tempo discreto e ransformada de Fourier discreta tornam particularmente conveniente analisar e projetar sistemas no domínio de Fourier. É igualmente importante que existam algoritmos eficientes para calcular explicitamente o DFT. Como resultado, o DFT é um componente importante em muitas aplicações práticas de sistemas de tempo discreto.(OPPENHEIM; SCHAFER, 2010) Uma das razões pelas quais a análise de Fourier é de grande importância no processamento digital de sinais é a existência de algoritmos eficientes para calcular a transformada discreta de Fourier. (OPPENHEIM; SCHAFER, 1975)

Em termos de multiplicações e adições, a classe de algoritmos FFT pode ser ordens de magnitude mais eficientes do que algoritmos de compilação. A eficiência dos algoritmos FFT é tão alta, de fato, que em muitos casos o procedimento mais eficiente para implementar uma convolução é calcular a transformada das sequências a serem convolvidas, multiplicar suas transformadas e, em seguida, calcular a transformada inversa do produto das transformadas. (OPPENHEIM; SCHAFER, 2010)

As transformadas de menor comprimento podem ser avaliadas por métodos diretos ou podem ser posteriormente decompostas em transformações ainda menores. A maneira como esse princípio é implementado leva a uma variedade de algoritmos diferentes, todos com melhorias comparáveis na velocidade computacional. (OPPENHEIM; SCHAFER, 2010)

Existem duas classes básicas de algoritmos FFT. A primeira classe, chamada decimação no tempo, seu nome deriva do fato de que, no processo de organizar a computação em transformações menores, a sequência $x[n]$ é decomposta em sucessivas subsequências menores. Na segunda classe geral de algoritmo, a sequência de coeficientes de transformada de Fourier discretos $X[k]$ é decomposta em subsequências menores, daí seu nome, decimação na frequência.(OPPENHEIM; SCHAFER, 2010)

Este trabalho será focado particularmente apenas no desenvolvimento e compreensão da classe de algoritmos de decimação no tempo, também chamada pela abreviação DIT, em especial o algoritmo FFT Radix-2.

2 Algoritmo FFT de Decimação no Tempo

2.1 A Técnica

Para alcançar o aumento dramático na eficiência a que aludimos, é necessário decompor o cálculo DFT em cálculos DFT sucessivamente menores. Neste processo, exploramos a simetria e a periodicidade do exponencial complexo $W_N^{kn} = e^{-j(2\pi/N)kn}$. Algoritmos nos quais a decomposição é baseada na decomposição da sequência x (n), em subsequências sucessivamente menores, são chamados de algoritmos de decimação no tempo. O princípio da decimação no tempo é mais convenientemente ilustrado considerando o caso especial de N uma potência inteira de 2. (OPPENHEIM; SCHAFER, 1975)

Geralmente para descrever o tamanho das DFTS se utiliza da notação r, para o cálculo da DFT de ponto-N tenha um padrão regular. O Número r é chamado de "raiz"do algoritmo FFT, ou "Radix"no inglês. Nesta etapa vamos apresentar o algoritmo conhecido pelo nome de "Radix-2", o qual é de longe o mais amplamente utilizado.

Suponha que temos uma sequência $x(n)$ cujo comprimento N é uma potência de dois, isto é, $N = 2^l$. Agora, vamos expressar a relação para a DFT dada na equação

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \tag{1}$$

quebrando o somatório em duas partes, uma com os elementos $x(n)$ de índice par e outra com os elementos $x(n)$ de índice ímpar, obtendo

$$= \sum_{n=0}^{N/2-1} x(2n) W_N^{2nk} + \sum_{n=0}^{N/2-1} x(2n+1) W_N^{(2n+1)k} \tag{2}$$

$$= \sum_{n=0}^{N/2-1} x(2n) W_N^{2nk} + W_N^k \sum_{n=0}^{N/2-1} x(2n+1) W_N^{2nk} \tag{3}$$

Se notarmos que para N par temos

$$W_N^{2nk} = e^{-j(2\pi/N)2nk} = e^{-j[(2\pi/(N/2))nk} = W_{N/2}^{nk}, \tag{4}$$

então a equação 3 se torna

$$X(k) = \sum_{n=0}^{N/2-1} x(2n) W_{N/2}^{nk} + W_N^k \sum_{n=0}^{N/2-1} x(2n+1) W_{N/2}^{nk} \tag{5}$$

e podemos ver que cada somatório pode representar uma DFT distinta de tamanho $N/2$. Portanto, uma DFT de tamanho N pode ser calculada através de duas DFTs de tamanho $N/2$, além das multiplicações por W_N^k . Note que cada nova DFT tem somente $N/2$ coeficientes e para o cálculo desses coeficientes precisamos, agora, de apenas $(N/2)^2$ multiplicações complexas. Adicionalmente, já que temos um coeficiente distinto W_N^k para cada k entre 0 e N 1, precisamos efetuar N multiplicações por W_N^k.(DINIZ *et al.*, 2014) Portanto, o cálculo da DFT de acordo com a equação 5 requer

$$2\Big(\frac{N}{2}\Big)^2 + N = \frac{N^2}{2} + N \tag{6}$$

multiplicações complexas. Como $(N + N^2/2)$ é menor que N^2 para $N > 2$, a equação 5 já resulta num decréscimo de complexidade quando comparada ao cálculo usual da DFT. Devemos também comparar o número de adições complexas das duas formas de cálculo. O cálculo usual de uma DFT de comprimento N precisa de um total de $N(N-1) = N^2 - N$ adições. Na equação 5, precisamos calcular duas DFTs de comprimento $N/2$. Em seguida, após a multiplicação por W_N^k, devem-se efetuar as N adições das duas DFTs parciais, uma para cada k entre 0 e $N-1$. (DINIZ *et al.*, 2014) Portanto, o número total de adições complexas na equação 5 é

$$2\Big[\Big(\frac{N}{2}\Big)^2 - \frac{N}{2}\Big] + N = \frac{N^2}{2} \tag{7}$$

o que também corresponde a uma redução da complexidade.(DINIZ *et al.*, 2014) Do que foi tratado até agora, podemos observar com clareza que o procedimento mostrado na equação 5 é aplicado recursivamente a cada uma das DFTs resultantes até que todas sejam de comprimento 2. O procedimento completo é formalizado em um algoritmo escrevendo-se a equação 5 como

$$X(k) = X_e(k) + W_N^k X_o(k) \tag{8}$$

ode $X_e(k)$ e $X_o(k)$ são, respectivamente as DFTs de compirmento $N/2$ das amostras com índices pares e ímpares de $x(n)$, isto é,

$$X_e(k) = \sum_{n=0}^{N/2-1} x(2n) W_{N/2}^{nk} = \sum_{n=0}^{N/2-1} x_e(n) W_{N/2}^{nk} \tag{9}$$

e

$$X_o(k) = \sum_{n=0}^{N/2-1} x(2n+1) W_{N/2}^{nk} = \sum_{n=0}^{N/2-1} x_o(n) W_{N/2}^{nk} \tag{10}$$

Um diagrama para o algoritmo completo da FFT descrito anteriormente é obtido pela repetição da célula básica da Figura 1 para $L = N, N/2, ..., 2$ e para $k = 0, 1, ..., (L/2 - 1)$. (DINIZ *et al.*, 2014)

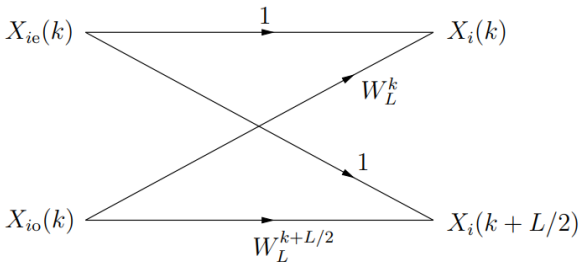

Figura 1: Célula básica do algoritmo de FFT com decimação no tempo
Fonte: (DINIZ *et al.*, 2014)

2.2 A Complexidade

A aplicação recursiva do procedimento aqui descrito pode conduzir o cálculo de uma DFT de comprimento $N = 2^l$, ao longo de l etapas, até o cálculo de 2^l DFTs de comprimento 1, porque cada etapa converte uma DFT de comprimento L em duas DFTs de comprimento $L/2$, mais uma multiplicaçaão complexa por W_L^k e uma soma complexa. Portanto, supondo que $M(N)$ e $A(N)$ são, respectivamente, os números de multiplicações e adições complexas necessárias para se calcular uma DFT de comprimento N, valem as seguintes relações: (DINIZ *et al.*, 2014)

$$M(N) = 2M\Big(\frac{N}{2}\Big) + N \tag{11}$$

$$A(N) = 2A\Big(\frac{N}{2}\Big) + N \tag{12}$$

A fim de calcularmos os valores de $M(N)$ e $A(N)$, temos de resolver essas equações recursivas. As condições iniciais são $M(1) = 1$ e $A(1) = 0$, já que uma DFT de comprimento 1 não requer somas, e envolve uma multiplicação por W_l^0.Podemos calcular o número de multiplicações empregando a mudança de variáveis $N = 2^l$ e $T(l) = M(N)/N$. Com isso, a equação 11 se torna

$$T(l) = T(l-1) + 1$$

Como $T(0) = M(1)$, então $T(l) = l + 1$. Portanto,

$$\frac{M(N)}{N} = 1 + log_2 N,$$

e assim logo concluímos que o número verdadeiro de multiplicações é

$$M(N) = N + Nlog_2 N.$$

Para calcular o número de somas, podemos usar a mesma mudançaa de variáveis, isto á, fazer $N = 2^l$ e $T(l) = A(N)/N$. Com isso, a equação 12 se torna

$$T(l) = T(l-1) + 1,$$

mas desta vez $T(0) = A(1) = 0$, e então $T(l) = l$. Portanto,

$$\frac{A(N)}{N} = log_2 N,$$

e assim logo concluímos que o número de somas é

$$A(N) = Nlog_2 N.$$

Em síntese, a FFT pode ser calculada usando-se $Nlog_2 N$ multiplicações e adições complexas, o que significa uma economia da ordem de $\frac{N}{log_2 N}$, quando comparada com a implementação direta. (DINIZ *et al.*, 2014)

2.3 O Algoritmo

Listings


```
1
2  % Funcao FFT Radix 2 Decimacao no Tempo (DIT)
3  % Processamento de Sinais
4  % Welliton Jhonathan Leal Babinski
5
6  %Sinal discreto e nao periodico de entrada
7  %x = [2; 4; 5 ;6 ;9 ;8 ;7; 2 ;12 ;13; 14 ;2;
       12; 7; 2; 14];
8
9  function [y] = fftR2dit(x)
10
11 tic
12 % iniciando a contagem do tempo
13
14 p = nextpow2(length(x));
15 %checando o tamanho do vetor de entrada
16 x = [x zeros(1,(2^p)-length(x))];
17 %complementando o vetor com zeros se
       necessario
18 N = length(x);
19 %calculando o tamanho do array
20 R = log2(N);
21 %calculando o n mero de estagios de conversao
22 Metade = 1;
23 %Setando o valor de "Metade" inicial
24 x = bitrevorder(x);
25 %colocando os samples em ordem de bit-reversa
26
27 for stage = 1:R
28 %Estagios da transformada
29     for index = 0:(2^stage):(N-1)
30 %serie de celulas (butterflies) pra cada
       estagio
31         for n = 0:(Metade-1)
32 %criando a celula (butterfly) e salvando os
       resultados
33             pos = n + index + 1;
34 %indexando a amostra
35             pow = (2^(R-stage))*n;
36 %parte da potencia do multiplicador complexo
37             WN = exp((-1i)*(2*pi)*pow/N);
38 %Multiplicador complexo
39             a = x(pos) + x(pos+Metade).*WN;
40 %criando a primeira parte da celula
41             b = x(pos) - x(pos+Metade).*WN;
42 %criando a segunda parte da celula
43             x(pos) = a;
44 %salvando o calculo da primeira parte
45             x(pos + Metade) = b;
46 %salvando o calculo da segunda parte
47         end
48     end
49     Metade = 2 * Metade;
50 %calculando o proximo valor "Metade"
51 end
52 y = x;
53 toc
54 %finalizando a contagem do tempo
55
56 L = 0:length(x)-1;
57 figure
58 %plotando os modulos das amostras
59 subplot(2,1,1)
60 stem(L,abs(y))
61 title("M dulo com Algoritmo de Decima  o no
       Tempo FFT Radix-2");
62 xlabel('k');
63
64 %plotando as fases das amostras
65 subplot(2,1,2)
66 stem(L,angle(y))
67 title("Fase com Algoritmo de Decima  o no
       Tempo FFT Radix-2");
68 xlabel('k');
69 end
```

Listing 1: Algoritmo de Decimação no Tempo FFT Radix-2

3 Resultados

A partir de um sinal de entrada discreto e não periódico foi possível verificarmos a eficiencia de ambos algoritmos, o Radix-2 implementado e o fft nativo do Matlab, o sinal de entrada segue abaixo:

```
>> x = [2; 4; 5 ;6 ;9 ;8 ;7; 2 ;12 ;13; 14 ;2; 12; 7; 2; 14]
```

3.1 Respostas do Algoritmo FFT Radix-2 DIT Implementado

3.2 Respostas do Algoritmo FFT Nativo do Matlab

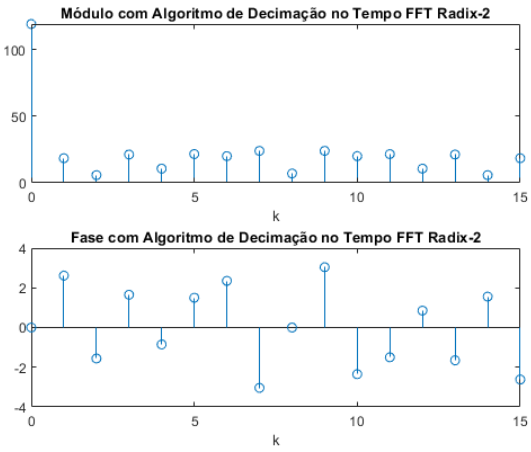

Figura 2: Módulo e Fase do sinal utilizando o Radix-2
Fonte: *Software* Matlab

```
>> fftR2dit(x)
Elapsed time is 0.013817 seconds.
>> fftR2dit(x)
Elapsed time is 0.010945 seconds.
>> fftR2dit(x)
Elapsed time is 0.001077 seconds.
>> fftR2dit(x)
Elapsed time is 0.001469 seconds.
>> fftR2dit(x)
Elapsed time is 0.001068 seconds.
```

Figura 3: Algumas respostas de tempo do Algoritmo FFT Radix-2 Implementado
Fonte: *Software* Matlab

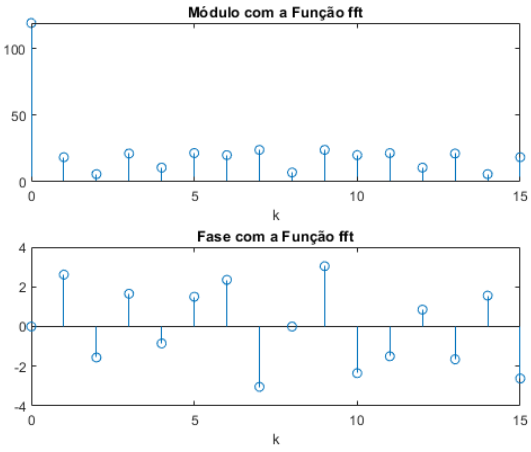

Figura 4: Módulo e Fase do sinal utilizando a fft do Matlab
Fonte: *Software* Matlab

4 Conclusão

A realização deste trabalho e pesquisa, proporcionou primeiro entendermos a construção da base dos algoritmos conhecidos como Transformada Rápida de Fourier

```
>> fft_func(x)
Elapsed time is 0.000154 seconds.
>> fft_func(x)
Elapsed time is 0.000142 seconds.
>> fft_func(x)
Elapsed time is 0.000087 seconds.
>> fft_func(x)
Elapsed time is 0.000085 seconds.
>> fft_func(x)
Elapsed time is 0.000090 seconds.
```

Figura 5: Algumas Respostas de tempo do fft do Matlab
Fonte: *Software* Matlab

(FFT), partindo desde a sua formalização matemática até a aplicação prática, existem vários algoritmos mas alguns se destacam como o Radix-2 analisado.

Possibilitou também visualizarmos a sua eficiencia diante de qualquer metodo direto de DFT, afinal a complexidade que era em torno de N^2 multiplicações complexas limitando rigidamente o uso prático de sinais longos, foi reduzida consideravelmente para $Nlog_2N$. Nessa mesma linha observamos a superioridade do algoritmo FFT nativo do Matlab sobre o desenvolvido, a maior rapidez de milisegundos observada nos resultados resulta uma enorme diferença na otimização de velocidade de grandes processamentos.

Finalizando, graças ao algoritmo eficiente desenvolvido em 1965 por Cooley Tukey o sinais analógicos puderam ser trazidos para o mundo digital com mais facilidade e possibilitaram centenas de novas formas de manipulações e aplicações dos mesmos no meio computacional, assim como o transistor foi pro mundo analógico, o algoritmo da FFT é provavelmente uma das, senão a mais importante invenção do mundo digital.

Referências

DINIZ, P. S.; SILVA, E. A. da; NETTO, S. L. **Processamento Digital de Sinais-: Projeto e Análise de Sistemas**. [S.l.]: Bookman Editora, 2014.

OPPENHEIM, A.; SCHAFER, R. **Discrete-Time Signal Processing, Third Editon**. [S.l.]: Prentice Hall, 2010.

OPPENHEIM, A. V.; SCHAFER, R. W. **Digital Signal Processing:(by) Alan V. Oppenheim (and) Ronald W. Schafer**. [S.l.]: Prentice-Hall, 1975.