Factoring the DFT: Derivation of the FFT algorithm.

Andrew A. Lamb

January 9, 2003

1 Introduction

This document derives the Fast Fourier Transform (FFT) algorithm from the definition of the Discrete Fourier Transform (DFT) using matrix notation, and then shows how that the FFT algorithm is the same as a clever matrix factorization of the DFT. The intent of the document is to show that special properties of complex exponentials must be exploited to derive the FFT, casting serious doubt in the author's mind if an automatic factorization algorithm for general matrices is feasible. Note that this derivation follows closely the derivation given by Sewell[4].

2 Notation, Definitions and Identities

In this section, we will explain the notation and prove the key identities of complex exponentials used in the FFT derivation.

The N-point Discrete Fourier Transform $X[k]$ for a N sample discrete time signal is defined by the following equation (see Oppenheim[3] for a full treatment).

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk} \tag{1}$$

where W_N is the Nth root of unity such that

$$W_N = e^{-\frac{j2\pi}{N}} \quad and \quad j = \sqrt{-1} \tag{2}$$

Euler's relation relates complex exponentials to sinusoids in the following way.

$$e^{j\omega} = \cos(\omega) + j\sin(\omega) \tag{3}$$

One interpretation of $e^{j\omega}$ is a unit length vector in the complex plane. The unit vector starting at the origin is at an angle of ω relative to the real axis as shown in Figure 1. This interpretation might be helpful to the reader in verifying properties of W_N. W_N plays prominently in both the definition of the DFT and its properties are essential to the FFT. The required identities of W_N and their derivations are shown in Equation 4.

$$\begin{aligned} W_N^2 &= e^{-\frac{j2\pi}{N}2} = e^{-\frac{j2\pi}{N/2}} = W_{N/2} \\ W_N^N &= e^{-\frac{j2\pi}{N}N} = e^{-j2\pi} = 1 \\ W_N^{N/2} &= e^{-\frac{j2\pi}{N}\frac{N}{2}} = e^{-j\pi} = -1 \end{aligned} \tag{4}$$

3 FFT derivation

To start our FFT derivation, we first cast the problem of computing the N values of $X[k]$ from the N values of $x[n]$ as a matrix multiplication such that $\vec{x}\mathbf{A} = \vec{X}$. $\vec{x}$ is a row vector of length N such that $x_i = x[i]$ and $\vec{X}$ is also a row vector of length N such that $X_i = X[i]$. For our purposes, we assume that N is a power of 2.

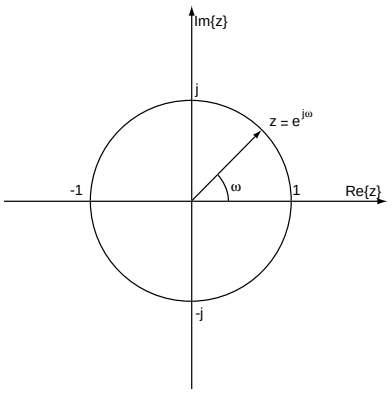

Figure 1: $z = e^{j\omega}$ as a unit vector in the complex plane. If we define $z = Re\{z\} + j \cdot Im\{z\}$ then $Re\{z\} = \cos(\omega)$ and $Im\{z\} = \sin(\omega)$.

If the length of $x[n]$ is not a power of two, we can "zero pad" $x[n]$ to the appropriate length. Zero padding $x[n]$ does not introduce any extraneous frequency components to $X[k]$, so it can always be done safely. See Oppenheim[3] for a full theoretical treatment on the subject. From Equation 1 we can see that $\mathbf{A}$ must be a NxN matrix such that $a_{i,k} = W_N^{ik}$. We can write out the equation $\vec{x}\mathbf{A} = \vec{X}$ in the following way.

$$\begin{bmatrix} x_0 & x_1 & \cdots & x_{N-1} \end{bmatrix} \begin{bmatrix} W_N^{0\cdot 0} & W_N^{0\cdot 1} & \cdots & W_N^{0\cdot(N-1)} \\ W_N^{1\cdot 0} & W_N^{1\cdot 1} & \cdots & W_N^{1\cdot(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ W_N^{(N-1)\cdot 0} & W_N^{(N-1)\cdot 1} & \cdots & W_N^{(N-1)\cdot(N-1)} \end{bmatrix} = \begin{bmatrix} X_0 & X_1 & \cdots & X_{N-1} \end{bmatrix} \tag{5}$$

The non obvious essential step of the derivation is rewriting Equation 5 by dividing up both $\vec{x}$ and $\vec{X}$ into their even indexed elements followed by their odd indexed elements (Equation 6) and modifying $\mathbf{A}$ appropriately (Equation 7).

$$\begin{array}{rcl} \left[\begin{array}{c|c} \vec{x}_{even} & \vec{x}_{odd} \end{array}\right] & = & \left[\begin{array}{cccc|cccc} x_0 & x_2 & \cdots & x_{N-2} & x_1 & x_3 & \cdots & x_{N-1} \end{array}\right] \\ \left[\begin{array}{c|c} \vec{X}_{even} & \vec{X}_{odd} \end{array}\right] & = & \left[\begin{array}{cccc|cccc} X_0 & X_2 & \cdots & X_{N-2} & X_1 & X_3 & \cdots & X_{N-1} \end{array}\right] \end{array} \tag{6}$$

$$\left[\begin{array}{c|c} \vec{x}_{even} & \vec{x}_{odd} \end{array}\right] \left[\begin{array}{c|c} \mathbf{P} & \mathbf{Q} \\ \hline \mathbf{R} & \mathbf{S} \end{array}\right] = \left[\begin{array}{c|c} \vec{X}_{even} & \vec{X}_{odd} \end{array}\right] \tag{7}$$

Our goal is to derive relationships between $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{S}$ such that we can reduce the computational requirements. We start by defining two new matrices $\mathbf{B}$ and $\mathbf{D}$. We let $\mathbf{B}$ be the $(N/2)$x$(N/2)$ DFT matrix, which means that $b_{i,k} = W_{N/2}^{ik}$.

$$\mathbf{B} = \begin{bmatrix} W_{N/2}^{0\cdot 0} & W_{N/2}^{0\cdot 1} & \cdots & W_{N/2}^{0\cdot(N/2-1)} \\ W_{N/2}^{1\cdot 0} & W_{N/2}^{1\cdot 1} & \cdots & W_{N/2}^{1\cdot(N/2-1)} \\ \vdots & \vdots & \ddots & \vdots \\ W_{N/2}^{(N/2-1)\cdot 0} & W_{N/2}^{(N/2-1)\cdot 1} & \cdots & W_{N/2}^{(N/2-1)\cdot(N/2-1)} \end{bmatrix} \tag{8}$$

And we let $\mathbf{D}$ be a diagonal $(N/2)$x$(N/2)$ matrix such that $d_{i,i} = W_N^i$.

$$\mathbf{D} = \begin{bmatrix} W_N^0 & 0 & \cdots & 0 \\ 0 & W_N^1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & W_N^{(N/2-1)} \end{bmatrix} \tag{9}$$

Now, we are ready to express $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{S}$ in terms of $\mathbf{B}$ and $\mathbf{D}$.

$$\begin{array}{lll}
p_{i,k} = & a_{2i,k} = W_N^{2ik} = W_{N/2}^{ik} = b_{i,k} & \Rightarrow \quad \mathbf{P} = \mathbf{B} \\
q_{i,k} = & a_{2i,N/2+k} = W_N^{2i(N/2+k)} = W_N^{Ni} W_N^{2ik} = 1^i W_{N/2}^{ik} = b_{i,k} & \Rightarrow \quad \mathbf{Q} = \mathbf{B} \\
r_{i,k} = & a_{2i+1,k} = W_N^{(2i+1)k} = W_N^{2ik} W_N^k = W_{N/2}^{ik} W_N^k = b_{i,k} d_{k,k} & \Rightarrow \quad \mathbf{R} = \mathbf{BD} \\
s_{i,k} = & a_{2i+1,N/2+k} = W_N^{(2i+1)(N/2+k)} = W_N^{Ni+2ik+N/2+k} = W_N^{Ni} W_N^{2ik} W_N^{N/2} W_N^k & \\
= & (1^i) W_{N/2}^{ik} (-1) W_N^k = -W_{N/2}^{ik} W_N^i = -b_{i,k} d_{k,k} & \Rightarrow \quad \mathbf{S} = -\mathbf{BD}
\end{array} \tag{10}$$

We can now rewrite Equation 7 in terms of $\mathbf{B}$ and $\mathbf{D}$ in the following manner:

$$\left[\begin{array}{c|c} \vec{x}_{even} & \vec{x}_{odd} \end{array} \right] \quad \left[\begin{array}{c|c} \mathbf{B} & \mathbf{B} \\ \hline \mathbf{BD} & -\mathbf{BD} \end{array} \right] \quad = \quad \left[\begin{array}{c|c} \vec{X}_{even} & \vec{X}_{odd} \end{array} \right] \tag{11}$$

If we expand out Equation 11, the FFT algorithm for calculating the DFT falls out.

$$\begin{array}{lllll}
\vec{x}_{even}\mathbf{B} & + & \vec{x}_{odd}\mathbf{BD} & = & \vec{X}_{even} \\
\vec{x}_{even}\mathbf{B} & - & \vec{x}_{odd}\mathbf{BD} & = & \vec{X}_{odd}
\end{array} \tag{12}$$

With all of the math behind us, we can now calculate the DFT of $\vec{x}$ using the following algorithm.

1. Compute $\vec{x}_{even}\mathbf{B}$ and $\vec{x}_{odd}\mathbf{B}$. This can be done recursively by remembering that $\mathbf{B}$ is simply the $N/2$ DFT matrix, so we are calculating two $N/2$-point DFTs on $\vec{x}_{even}$ and $\vec{x}_{odd}$. Since we are assuming that N is a power of two, $N/2$ is also a power of two. The base case is $N/2 = 1$ from which it follows that $\mathbf{B} = [1]$.

2. Compute $\vec{u} \equiv (\vec{x}_{odd}\mathbf{B})\mathbf{D}$ using the following recursive formula which requires $N/2 + N/2 = N$ multiplications. Note that $\vec{u}$ is a $N/2$ length row vector.

$$\begin{array}{lcl}
d_{0,0} & = & W_N \\
u_k & = & (\vec{x}_{odd}\mathbf{B})_k d_{k,k} \\
d_{k+1,k+1} & = & d_{k,k} W_N
\end{array} \tag{13}$$

3. Calculate $\vec{X}_{even}$ and $\vec{X}_{odd}$ using the formulas $\vec{X}_{even} = \vec{x}_{even}\mathbf{B} + \vec{u}$ and $\vec{X}_{odd} = \vec{x}_{even}\mathbf{B} - \vec{u}$.

4. As the final step, combine $\vec{X}_{even}$ and $\vec{X}_{odd}$ appropriately to form the overall $\vec{X}$.

The work required by the above algorithm can be described by the recurrence $T(N) = 2T(N/2) + N$ which has as its solution $T(N) = O(N \lg N)$. This is what we expected as we know that the FFT requires $O(N \lg N)$ time.

This derivation demonstrates where the asymptotic savings of the FFT comes from. There are myriads of other ways to optimize the computation of the DFT to reduce hidden constant factors. Optimizations such as converting from a recursive to an iterative implementation, performing the computation "in place" (eg doing the computation without explicitly splitting up $\vec{x}$ into $\vec{x}_{even}$ and $\vec{x}_{odd}$ and using butterfly structures are all well known and beyond the scope of this document. For a very fast runtime implementation, the author suggests using FFTW[2, 1], which is a library for calculating the DFT using the FFT which uses runtime tuning to maximize performance.

4 FFT as Factorization

It is also instructive to view the algorithm above as a factorization of the DFT matrix $\mathbf{A}$. (I am still working out this part)

5 Conclusion

References

[1] M. Frigo. A Fast Fourier Transform Compiler. PLDI, 1999.

[2] M. Frigo and S. Johnson. Home page of fftw. http://www.fftw.org.

[3] Alan V. Oppenheim, Ronald W. Shafer, and John R. Buck. *Discrete-Time Signal Processing*. Prentice Hall, 1999.

[4] Granville Sewell. *Computational Methods of Linear Algebra*. Ellis Horwood, Chichester, England, 1990.