Multiplication and the Fast Fourier Transform

Rich Schwartz

October 22, 2012

The purpose of these notes is to describe how to do multiplication quickly, using the fast Fourier transform. As usual, nothing in these notes is original to me.

1 The Discrete Fourier Transform

Let

$$\omega = \exp(2\pi i/n) \tag{1}$$

be the usual nth root of unity.

Let $\delta_{ab} = 1$ if $a = b$ and otherwise 0. We have

$$\sum_{c=0}^{n-1} \omega^{c(b-a)} = n\delta_{ab}. \tag{2}$$

An equivalent version of Equation 2 is that the following two matrices

$$M = \frac{1}{\sqrt{n}} \begin{bmatrix} 1 & 1 & 1 & \cdots \\ 1 & \omega^{-1} & \omega^{-2} & \cdots \\ 1 & \omega^{-2} & \omega^{-4} & \cdots \\ \cdots & & & \end{bmatrix} \tag{3}$$

$$M^{-1} = \frac{1}{\sqrt{n}} \begin{bmatrix} 1 & 1 & 1 & \cdots \\ 1 & \omega^{1} & \omega^{2} & \cdots \\ 1 & \omega^{2} & \omega^{4} & \cdots \\ \cdots & & & \end{bmatrix} \tag{4}$$

are inverses of each other.

The *discrete Fourier transform* is the linear transformation $\Phi : \boldsymbol{C}^n \to \boldsymbol{C}^n$ whose matrix is M. So, given $z = (z_1, ..., z_n)$, we have $\Phi(z) = (Z_1, ..., Z_n)$, where

$$Z_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} z_j \omega^{-kj}. \tag{5}$$

Φ^{-1} has the same form except that $(-kj)$ is replaced by (kj).

The map Φ is an isometry relative to the Hermitian inner product

$$\langle Z, W \rangle = \sum_{i=0}^{n-1} z_i \overline{w_i} \tag{6}$$

One can see this just by noting that, by Equation 2, the rows of M form an orthonormal basis. The same, of course, holds for Φ^{-1}.

2 The Fast Fourier Transform

The straightforward method of computing Φ on an element of $\boldsymbol{C}^n$ takes $O(n^2)$ operations. However, there are algorithms for computing Φ in $O(n \log(n))$ steps. Here, I'll explain the Cooley-Tukey algorithm, which works when n is a power of 2. It seems that this algorithm was known to Gauss. Indeed, if one reads Gauss's proof that the regular 17-gon is constructible, one will see essentially the same idea in it.

Let's compute the Fourier transform of $z = (z_1, ..., z_{2n})$. Let Φ_{2n} denote the Fourier transform relative to $\boldsymbol{C}^{2n}$ and let Φ_n denote the Fourier transform relative to $\boldsymbol{C}^n$. Define

$$\omega_n = \exp(2\pi i/n); \qquad \omega_{2n} = \exp(2\pi i/2n). \tag{7}$$

Let $[X]_k$ denote the kth coordinate of a vector X. We have

$$[\Phi_{2n}(z)]_k = \sum_{i=1}^{2n-1} z_i \omega_{2n}^{ik} = \sum_{i \text{ even}} z_i \omega_{2n}^{ki} + \sum_{i \text{ odd}} z_i \omega_{2n}^{ki}. \tag{8}$$

We define

$$E_k = [\Phi_n(Z_E)]_k, \qquad O_k = [\Phi_n(Z_O)]_k, \tag{9}$$

Here Z_E and Z_O are the vectors made respectively from the even and odd components of Z. With this notation, Equation 8 can be written more succinctly as follows.

$$[\Phi_{2n}(Z)]_k = E_k + \omega_{2n}^k O_k. \tag{10}$$

Equation 10 holds for all $k = 0, ..., 2n-1$, but we only need compute E_k and O_k for $k = 0, ...n-1$ because

$$E_{k+n} = E_k; \qquad O_{k+n} = -O_k. \tag{11}$$

Suppose it takes $T(n)$ operations to compute the Fourier transform of a vector in $\boldsymbol{C}^n$. The trick above shows that we can compute the Fourier tranaform of a vector in $\boldsymbol{C}^{2n}$ using $2T(n) + 8n$. Here is a breakdown of the computation.

- We can compute $\{O_K\}$ and $\{E_k\}$ for $k = 0, ..., n-1$ in $2T(n)$ steps.
- We can compute ω_{2n}^k for $k = 0, ..., 2n-1$ in $2n$ steps.
- It takes 3 more steps to compute each instance of Equation 10. So, this is a total of $6n$ additional steps.

We clearly have $T(2^0) \leq 8$. An easy induction argument shows

$$T(2^k) \leq 8 \times 2^k \times (k+1) \tag{12}$$

This shows that, for $n = 2^k$, the Fourier transform of a vector in $\boldsymbol{C}^n$ can be computed in $O(n \log(n))$ steps. It is worth mentioning that "step" here refers to operations that are more complicated than simple floating point operations. For instance, a typical step involves multiplying a complex number of size $\log(n)$ with an nth root of unity. An actual analysis of the number of floating point operations needed to compute the Fourier transform would depend on how efficiently these individual steps could be done.

3 The Convolution

Let $A = (a_0, ..., a_{n-1})$ and $B = (b_0, ..., b_{n-1})$ be vectors in $\boldsymbol{C}^n$. We define the simple operation

$$AB = (a_0 b_0, ..., a_{n-1} b_{n-1}). \tag{13}$$

We also define the more complicated operation $A * B$ as follows. We introduce the dummy variables $W_0, ..., W_{n-1}$ which formally behave like the roots of unity. That is $W_{i+n} = W_i$ and $W_i W_j = W_{i+j}$. We then define

$$\overline{A} = \sum_{i=0}^{n-1} a_i W_i; \qquad \overline{B} = \sum_{i=0}^{n-1} b_i W_i \tag{14}$$

Technically, these sums belong to the ring $R = \boldsymbol{C}[W_0, ..., W_{n-1}]/I$, where I is the ideal generated by the 2 relations. We then define $A * B = C$, where

$$C \cdot (W_0, ..., W_{n-1}) = \overline{A} \times \overline{B}. \tag{15}$$

The $(\times)$ symbol is just polynomial multiplication in R. The vector C is called the *convolution* of A and B. Here is an example which shows how the operation works.

Example: Suppose $n = 3$ and $A = (1, 2, 5)$ and $B = (8, 4, 7)$. We have

$$\overline{A} = 1W_0 + 2W_2 + 5W_3; \qquad \overline{B} = 8W_0 + 4W_1 + 7W_2.$$

Then

$$\overline{A} \times \overline{B} = (1W_0 + 2W_1 + 5W_2) \times (8W_0 + 4W_1 + 7W_2) =$$

$$W_0(1.8 + 2.7 + 5.4) + W_1(1.4 + 2.8 + 5.7) + W_2(1.7 + 2.4 + 5.8) =$$

$$44W_0 + 55W_1 + 55W_2 = (44, 55, 55) \cdot (W_0, W_1, W_2).$$

Hence $A * B = (44, 55, 55)$.

The two operations we just defined are related by the following equations.

$$\Phi(A)\Phi(B) = \Phi(A * B). \tag{16}$$

$$\Phi^{-1}(A)\Phi^{-1}(B) = \Phi^{-1}(A * B). \tag{17}$$

$$\Phi(A) * \Phi(B) = \Phi(AB). \tag{18}$$

$$\Phi^{-1}(A) * \Phi^{-1}(B) = \Phi^{-1}(AB). \tag{19}$$

These formulas say, in short, that the Fourier transform converts back and forth between the two kinds of products we have defined.

Lemma 3.1 *Equations 16 and 19 are equivalent.*

Proof: Choose A' and B' so that $\Phi(A') = A$ and $\Phi(B') = B$. By Equation 16,

$$AB = \Phi(A')\Phi(B') = \Phi(A' * B').$$

Now apply Φ^{-1} to both sides. We get

$$\Phi^{-1}(AB) = A' * B' = \Phi^{-1}(A) * \Phi^{-1}(B).$$

Hence Equation 16 imples Equation 19. Reversing the steps, we see that Equation 19 implies Equation 16. ♠

Similarly, Equations 17 and 18 are equivalent. We will prove Equation 18. Equation 19 has essentially the same proof.

Lemma 3.2 *Equation 18 is true.*

Proof: Let $\langle A,B\rangle_1$ denote the left hand side of Equation 18 and let $\langle A,B\rangle_2$ denote the right hand side. We want to show that these two operations are equal. Note that both operations are symmetric and bilinear. That is,

$$\langle A,B\rangle = \langle B,A\rangle; \qquad \langle A,B_1+B_2\rangle = \langle A,B_1\rangle + \langle A,B_2\rangle.$$

For this reason, it suffices to check Equation 18 on pairs of basis elements. We choose the standard basis.

Let $A=e_a$ and $B=e_b$. We compute

$$\Phi(A) = \frac{1}{\sqrt{n}}(1,\omega^{-a},\omega^{-2a},...,); \qquad \Phi(B) = \frac{1}{\sqrt{n}}(1,\omega^{-b},\omega^{-2b},...).$$

The dth coefficient C_d of $\Phi(A)*\Phi(B)$ is obtained by adding up the products of all the coefficients whose indices sum to d. This gives us

$$C_d = \frac{1}{n}\sum_{c=0}^{n-1}\omega^{-ca}\omega^{cb-db} = \frac{1}{n}\omega^{-db}\sum_{c=0}^{n-1}\omega^{c(b-a)}. \tag{20}$$

Suppose first that $a\neq b$. Then $AB=(0,...,0)$ and $\Phi(AB)=0$. Equations 20 and 2 say that $C_d=0$ for all d. Hence, Equation 18 holds when $a\neq b$.

When $a=b$ we have $AB=A$. In this case, Equations 20 and 2 say that $C_d=\omega^{-da}$. But then $C=\Phi(AB)$, as claimed. ♠

4 A Multiplication Algorithm

We choose two integers M and N and write them (as usual) in base 10:

$$M=\sum_{i=0}^{k}a_i10^i; \qquad N=\sum_{i=1}^{k}b_i10^i. \tag{21}$$

We take k the same in both cases, but allow the possibility that some of the last coefficients are 0. The product is given by

$$MN = \sum_{h=0}^{2k} \Big(\sum_{i+j=h} a_i b_j \Big) 10^h. \tag{22}$$

The coefficient of 10^h might be larger than 9, and so we would need to perform "carrying" to get the true base-10 expansion of MN.

Equation 22 looks quite a bit like convolution. To make the analogy formal, choose some $n > 2k$ and associate the sequences

$$A = (a_1, ..., a_k, 0, ..., 0), \qquad B = (b_1, ..., b_k, 0, ..., 0) \in \boldsymbol{C}^n$$

to M and N respectively. We then let $C = A * B = (c_1, ..., c_n)$. We have

$$MN = \sum_{i=1}^{n} c_i 10^i \tag{23}$$

It is important to take $n > 2k$ to keep the terms from wrapping around. From Equation 16, we have

$$C = \Phi^{-1}(\Phi(A)\Phi(B)) \tag{24}$$

If we take n to be a power of 2, we can use the fast Fourier transform. This makes the method work much faster than ordinary multiplication.