Fast Fourier Transform

E.H.F.

June, 2004

1 Fourier Analysis of Periodic Functions

1.1 Continuous Periodic Functions

Consider a complex continuous function $f : \mathbf{R} \to \mathbf{C}$ such that it is periodic: $f(x+L) = f(x)$. In standard Fourier analysis, this function can be expressed as a linear combination of complex exponentials

$$f(x) = \frac{1}{\sqrt{2\pi}} \sum_{k \in \mathcal{Z}} f_k e^{i\alpha_k x} \tag{1}$$

where $\mathcal{Z}$ denotes the set of all integers, $\alpha_k = 2\pi k/L$ and the Fourier coefficients are

$$f_k = \frac{1}{\sqrt{2\pi}} \int_0^L e^{-i\alpha_k x} f(x). \tag{2}$$

There are a countable number of Fourier coefficients due to the periodicity of $f(x)$. When we perform a numerical calculation, this function must first be discretized. Then it is important to understand Fourier analysis of discrete periodic functions.

1.2 Periodic Functions on the Integers

Let $f_j \in \mathbf{C}$ be a complex function for all integers j, and let this function be periodic: $f_j = f_{n+j}$. We want to express this function as a linear combination of complex exponentials $e^{i\alpha j}$. Since f_j is periodic, the complex exponentials must also be periodic, so then $e^{i\alpha j} = e^{i\alpha(n+j)}$ which implies that $\alpha n = 2\pi k$ for some integer k. We define

$$\alpha_k \equiv \frac{2\pi k}{n} \tag{3}$$

so then $e^{i\alpha_k j}$ indexed by k as a function of the integer j are the basis functions. Since $e^{i\alpha_k j} = e^{i\alpha_{n+k} j}$ for all j, there are only n linearly independent basis functions. We choose $e^{i\alpha_k j}$ for $k = 0, 1, \ldots, n-1$. Then for any f_j we have

$$f_j = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} \hat{f}_k e^{i\alpha_k j} \tag{4}$$

where $\hat{f}_k$ are the n Fourier coefficients. To determine these coefficients, we multiply both sides of the equation by $e^{-i\alpha_l j}/\sqrt{n}$ and sum over j:

$$\frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} f_j e^{-i\alpha_l j} = \frac{1}{n} \sum_{j,k=0}^{n-1} \hat{f}_k e^{i\alpha_k j} e^{-i\alpha_l j} \tag{5}$$

For the summation over j on the right side of this equation, we must evaluate

$$\frac{1}{n} \sum_{j=0}^{n-1} \exp\left(i \frac{2\pi}{n}(k-l)j\right) = \frac{1}{n} \sum_{j=0}^{n-1} e^{i\alpha_{k-l} j} \tag{6}$$

since $\alpha_k \equiv 2\pi k/n$. If we define

$$J_k \equiv \frac{1}{n} \sum_{j=0}^{n-1} e^{i\alpha_k j} \tag{7}$$

then the previous sum in Eqn. 6 is equivalent to J_{k-l}. First, consider J_0. Then each exponential has a value of 1, so then $J_0 = 1$. Now consider J_k for integers $k \neq 1$. We note that

$$J_k e^{i\alpha_k} = \sum_{j=0}^{n-1} e^{i\alpha_k j} e^{i\alpha_k} = \sum_{l=1}^{n} e^{i\alpha_k l} = \sum_{l=0}^{n-1} e^{i\alpha_k l} = J_k \tag{8}$$

To obtain the expression to the right of the second equality, we simply change variables so that $l = j+1$. For the expression to the right of the third equality, we use periodicity to note $e^{i\alpha_k n} = e^{i\alpha_k 0} = 1$. Since $e^{i\alpha_k} \neq 0$ for all integers $k \neq 0$ and $n > 0$, then $J_k = 0$ for all $k \neq 0$. Then

$$\frac{1}{n} \sum_{j=0}^{n-1} \exp\left(i \frac{2\pi}{n}(k-l)j\right) = J_{k-l} = \delta_{kl} \tag{9}$$

where δ_{kl} is the Kronecker delta function. A function f_j is a n component vector with complex entries, and the set of all functions f_j is a vector space

$\mathbf{C}^n$. If we define the inner product as the dot product of two vectors, then the basis functions $e^{-i\alpha_k j}/\sqrt{n}$ are orthonormal because of Eqn. 9. This implies that the Fourier coefficients are given by

$$\hat{f}_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} f_j e^{-i\alpha_k j} \tag{10}$$

where $\hat{f}_k$ is a complex number. One can think of $\hat{f}_k$ as a discrete Fourier transform of f_j. The information in the n complex numbers of f_j for $j = 0, \ldots, n-1$ can be represented as n different complex numbers in $\hat{f}_k$ for $k = 0, \ldots, n-1$.

1.3 Matrix Properties

The relationship between $\hat{f}_k$ and f_j is linear, so it is possible to obtain $\hat{f}_k$ from a matrix multiplication

$$\hat{f}_k = \sum_{j=0}^{n-1} F_{kj} f_j \tag{11}$$

where it is straightforward to verify that

$$F_{kj} = \frac{1}{\sqrt{n}} w_n^{kj} \ ; \ w_n \equiv e^{-i2\pi/n}. \tag{12}$$

Consider the conjugate transpose matrix of F_{kj}

$$F_{kj}^\dagger = \frac{1}{\sqrt{n}} \left(w_n^\dagger\right)^{kj} \ ; \ w_n^\dagger \equiv e^{i2\pi/n}. \tag{13}$$

Taking the matrix product of F_{kj} and its conjugate transpose, we obtain

$$\sum_{l=0}^{n-1} F_{kl} F_{lj}^\dagger = \frac{1}{n} \sum_{l=0}^{n-1} e^{i(\alpha_k - \alpha_j)l} = J_{k-j} = \delta_{kj} \tag{14}$$

so then the conjugate transpose is also the inverse of F_{kj}. This implies that F_{kj} is a unitary matrix and represents a length preserving rotation in the vector space $\mathbf{C}^n$.

2 Fast Fourier Transform

When we compute the Fourier transform of a continuous periodic function in a computer, we first discretize the function at n points and then perform a discrete Fourier transform. Computing $\hat{f}_k$ for each integer k requires n complex multiplications, and there are n components of $\hat{f}_k$, so the matrix multiplication takes n^2 complex multiplication operations. Fortunately, we can perform this matrix multiplication with fewer operations using an idea called recursion.

2.1 Recursion

The idea behind recursion is to break up a large computation into smaller computations of the same type. A recursive algorithm continues to break up the computations until it finds a computation that is trivial. These trivial computations are combined to give the result of the large computation.

The way this actually works in code is that a recursive functions calls itself. Pseudo code for a recursive algorithm *fftrecursive* looks like the following:

- if this is a trivial calculation, return the result of the trivial calculation
- if this not the trivial calculation,
  - break the calculation up into smaller calculations
  - perform these smaller calculations by calling *fftrecursive*
  - put the results of these smaller calculations together and return the result

A recursive algorithm has a tree type structure, since it branches off and branches off until it finds trivial calculations. Then it goes back up the tree putting together the results of smaller calculations until it finally returns the result of the original large calculation.

2.2 Breaking up the Fourier transform

To break up the large matrix multiplication into smaller matrix multiplications, we identify a certain property of $w_n \equiv e^{i2\pi/n}$: if $m = n/2$ then

$$w_m = w_n^2. \tag{15}$$

When we insert $m = n/2$ into $e^{i2\pi/m}$, we insert m into the denominator in the exponential and hence obtain the square of the $e^{i2\pi/n}$. The idea is to break up one matrix multiplication by an $n \times n$ matrix into two matrix multiplications by $(n/2) \times (n/2)$ matrices.

To show how to do this, we write down the original problem again as

$$\hat{f}_k^n = \sum_{j=0}^{n-1} F_{kj}^n f_j^n \tag{16}$$

where we add a superscript n to the names of the vectors and matrices so we can distinguish them from vectors and matrices of dimension $m = n/2$. Imagine making all the odd components of f_j zero and performing the summation over j in Eqn. 16. For a nonzero contribution to this sum, the term from F_{kj}^n differs from the term in F_{kj}^n from the previous nonzero contribution by a factor of w_n^2. The elements of F_{kj}^n that are important in the summation are also elements of F_{kj}^m or $F_{kj}^{n/2}$. If we define

$$f_j^{m,e} \equiv f_{2j}^n \ , \ f_j^{m,o} \equiv f_{2j+1}^n \tag{17}$$

for $j = 0, \ldots, m-1$ as the even and odd components of the vector f_j^n, then we could transform $f_j^{m,e}$ and $f_j^{m,o}$ by F_{kj}^m and put the result back together to obtain the result in Eqn. 16.

The formula for putting the results of the smaller $m \times m$ calculation together comes from the following computation:

$$\begin{aligned}
\hat{f}_k^n = \sum_{j=0}^{n-1} F_{kj}^n f_j^n &= \sum_{j=0}^{n-1} f_j^n w_n^{kj} n^{-1/2} \\
&= \left(\sum_{l=0}^{m-1} f_{2l}^n w_n^{k2l} + \sum_{l=0}^{m-1} f_{2l-1}^n w_n^{k(2l+1)} \right) n^{-1/2} \\
&= \left(\sum_{l=0}^{m-1} f_l^{m,e} w_m^{kl} + w_n^k \sum_{l=0}^{m-1} f_l^{m,o} w_m^{kl} \right) n^{-1/2}
\end{aligned}$$

Using the relationship $w_m^{kl} = \sqrt{m} F_{kl}^m$, we can rewrite the first half of the terms $k = 0, 1, \ldots, m-1$

$$\hat{f}_k^n = \sqrt{\frac{m}{n}} \sum_{l=0}^{m-1} F_{kl}^m f_l^{m,e} + w_n^k \sqrt{\frac{m}{n}} \sum_{l=0}^{m-1} F_{kl}^m f_l^{m,o} = 2^{-1/2} \left(\hat{f}_k^{m,e} + w_n^k \hat{f}_k^{m,o} \right). \tag{18}$$

For the remaining terms which we represent as $m+k$ for $k = 0, 1, \ldots, m-1$, we have

$$\hat{f}^n_{m+k} = \sum_{l=0}^{m-1} \left(f^{m,e}_l w_m^{l(m+k)} + w_n^{m+k} f^{m,o}_l w_m^{l(m+k)} \right) \tag{19}$$

for $k = 0, 1, \ldots, m-1$. Since $w_m^{ml} = 1$ for all l and $w_n^{m+k} = w_n^k w_n^{n/2} = w_n^k e^{-i\pi} = -w_n^k$, we obtain

$$\hat{f}^n_{m+k} = 2^{-1/2} \left(\hat{f}^{m,e}_k - w_n^k \hat{f}^{m,o}_k \right) \tag{20}$$

for $k = 0, 1, \ldots m-1$. So the result is

$$\hat{f}^n_k = 2^{-1/2} \left(\hat{f}^{m,e}_k + w_n^k \hat{f}^{m,o}_k \right) \tag{21}$$

$$\hat{f}^n_{m+k} = 2^{-1/2} \left(\hat{f}^{m,e}_k - w_n^k \hat{f}^{m,o}_k \right) \tag{22}$$

for $k = 0, 1, \ldots m-1$. The full result for an n dimensional matrix product can be constructed from two $m = n/2$ dimensional matrix products. Note that this reconstruction procedure requires an addition m complex multiplications to compute $w_n^k \hat{f}^{m,o}_k$.

2.3 Algorithm

The beauty of breaking down the matrix multiplications is that this procedure can be repeated. For simplicity, suppose the original dimension of the matrix is a power of 2, so $n = 2^l$. This process of splitting the computation in half happens l times; at this point, there are n discrete Fourier transforms of one complex number, which is simply the same complex number.

$$\hat{f}^1_0 = f^1_0 \tag{23}$$

This is the trivial calculation in the recursive algorithm. The full solution is constructed from these numbers by applying Eqn. 21 and 22 for $l-1$ remaining levels of the problem. Each of these levels requires $n/2$ operations, so then the number of operations is $(n/2)l \approx n \log_2 n$. As n gets large, this is a significant improvement from the n^2 operations required of a straightforward matrix multiplication.