

The Fast Fourier Transform

Notes by B. Osofsky

1 The Discrete Fourier Transform

The Discrete Fourier Transform takes a sequence of complex numbers $\{c_0, c_1, \dots, c_{n-1}\}$ to the sequence $\{y_0, y_1, \dots, y_{n-1}\}$ where $y_k = \sum_{m=0}^{n-1} c_m e^{\frac{2\pi m k}{n} i}$ for $i = \sqrt{-1}$. That is,

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \cdots & \omega_n^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \cdots & \omega_n^{(n-1)^2} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix}$$

where ω_n is a primitive n^{th} root of unity, here $e^{\frac{2\pi i}{n}}$. Call that coefficient matrix $F_n[\omega_n]$. Let an overbar denote complex conjugate, and let A^H denote the complex conjugate of the transpose of the matrix A . For any n^{th} root of unity ω distinct from 1, since $0 = \omega^n - 1 = (\omega - 1) \sum_{k=0}^{n-1} \omega^k$ and $\omega - 1 \neq 0$, $\sum_{k=0}^{n-1} \omega^k$ must be equal to 0. Also, $|\omega|^2 = \omega \bar{\omega} = 1$. Then by straightforward multiplication for any n^{th} root of unity

$$\begin{aligned} F[\omega_n]^H &= F[\bar{\omega}_n] \\ F[\omega_n]^H F[\omega_n] &= n\mathbf{I} \end{aligned}$$

where \mathbf{I} is the $n \times n$ identity matrix. Hence, given the vector of y 's, one gets the c 's by multiplying

$$\frac{1}{n} F[\bar{\omega}_n] \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

Thus both the Discrete Fourier Transform matrix and its inverse are obtained by multiplying by a matrix $F[\tilde{\omega}_n]$ for some primitive n^{th} root of unity, and dividing everything by n if taking the inverse. In general this involves $\mathcal{O}(\mathcal{E}n^2)$ (complex) flops, which, for n large, can become prohibitive in terms of time involved in the computation. The Fast Fourier Transform reduces the number of flops to a much more manageable $\mathcal{O}(n \cdot \log_2 n)$. In this note we discuss the two major ideas used to implement this Fast Fourier Transform. One of the ideas, namely that the discrete Fourier transform of size $2m$ can be expressed as a sum of two discrete Fourier transforms of size m extends back to a lemma attributed to Danielson and Lanczos (1942). Indeed, some sources call the Fast Fourier Transform the Danielson-Lanczos Algorithm, although it is not clear that they had the second necessary insight.

2 The Fast Fourier Transform

Let us analyze the matrix $F[\omega_n]$ when $n = 2m$. We first note that $\omega_{2m}^m = -1$ since it is a square root of unity which is not equal to 1, and $\omega_{2m}^2 = \omega_m$. Set $F_k = F[\omega_k]$ to simplify

notation. If one begins indexing from 0 rather than 1, the entry in the k, ℓ position is $\omega_{2m}^{k\ell}$. Rewriting these powers of ω_{2m} and putting in a divider of single dots to help display what is happening, we see

$$F_{2m} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & \cdots & 1 & 1 \\ 1 & \omega_{2m} & \omega_{2m}^2 & \omega_{2m} \cdot \omega_{2m}^2 & \cdots & \cdots & (\omega_{2m}^2)^{m-1} & \omega \cdot (\omega_{2m}^2)^{m-1} \\ 1 & \omega_{2m}^2 & (\omega_{2m}^2)^2 & \omega_{2m} \cdot (\omega_{2m}^2)^2 & \cdots & \cdots & (\omega_{2m}^2)^{2m-2} & (\omega_{2m}^2)^{(2m-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 1 & \omega_{2m}^{m-1} & (\omega_{2m}^2)^{m-1} & (\omega_{2m}^{m-1}) \cdot (\omega_{2m}^2)^{m-1} & \cdots & \cdots & (\omega_{2m}^2)^{(m-1)^2} & (\omega_{2m}^{m-1}) \cdot (\omega_{2m}^2)^{(m-1)^2} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & -1 & 1 & -1 & \cdots & \cdots & 1 & -1 \\ 1 & -\omega_{2m} & \omega_{2m}^2 & -\omega_{2m} \cdot \omega_{2m}^2 & \cdots & \cdots & (\omega_{2m}^2)^{m-1} & -\omega_{2m} \cdot (\omega_{2m}^2)^{m-1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 1 & -\omega_{2m}^{m-2} & (\omega_{2m}^2)^{(m-1)} & -\omega_{2m}^{m-2} \cdot (\omega_{2m}^2)^{(m-1)} & \cdots & \cdots & (\omega_{2m}^2)^{(m-1)} & -\omega_{2m}^{m-2} \cdot (\omega_{2m}^2)^{(m-1)} \\ 1 & -\omega_{2m}^{m-1} & (\omega_{2m}^2)^{(2m-1)} & -\omega_{2m}^{m-1} \cdot (\omega_{2m}^2)^{(2m-1)} & \cdots & \cdots & (\omega_{2m}^2)^{(m-1)} & -\omega_{2m}^{m-1} \cdot (\omega_{2m}^2)^{(m-1)} \end{bmatrix}$$

Note that the even numbered columns are in powers of $\omega_{2m}^2 = \omega_m$, and the odd numbered columns are the entry in column 1 (the leading 1's are in column 0) times a power of ω_{2m}^2 . There actually are 4 copies of F_m embedded in F_{2m} if we can separate the even columns from the odd columns. A way to get rid of columns we do not want to consider is to multiply them by 0. So given the data $C = [c_0 \ c_1 \ \cdots \ c_{2m-2} \ c_{2m-1}]^T$, define four new vectors of data,

$$\begin{aligned} C_{\mathcal{E}} &= [c_0 \ 0 \ c_2 \ 0 \ c_4 \ \cdots \ 0 \ c_{2m-2}]^T \\ c_{\mathcal{E}} &= [c_0 \ c_2 \ c_4 \ \cdots \ c_{2m-2}]^T \\ C_{\mathcal{O}} &= [0 \ c_1 \ 0 \ c_3 \ 0 \ \cdots \ 0 \ c_{2m-1}]^T \\ c_{\mathcal{O}} &= [c_1 \ c_3 \ c_5 \ \cdots \ c_{2m-1}]^T \end{aligned}$$

The form of F_{2m} as given above shows that

$$\begin{aligned} F_{2m}C &= F_{2m}C_{\mathcal{E}} + F_{2m}C_{\mathcal{O}} \\ &= \begin{bmatrix} F_m c_{\mathcal{E}} \\ F_m c_{\mathcal{E}} \end{bmatrix} + \begin{bmatrix} \text{diag}(\omega_{2m}^k) F_m c_{\mathcal{O}} \\ -\text{diag}(\omega_{2m}^k) F_m c_{\mathcal{O}} \end{bmatrix} \end{aligned} \tag{*}$$

where $\text{diag}(\omega_{2m}^k)$ is the matrix with zeros everywhere except for the power ω_{2m}^k in the k, k position for $0 \leq k \leq m - 1$. Remember that, for any given data C , this is just a sum of two $2m \times 1$ matrices with complex entries. Thus the Discrete Fourier Transform of size $2m$ can be computed by computing two Discrete Fourier Transforms of size m , multiplying by an $m \times m$ diagonal matrix, and then combining the resulting column matrices in a rather simple pattern involving only addition, subtraction, and concatenation of matrices. Instead of $\mathcal{O}(m^2)$ flops, one has $\mathcal{O}(m)$ flops once the smaller Fourier transforms have been computed.

But even more significant, this procedure can clearly be applied recursively. If the initial number, say n , of data points is a power of 2, then after $\log_2 n$ iterations one is reduced to the case of n F_1 's, and F_1 of a value c is just the number c itself. So if you know which c 's correspond to which F_1 's, you simply use $(*)$ to combine pairs of F_1 's to get F_2 's, and so on back to F_n , in $\mathcal{O}(n \cdot \log_2 n)$ steps.

Let us see how this works for $n = 8$. We have $\omega_8 = \sqrt{i} = \frac{1+i}{\sqrt{2}}$ and, putting parentheses around previous computed values which do not need to be recomputed, we circle the only arithmetic operations that have to be performed. Other operations which must be performed, including computing the four cube roots of unity in the upper half plane and the initial ordering of the input data, are of a bookkeeping nature.

$$\begin{aligned} c &= [c_0 \ c_1 \ c_2 \ c_3 \ c_4 \ c_5 \ c_6 \ c_7]^T; \\ c_{\mathcal{E}} &= [c_0 \ c_2 \ c_4 \ c_6]^T; \ c_{\mathcal{O}} = [c_1 \ c_3 \ c_5 \ c_7]^T; \\ c_{\mathcal{E}\mathcal{E}} &= [c_0 \ c_4]^T; \ c_{\mathcal{E}\mathcal{O}} = [c_2 \ c_6]^T; \ c_{\mathcal{O}\mathcal{E}} = [c_1 \ c_5]^T; \ c_{\mathcal{O}\mathcal{O}} = [c_3 \ c_7]^T; \\ c_{\mathcal{E}\mathcal{E}\mathcal{E}} &= c_0; \ c_{\mathcal{E}\mathcal{E}\mathcal{O}} = c_4; \ c_{\mathcal{E}\mathcal{O}\mathcal{E}} = c_2; \ c_{\mathcal{E}\mathcal{O}\mathcal{O}} = c_6; \ c_{\mathcal{O}\mathcal{E}\mathcal{E}} = c_1; \ c_{\mathcal{O}\mathcal{E}\mathcal{O}} = c_5; \ c_{\mathcal{O}\mathcal{O}\mathcal{E}} = c_3; \ c_{\mathcal{O}\mathcal{O}\mathcal{O}} = c_7; \end{aligned}$$

Once we have the ordering of the c_i , there are no flops at this level.

$$\begin{aligned} \text{The } F_2 \begin{bmatrix} c_\alpha \\ c_b \end{bmatrix} \text{ in order are } F_2 \begin{bmatrix} c_0 \\ c_4 \end{bmatrix} &= \begin{bmatrix} F_1[c_0] + (1) \times F_1[c_4] \\ F_1[c_0] - (1) \times F_1[c_4] \end{bmatrix} = \begin{bmatrix} (c_0) \oplus (1 \otimes (c_4)) \\ (c_0) \ominus (1 \times c_4) \end{bmatrix}; \\ F_2 \begin{bmatrix} c_2 \\ c_6 \end{bmatrix} &= \begin{bmatrix} F_1[c_2] + (1) \times F_1[c_6] \\ F_1[c_2] - (1) \times F_1[c_6] \end{bmatrix} = \begin{bmatrix} c_2 \oplus (1 \otimes c_6) \\ (c_2) \ominus (1 \times c_6) \end{bmatrix}; \quad F_2 \begin{bmatrix} c_1 \\ c_5 \end{bmatrix} = \begin{bmatrix} (c_1) \oplus (1 \otimes c_5) \\ (c_1) \ominus (1 \times c_5) \end{bmatrix}; \\ F_2 \begin{bmatrix} c_3 \\ c_7 \end{bmatrix} &= \begin{bmatrix} (c_3) \oplus (1 \otimes (c_7)) \\ (c_3) \ominus (1 \times c_7) \end{bmatrix}. \end{aligned}$$

We have done 8 additions/subtractions and 4 multiplications at this level.

$$\text{The } F_4 \begin{bmatrix} c_\alpha \\ c_\beta \\ c_\gamma \\ c_\delta \end{bmatrix} \text{ in order are}$$

$$\begin{aligned} F_4 \begin{bmatrix} c_0 \\ c_2 \\ c_4 \\ c_6 \end{bmatrix} &= \begin{bmatrix} F_2 \begin{bmatrix} c_0 \\ c_4 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} F_2 \begin{bmatrix} c_2 \\ c_6 \end{bmatrix} \\ F_2 \begin{bmatrix} c_0 \\ c_4 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} F_2 \begin{bmatrix} c_2 \\ c_6 \end{bmatrix} \end{bmatrix} \\ &= \begin{bmatrix} \left(\begin{bmatrix} (c_0 + c_4) \\ (c_0 - c_4) \end{bmatrix} \right) + \begin{bmatrix} 1 \times (c_2 + c_6) \\ i \times (c_2 - c_6) \end{bmatrix} \\ \left(\begin{bmatrix} (c_0 + c_4) \\ (c_0 - c_4) \end{bmatrix} \right) - \begin{bmatrix} (c_2 + c_6) \\ (i(c_2 - c_6)) \end{bmatrix} \end{bmatrix} = \begin{bmatrix} (c_0 + c_4) \oplus 1 \otimes (c_2 + c_6) \\ (c_0 - c_4) \oplus i \otimes (c_2 - c_6) \\ (c_0 + c_4) \ominus (c_2 + c_6) \\ (c_0 - c_4) \ominus (i \times (c_2 - c_6)) \end{bmatrix}; \\ F_4 \begin{bmatrix} c_1 \\ c_3 \\ c_5 \\ c_7 \end{bmatrix} &= \begin{bmatrix} F_2 \begin{bmatrix} c_1 \\ c_5 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} F_2 \begin{bmatrix} c_3 \\ c_7 \end{bmatrix} \\ F_2 \begin{bmatrix} c_1 \\ c_5 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} F_2 \begin{bmatrix} c_3 \\ c_7 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} (c_1 + c_5) \oplus (1 \otimes (c_3 + c_7)) \\ (c_1 - c_5) \oplus (i \otimes (c_3 - c_7)) \\ (c_1 + c_5) \ominus (c_3 + c_7) \\ (c_1 - c_5) \ominus (i \times (c_3 - c_7)) \end{bmatrix}. \end{aligned}$$

We have done 2 times 4 additions and 2 times 2 multiplications for a total of 8 additions plus 4 multiplications on this level.

And finally

$$\begin{aligned}
 F_8 \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix} &= \begin{bmatrix} F_4 \begin{bmatrix} c_0 \\ c_2 \\ c_4 \\ c_6 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \sqrt{i} & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & 0 & 0 & i\sqrt{i} \end{bmatrix} F_4 \begin{bmatrix} c_1 \\ c_3 \\ c_5 \\ c_7 \end{bmatrix} \\ F_4 \begin{bmatrix} c_0 \\ c_2 \\ c_4 \\ c_6 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \sqrt{i} & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & 0 & 0 & i\sqrt{i} \end{bmatrix} F_4 \begin{bmatrix} c_1 \\ c_3 \\ c_5 \\ c_7 \end{bmatrix} \end{bmatrix} \\
 &= \begin{bmatrix} (c_0 + c_4 + c_2 + c_6) \\ (c_0 - c_4 + ic_2 - ic_6) \\ (c_0 + c_4 - c_2 - c_6) \\ (c_0 - c_4 - ic_2 + ic_6) \\ (c_0 + c_4 - (c_2 - c_6)) \\ (c_0 - c_4) - (ic_2 - ic_6) \\ (c_0 + c_4 + c_2 + c_6) \\ (c_0 - c_4 + ic_2 - ic_6) \end{bmatrix} + \begin{bmatrix} (c_1 + c_5 + c_3 + c_7) \\ \sqrt{i}(c_1 - c_5 + i(c_3 - c_7)) \\ i(c_1 + c_5 - c_3 - c_7) \\ i\sqrt{i}(c_1 - c_5 - ic_3 + ic_7) \\ -(c_1 + c_5 + c_3 + c_7) \\ -\sqrt{i}((c_1 - c_5) + i(c_3 - c_7)) \\ -i(c_1 + c_5 - c_3 - c_7) \\ -i\sqrt{i}((c_1 - c_5) - i(c_3 + c_7)) \end{bmatrix} \\
 &= \begin{bmatrix} (c_0 + c_4 + c_2 + c_6) \oplus 1 \otimes (c_1 + c_5 + c_3 + c_7) \\ (c_0 - c_4 + ic_2 - ic_6) \oplus \sqrt{i} \otimes (c_1 - c_5 + i(c_3 - c_7)) \\ (c_0 + c_4 - c_2 - c_6) \oplus i \otimes (c_1 + c_5 - c_3 - c_7) \\ (c_0 - c_4 - ic_2 + ic_6) \oplus i\sqrt{i} \otimes ((c_1 - c_5) - i(c_3 + c_7)) \\ (c_0 + c_4 + c_2 + c_6) \ominus (c_1 + c_5 + c_3 + c_7) \\ (c_0 - c_4 + ic_2 - ic_6) \ominus (\sqrt{i}(c_1 - c_5) + i\sqrt{i}(c_3 - c_7)) \\ (c_0 + c_4 - c_2 - c_6) \ominus (i(c_1 + c_5 - c_3 - c_7)) \\ (c_0 - c_4 + ic_2 - ic_6) \ominus (i\sqrt{i}(c_1 - c_5) - \sqrt{i}(c_3 + c_7)) \end{bmatrix} \\
 &= \begin{bmatrix} c_0 + c_4 + c_2 + c_6 + c_1 + c_5 + c_3 + c_7 \\ c_0 - c_4 + ic_2 - ic_6 + \sqrt{i}c_1 - \sqrt{i}c_5 + i\sqrt{i}c_3 - i\sqrt{i}c_7 \\ c_0 + c_4 - c_2 - c_6 + ic_1 + ic_5 - ic_3 - ic_7 \\ c_0 - c_4 - ic_2 + ic_6 + i\sqrt{i}c_1 - i\sqrt{i}c_5 + \sqrt{i}c_3 - \sqrt{i}c_7 \\ c_0 + c_4 + c_2 + c_6 - c_1 - c_5 - c_3 - c_7 \\ c_0 - c_4 + ic_2 - ic_6 - \sqrt{i}c_1 - \sqrt{i}c_5 + i\sqrt{i}c_3 - i\sqrt{i}c_7 \\ c_0 + c_4 - c_2 - c_6 - ic_1 + ic_5 - ic_3 - ic_7 \\ c_0 - c_4 + ic_2 - ic_6 - i\sqrt{i}c_1 - i\sqrt{i}c_5 - \sqrt{i}c_3 + \sqrt{i}c_7 \end{bmatrix}
 \end{aligned}$$

Again we have done 8 additions and 4 multiplications at this point. The total number of operations is $8 \log_2 8 = 24$ additions and $4 \log_2 8 = 12$ multiplications, with equal numbers at each level. Were one to just multiply matrices, one would need 64 multiplications (32 if multiplications by ± 1 are not counted) and 56 additions to compute the same transform.

If the FFT program is carefully written not to do multiplications by ± 1 entries in F_n , or at least to avoid many of them, one can save time, but as 2^n gets larger, the savings by avoiding these multiplications by ± 1 become less significant.

The problem now reduces to finding an order for the numbers from 0 to $2^{\log_2 n} - 1$ which will give the correct order when you unscramble the coefficients in this process. Clearly c_0 will always correspond to an even position in successively smaller $c_{\mathcal{E}}$'s, so $c_{\mathcal{E}\mathcal{E}\mathcal{E}\dots\mathcal{E}} = c_0$, one can quickly check that $c_{\mathcal{E}\mathcal{E}\dots\mathcal{E}\mathcal{O}} = c_{2^{n-1}}$; and $c_{\mathcal{O}\mathcal{O}\dots\mathcal{O}} = c_{2^n-1}$ but it may not be immediately evident what other indices such as $c_{\mathcal{E}\dots\mathcal{E}\mathcal{O}\mathcal{E}}$ will be. But we observe the following. In dividing a vector C_* of length $2m$ into even and odd parts, the $2k$ entry goes to the even part and the $2k+1$ entry goes to the odd part, that is, if the least significant bit in the binary expansion of a row index is 0 one goes to the even part and if that index is 1 one goes to the odd part. This least significant bit is then not used, and the index of every entry is then divided by 2. This is precisely the way one computes the binary expansion of the original index. Replace the \mathcal{E} 's by 0 and the \mathcal{O} 's by 1 and you will get the binary expansion of the index if you read the bits from left to right, rather than the standard right to left. So reverse the bits in the binary expansions of the k^{th} index and the result will be the index of the c in position k .

Let us look at the computations with $2^3 = 8$. Replace \mathcal{E} by 0 and \mathcal{O} by 1 and write the bits in reverse order.

$$\begin{aligned} c_{\mathcal{E}\mathcal{E}\mathcal{E}} &= c_{000}; & c_{\mathcal{E}\mathcal{E}\mathcal{O}} &= c_{100}; & c_{\mathcal{E}\mathcal{O}\mathcal{E}} &= c_{010}; & c_{\mathcal{E}\mathcal{O}\mathcal{O}} &= c_{110}; \\ c_{\mathcal{O}\mathcal{E}\mathcal{E}} &= c_{001}; & c_{\mathcal{O}\mathcal{E}\mathcal{O}} &= c_{101}; & c_{\mathcal{O}\mathcal{O}\mathcal{E}} &= c_{011}; & c_{\mathcal{O}\mathcal{O}\mathcal{O}} &= c_{111}. \end{aligned}$$

This bit reversal is the second basic ingredient for making the fast Fourier transform work. Ordering by increasing order of the bit reversed number gives the ordering which will unscramble the c_i as one moves through the successively larger Fourier transforms. We illustrate by arranging the c_i in the appropriate order for $n = 16$. Although many FFT routines actually scramble the c_i to avoid using an extra n storage locations to index the original vector C , here I simply wrote down, in numerical order, the 4 bit binary expansions of the numbers from 0 to 15 in numerical order and multiplied the coefficient in position k by 2^{3-k} rather than by 2^k and then added. They go

$$c_0, c_8, c_4, c_{12}, c_2, c_{10}, c_6, c_{14}, c_1, c_9, c_5, c_{13}, c_3, c_{11}, c_7, c_{15}$$

and using these as the ordered F_1 transforms, one proceeds as above to form 8 F_2 transforms (with 1×8 multiplications coming only from that diagonal matrix and 16 additions/subtractions) followed by 4 F_4 transforms (with 2×4 multiplications and 4×4 additions/subtractions) using pairs of the already computed F_2 transforms and beginning to unscramble the c 's. One then gets 2 F_8 transforms (with 4×2 multiplications and 8×2 additions/subtractions) using these F_4 transforms with further unscrambling of the c 's, and finally 1 F_{16} transform (with 8 multiplications and 16 additions/subtractions) using these F_4 transforms and completely unscrambling the c 's. Again we have $\log_2 2^k \times 2^{k-1}$ multiplications and twice that number of additions/subtractions. After arranging the c_i in the proper order, the total number of flops needed to calculate a product $F_n C$ where $n = 2^k$ is then $k 2^{k-1} (2 + 1)$ which is manageable when n^2 flops is too big. And the bookkeeping needed is of order $n = \log_2 2^n$ so the whole algorithm is $\mathcal{O}(n \log_2 n)$.