Appendix K

The FFT

K.1. WHAT it is, WHERE it is used, and WHY.

The Fast Fourier Transform, or FFT, is an efficient algorithm for calculating the Discrete Fourier Transform, or DFT. What is the DFT, and why do we want to calculate it, efficiently or otherwise? There are actually many DFTs and corresponding FFTs. For a natural number N, the N-point DFT transforms a vector of N complex numbers, $\mathbf{f}$, into another vector of N complex numbers, $\hat{\mathbf{f}}$. We will usually suppress the explicit dependence of $\hat{\mathbf{f}}$ on N, except when we use a relationship between DFTs of different sizes to develop the FFT algorithm.

If the components of $\mathbf{f}$ are interpreted as the values of a signal at equally spaced points in space or time, the components of $\hat{\mathbf{f}}$ are the coefficients of components of $\mathbf{f}$ having N equally spaced frequencies, such as the level indicators on an audio-system equalizer. The reasons for wanting to compute $\hat{\mathbf{f}}$ are many and varied. One important reason is that, for the purpose of solving differential equations, the equally spaced values of the individual frequency components form an orthonormal basis

$$\mathbf{f}_k,\ k = 0, \dots, N-1$$

of eigenvectors for a large family of matrices that arise in approximating constant coefficient differential differential operators with $2\pi-$periodic boundary conditions. If we denote the grid-spacing by $h = \frac{2\pi}{N}$, put $i = \sqrt{-1}$, and if the components of the basis vector $\mathbf{f}_k$ are defined by

$$f_{k,j} := \exp(ikjh),\ j = 0, \dots, N-1, \tag{1}$$

then these vectors satisfy the orthogonality relations:

$$< \mathbf{f}_k, \mathbf{f}_l > := \frac{1}{N} \sum_{j=0}^{N-1} f_{k,j} \overline{f_{l,j}} = \delta_{k,l} \tag{2}$$

where

$$\delta_{k,l} = 0 \text{ for } k \neq l \text{ and } \delta_{k,l} = 1 \text{ for } k = l.$$

The eigenvector property referred to above can be expressed as:

$$\mathbf{A}\mathbf{f}_k = \lambda_{\mathbf{A},k}\, \mathbf{f}_k, \quad \lambda_{\mathbf{A},k} \in \mathbf{C} \tag{3}$$

where $\mathbf{A}$ is any matrix whose entries satisfy

$$a_{j,k} = a_{j+m \mod N, k+m \mod N}. \tag{4}$$

An example of such an $\mathbf{A}$ is the centered second difference approximation of the second derivative with periodic boundary conditions on a grid of N equally spaced points:

$$\frac{1}{h^2}(\mathbf{T} - 2\mathbf{I} + \mathbf{T}^{-1}), \; h = \frac{2\pi}{N}, \tag{5}$$

where $\mathbf{T}$ is the the cyclic translation of components

$$\mathbf{T}z_j = z_{(j+1 \mod N)}, \tag{6}$$

In other words, *using a fixed orthonormal basis (the Fourier basis) the DFT diagonalizes any operator on the discrete unit circle group that commutes with translations*. These properties of the Fourier bases $\{\mathbf{f}_k\}$ may be verified directly using the geometric difference identity

$$\mathbf{T}\mathbf{f}_k = e^{ikh}\mathbf{f}_k$$

and the geometric summation identity

$$\sum_{j=0}^{N-1} e^{i(k-l)jh} = \frac{e^{i(k-l)Nh} - 1}{e^{i(k-l)h} - 1} = \frac{e^{2\pi i(k-l)} - 1}{e^{i(k-l)h} - 1} = 0$$

on the discrete circle group. The latter may be seen as a consequence of the former, and in a companion appendix, we also show that both are specials cases of a beautiful and far-reaching construction of orthogonal bases of eigenvectors for functions on any abelian group ! For this reason, even those familiar with the Fourier theory should find the treatment there useful and enlightening. References are also provided to generalizations of the Fourier transforms to non-abelian groups and other spaces, such as Radon transforms used in computed tomography (CAT) scans.

Since we wish to define the N-point DFT so that

$$\mathbf{f} = \sum_{k=0}^{N-1} \hat{f}_k \mathbf{f}_k, \tag{6}$$

we can form the inner product of (6) with each basis vector and apply the orthogonality properties (2) to obtain the most straightforward definition of the DFT,

$$\hat{f}_k = \frac{1}{N} \sum_{j=0}^{N-1} f_j \overline{f_{k,j}}, \tag{7}$$

as a linear mapping: $\mathbf{f} \in \mathbf{C}^N \mapsto \hat{f} \in \mathbf{C}^N$. If we let

$$\omega = \omega_N = \exp(-ih) = e^{-2\pi i/N}, \tag{8}$$

denote a primitive Nth root of unity, and define the matrix

$$\mathbf{F} = \mathbf{F}_N = \{F_{kj}\} = \exp(-ikjh) = \omega^{kj},\ 0 \le j, k \le N-1, \tag{9}$$

then

$$\hat{f} = \frac{1}{N}\mathbf{F}\mathbf{f} \tag{10}$$

is the matrix form of the DFT in the standard basis for $\mathbf{C}^N$.

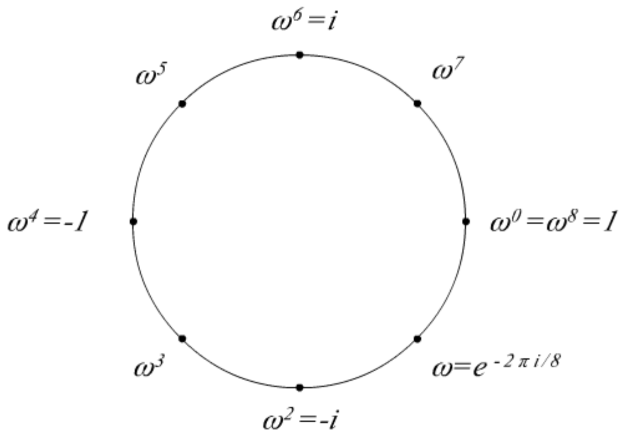

Figure F.1. The Conjugate Discrete Circle Group with $2^3 = 8$ elements.

While the sign convention we have used for the exponent in the definition of ω conveniently minimizes negative signs when working with $\mathbf{F}$, it is important to be aware that the opposite convention is also useful and common when focusing on the Fourier basis vectors rather then their conjugates, e.g., in the discussion of symmetric indexing below accompanying figure F.2.

The inverse discrete Fourier Transform, or IDFT is therefore

$$\hat{\mathbf{f}} \mapsto \mathbf{f} = \mathbf{F}^*\hat{\mathbf{f}} = \overline{\mathbf{F}\overline{\hat{\mathbf{f}}}}. \tag{10}$$

This shows that we can use the DFT (and therefore the FFT) to compute the inverse DFT just by changing $-i$ to i in the definition of ω in our implementation, by pre- and post-conjugating the input and output, or reversing the input modulo N, and omitting the post-normalization step. We may describe the inverse DFT of $\hat{\mathbf{h}}$ as "the pointwise values of the vector whose (DFT) coefficients are $\hat{\mathbf{h}}$", and the DFT of $\mathbf{h}$ as "the coefficients of the vector whose pointwise values are $\mathbf{h}$". In some conventions, the factor of $\frac{1}{N}$ in the DFT appears instead with the inverse transform. Alternatively, it can be divided into factors of $\frac{1}{\sqrt{N}}$ in each direction, which makes both transforms unitary (i.e., norms of vectors are preserved in both directions rather than amplified in one and diminished in the other.)

If we use the definition (7) directly, then the calculation of the DFT involves N^2 complex multiplications. When N is a power of 2, the FFT uses patterns in the components of $\mathbf{F}_N$ to reduce this to $\frac{1}{2}N\log_2 N$ complex multiplications and comparably many additions. For small values of N this is not a major saving, but when N is the number of points in a 128^3 grid for a three-dimensional problem, it becomes a huge saving—on the order of a billion for one transform—and that saving is repeated every second in numerous digital signal processing applications, literally around the world. Below are the first few examples of such $\mathbf{F}_N$. For $N = 8$, $\omega^2 = i$, and in general, $\omega^N = 1$ so $\omega^{jk} = \omega^{(jk \bmod N)}$. The matrix $\mathbf{F}_8$ can also be rewritten in a form that foreshadows the recursive relation among these matrices that is the heart of the FFT.

$$\mathbf{F}_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \; \mathbf{F}_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}$$

$$\mathbf{F}_8 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega & \omega^6 & \omega^3 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega \end{pmatrix}$$

To better understand the significance of both the eigenvector and orthogonality properties of the basis associated with the FFT, let $\mathbf{A}$ be an arbitrary $N \times N$ matrix and consider the computational effort that is contained in the symbolic formulas for solutions of the differential equation IVP

$$\frac{d\mathbf{u}}{dt} = \mathbf{A}\mathbf{u}, \;\; \mathbf{u}(0) = \mathbf{f}, \tag{11}$$

and its approximation using Euler's method,

$$\mathbf{u}_{m+1} = (\mathbf{I} + \Delta t\mathbf{A})\mathbf{u}_m, \;\; \mathbf{u}_0 = \mathbf{f}. \tag{12}$$

The solution of the differential equation is given by:

$$\mathbf{u}(t) = \exp(t\mathbf{A})\mathbf{f} = \sum_{m=0}^{\infty} \frac{(tA)^m}{m!}, \tag{13}$$

which can be computed by calculating a sufficient number of terms in the exponential series to achieve the desired accuracy. Recall that each matrix-matrix product requires $O(N^3)$ multiplications per term:

$$\frac{1}{m}(t\mathbf{A})((t\mathbf{A})^{m-1}). \tag{14}$$

The saving that might be obtained by computing only matrix-vector products $(O(N^3))$ would be beneficial only if we were interested in one particular initial vector $\mathbf{f}$. Directly, the solution of the Euler approximation is given by

$$\mathbf{u}_m = (\mathbf{I} + \Delta t\mathbf{A})^m\mathbf{f}. \tag{15}$$

We can explicitly calculate $(\mathbf{I}+\Delta t\mathbf{A})^m$ with mN^3 complex multiplications, or mN^2 to calculate $(\mathbf{I}+\Delta t\mathbf{A})^m\mathbf{f}$ for a particular $\mathbf{f}$. But it is far more efficient and informative to compute the powers and exponential of $\mathbf{A}$ using a basis of eigenvectors and generalized eigenvectors that is guaranteed to exist. For our current purposes, we suppose $\mathbf{A}$ has N linearly independent eigenvectors $\mathbf{e}_0, \dots, \mathbf{e}_{N-1}$, so that if $\mathbf{S}$ is the $N \times N$ matrix whose kth column is $\mathbf{e}_k$, then

$$\mathbf{AS} = \mathbf{S}\Lambda.$$

where

$$\Lambda = \mathrm{diag}\{\lambda_1, \dots, \lambda_N\}.$$

To facilitate the analogy with the DFT, we write $\tilde{\mathbf{u}} = \mathbf{S}^{-1}\mathbf{u}$, so that $\mathbf{u} = \mathbf{S}\tilde{\mathbf{u}}$ expresses $\mathbf{u}$ as a linear combination of eigenvectors, with the components of $\tilde{\mathbf{u}}$ as coefficients. With this notation, $\widetilde{\mathbf{Au}} = \Lambda\tilde{\mathbf{u}}$, so the differential equation becomes

$$\frac{d\tilde{\mathbf{u}}}{dt} = \Lambda\tilde{\mathbf{u}},\ \ \tilde{\mathbf{u}}(0) = \tilde{\mathbf{f}}, \tag{16}$$

and its Euler's method approximation,

$$\tilde{\mathbf{u}}_{m+1} = (\mathbf{I}+\Delta t\Lambda)\tilde{\mathbf{u}}_m,\ \ \tilde{\mathbf{u}}_0 = \tilde{\mathbf{f}}. \tag{17}$$

In this form, the solution of the differential equation becomes

$$\tilde{\mathbf{u}}(t) = \exp(t\Lambda)\tilde{\mathbf{f}}, \tag{18}$$

whose calculation only requires N complex exponentials and N complex multiplications, and the solution of the Euler approximation is

$$\tilde{\mathbf{u}}_m = (\mathbf{I}+\Delta t\Lambda)^m\tilde{\mathbf{f}}, \tag{19}$$

whose calculation only requires N complex powers and N complex multiplications. To express the solution of our equations in terms of the original standard basis, we form linear combinations of the basis vectors: $\mathbf{u}(t) = \mathbf{S}\tilde{\mathbf{u}}(t)$ and $\mathbf{u}_m = \mathbf{S}\tilde{\mathbf{u}_m}$.

What are the hidden costs of these formulas? For a general $\mathbf{S}$, it requires $O(N^3)$ to compute $\tilde{\mathbf{u}}$, either by finding $\mathbf{S}^{-1}$ explicitly, or by finding the $\mathbf{LU}$ decomposition of $\mathbf{S}$ in order to solve $\mathbf{u} = \mathbf{S}\tilde{\mathbf{u}}$. Once these have been computed however, each additional transform only requires $O(N^2)$ arithmetic operations. But when the eigenvectors that form the columns of $\mathbf{S}$ are orthogonal, the $O(N^3)$ initial step is unnecessary since, up to a scaling, we only need to transpose and conjugate $\mathbf{S}$ to find $\mathbf{S}^{-1}$. The final step, converting back to the original basis, involves the matrix-vector product $\mathbf{S}\tilde{\mathbf{u}}$, requiring $O(N^2)$ complex multiplications, even for a non-orthogonal $\mathbf{S}$.

If $\mathbf{A}$ is any matrix that satisfies (4) (i.e., any matrix that commutes with the group of translations generated by the $\mathbf{T}$ of (6)), then the DFT is a particular example of the above construction, with $\mathbf{S}^{-1} = \frac{1}{N}\mathbf{F}_N$. In other words, the matrix $\mathbf{S} = \mathbf{F}_N^*$ diagonalizes $\mathbf{A}$, $\mathbf{S}^*\mathbf{S} = N\mathbf{I}$ (so $\frac{1}{\sqrt{N}}\mathbf{S}$ is unitary), and for this $\mathbf{S}$, $\tilde{\mathbf{u}} = \hat{\mathbf{u}}$. The inversion is just a consequence of the convention of defining S as the matrix whose columns are the eigenbasis, rather than its inverse, or equivalently, defining $\mathbf{F}_N$ as the matrix whose rows are the conjugates of the eigenbasis, rather than its inverse. From the above considerations, we see that even without the FFT, there are several ways in which the DFT substantially reduces the complexity of solving equations involving the matrices $\mathbf{A}$ that it diagonalizes,

The availability of explicit formulas for an orthonormal basis of eigenvectors whose conjugates make up the rows of the DFT matrix, immediately reduces the considerable computational effort that would otherwise be required to find eigenvalues and eigenvectors to the much simpler one of calculating $O(N^2)$ complex exponentials. Then, in addition, orthogonality reduces from $O(N^3)$ to $O(N^2)$ the number of arithmetic operations needed in constructing the transform that implements the change of basis to the eigenvector basis. In summary, the existence of the orthonormal eigenvectors makes it possible to compute useful functions of the matrix using only $O(N^2)$ scalar evaluations of the same functions.

The FFT takes a further simplification, eliminating the $O(N^2)$ steps from the process, and reducing them, and the overall complexity, to $O(N \log N)$. As we will see below, this is because the FFT even makes it unnecessary to explicitly compute the discrete Fourier basis vectors and the N^2 entries of $\mathbf{F}_N$. If this were necessary, it would involve calculating N^2 complex exponentials. A clue to the possibility of the FFT is the fact that $\mathbf{F}_N$ only contains $N/2$ distinct components, up to sign, the maximum number of complex exponentials that are needed to compute the N-point DFT-FFT. Indeed, no trigonometric or complex exponential evaluations should be required at all, since these values can be computed efficiently using only square roots and division by repeated bisection and normalization of edges 2^m-gons inscribed in the unit circle, starting with the square with vertices at the standard unit vectors! It makes sense to compute these values once and for all outside any FFT implementation, and store them for lookup each time they are needed in every call of the FFT.

A related application of the DFT and FFT is for performing fast convolution. If we let $\mathbf{z}$ be the vector in $\mathbf{C}^N$ whose components are

$$z_j = e^{ijh},\ j = 0, \dots, N-1,$$

then we can view the DFTs and discrete Fourier representation of vectors $\mathbf{f}$ and $\mathbf{g}$ as discrete polynomial coefficients and expansions,

$$\mathbf{f} = \sum_{k=0}^{N-1} \hat{f}_k \mathbf{z}^k$$

and

$$\mathbf{g} = \sum_{k=0}^{N-1} \hat{g}_k \mathbf{z}^k.$$

Just as we multiply ordinary polynomials by convolution of their coefficients, the expansion of the pointwise product is

$$\mathbf{h} = \{h_j = f_j g_j\} = \sum_{k=0}^{N-1} \hat{h}_k \mathbf{z}^k,$$

where

$$\hat{h}_k = \sum_{l+m=k \text{ mod } N} \hat{f}_l \hat{g}_m \tag{18}$$

We call the expression on the right the cyclic (or circular) convolution of $\hat{\mathbf{f}}$ and $\hat{\mathbf{g}}$, and denote and rewrite it as

$$\hat{\mathbf{f}} *_N \hat{\mathbf{g}} := \sum_{l=0}^{N-1} \hat{f}_l \hat{g}_{(k-l \text{ mod } N)}. \tag{19}$$

Therefore, the inverse DFT of $\hat{\mathbf{h}} = \hat{\mathbf{f}} *_N \hat{\mathbf{g}}$ is $\mathbf{h} = \mathbf{fg}$, Applying the DFT to both sides of this statement, we may write

$$(\mathbf{fg})\hat{} = \hat{\mathbf{f}} *_N \hat{\mathbf{g}}. \tag{20}$$

Given the similarity between the forward and inverse DFT, with minor modification we can check that

$$(\mathbf{f} *_N \mathbf{g})\hat{} = N\,\hat{\mathbf{f}}\hat{\mathbf{g}} \tag{21}$$

and we see that the DFT changes multiplication to convolution and vice versa. Whether the factor N appears in (20) or in (21) depends on whether the normalization factor $\frac{1}{N}$ occurs in the definition of the forward or inverse DFT.

Since the circular convolution operation is bilinear and the 'delta function' $< 1, 0, \dots, 0 >$ acts as an identity for it, the solutions of the difference equations above can be represented as circular convolutions of the given initial data with a 'fundamental solution' whose initial data is a 'delta function'. In this way, the acceleration of computing solutions discussed above can be viewed as using the FFT as a tool for performing 'fast convolution'. Instead of performing straightforward convolution, a task that requires N^2 multiplications, we can transform, multiply the tranforms pointwise, and inverse transfom, to reduce the number of multiplications to $N(1 + \log_2 N)$.

Uses for the fast convolution go far beyond accelerating the solution of differential equations. For example, it can be used to reduce the cost of multiplying two numbers with millions of digits from trillions of individual digit multiplications back to millions. But we will leave such applications to other treatments.

Before proceeding with the FFT itself, there is a final point that deserves mention; namely the intimate relation between the Fourier transform coefficient $\hat{f}(k)$ of a continuous 2π-periodic function f and the discrete Fourier transform coefficient $\mathbf{f}_k$ of the the vector $\mathbf{f}$ obtained by sampling $f(x)$ at the N equi-spaced points, $x_j = jh$; $h = 2\pi/N,\ j = 0, \dots, N-1$. If we compare their definitions:

$$\hat{f}(k) = \frac{1}{2\pi}\int_0^{2\pi} f(x)e^{-ikx}\,dx \tag{22}$$

and (7),

$$\mathbf{f}_k = \frac{1}{N}\sum_{j=0}^{N-1} f(x_j)e^{-ikx_j},$$

and use the periodicity of f, we see that the sum that gives $\mathbf{f}_k$ is precisely the Trapezoidal Method approximation of the integral that gives $\hat{f}(k)$. Now in general, the error $E_n(f,a,b)$ in the N-subinterval Trapezoidal approximation of an integral $\int_a^b f(x)\,dx$ satisfies the estimate (see, e.g., [1]):

$$E_n(f,a,b) = -\frac{(b-a)^3}{12N^2} f''(\xi) \tag{23}$$

for some $\xi \in (a,b)$, and by constructing a Riemann sum from this estimate, we also get the asymptotic error estimate

$$E_n(f,a,b) \sim -\frac{(b-a)^2}{12N^2}(f'(b) - f'(a)) + O(N^{-3}). \tag{24}$$

When f is periodic, the leading order term vanishes, and in fact the Euler-Maclaurin summation formula can be used to show that the next, and all higher order terms also have coefficients that vanish for periodic f. **In other words, as the number of discretization points, N, tends to infinity, the difference between a continuous Fourier coefficient and the corresponding discrete Fourier coefficient tend to zero faster than any negative power of N.** This order of convergence is often called "spectral accuracy", or informally, "infinite-order accuracy".

Another approach to this comparison can be obtained by considering $f(x)$ as the sum of its convergent Fourier series:

$$f(x) = \sum_{n=-\infty}^{\infty} \hat{f}(n) e^{inx},$$

and again comparing the definitions of $\hat{f}(k)$ and $\mathbf{f}_k$:

$$\hat{f}(k) = \frac{1}{2\pi}\int_0^{2\pi} \left(\sum_{n=-\infty}^{\infty} \hat{f}(n) e^{inx} \right) e^{-ikx}\,dx \tag{25}$$

$$\mathbf{f}_k = \frac{1}{N}\sum_{j=0}^{N-1} \left(\sum_{n=-\infty}^{\infty} \hat{f}(n) e^{inx_j} \right) e^{-ikx_j}. \tag{26}$$

The observation that allows us to relate these expressions is that if the infinite sum is replaced by a finite sum

$$f_N(x) = \sum_{n>-N/2}^{n\leq N/2} \hat{f}(n) e^{inx},$$

they become identical, simply because the given range is a different form of the indices $(0,\dots,N-1 \mod N)$ for the discrete Fourier basis. Thus, the only difference arises from the contributions from n outside of this range. Components corresponding to those n contribute nothing to the continuous Fourier coefficient, since when $|n|>N$ and $|k|\le N$, $k\neq n$ and e^{inx} is orthogonal to e^{ikx}. However, for $|n|>N$, when e^{inx} is sampled on the N point grid x_j, the resulting vector is *not* orthogonal to the vectors discrete Fourier basis vectors that are obtained by sampling e^{ikx} on that grid. In fact it is identical to one of them ! This is because if $k=n+mN=n \mod N$, then using $Nh=2\pi$, we have $e^{ikx_j}=e^{i(n+mN)jh}=e^{injh}e^{2\pi im}=e^{inx_j}$, a relation known as *aliasing* . We say that a component of f with n beyond the resolution of the grid is 'aliased' onto the grid as a contribution to the component within the resolution of the grid whose index is congruent to the original component modulo N. So, where does the "infinite order accuracy" observed using the Trapezoidal approximation point of view come from with this approach? We have shown in the accompanying appendix on Fourier analysis, that the magnitude of Fourier components $\hat{f}(n)$ of a smooth function f also tend to zero faster than any power of n, and from this it is not difficult to show that the aliasing error arising from all components whose indices exceed N also decays faster than any power of N. That is, since the individual Fourier coefficient amplitudes of a smooth function decay faster than any negative power of the frequency, i.e.,

$$|\hat{f}(n\pm mN)|\le C|n+mN|^{-p}\le CN^{-p}m^{-p},\ m=1,2,\dots \tag{27}$$

then so does the sum of all of these magnitudes outside any frequency interval:

$$\sum_{m=1}^{\infty}|\hat{f}(n\pm mN)|\le C'N^{-p}. \tag{28}$$

The fact that the finite sum

$$f_N(x)=\sum_{n>-N/2}^{n\le N/2}\hat{f}(n)e^{inx}, \tag{29}$$

can be reconstructed exactly from its samples at the N equi-spaced points (with spacing $h=2\pi/N$), $x_j=jh,\ j=0,\dots,N-1$, is an almost trivial special case of the so-called sampling theorem that goes by many names including Whittaker, Shannon, Nyquist, Kotel'nikov, Raabe, and Someya. A modern version states that if f is a smooth function that is "band-limited", in the sense that its Fourier transform is supported on the open interval $(-K, K)$, then f can be reconstructed exactly from its samples at equally spaced points $f(mT)$, $m \in \mathbf{Z}$, where $T = \pi/K$ is called the sampling interval and thus $f_s := K/\pi$ is the sampling rate. The bound on the frequency support $K = \pi f_s$ is often called the Nyquist frequency. For our finite sum, we check that $T = \pi/(N/2) = 2\pi/N$, exactly our h above. There are many refinements of this result, among them the case when the band limit is not strict and frequencies at the band limit are present.

The sampling theorem not only gives a relation between the support bound (band limit) and the sampling interval that permits exact reconstruction from equi-spaced samples, it also provides an algorithm for performing this reconstruction. For the finite Fourier series case, changing the order of summation of the forward and inverse discrete Fourier transform yields

$$f(x) = \sum_{j=0}^{N-1} f(x_j)\left(\frac{1}{N}\sum_{k=0}^{N-1} e^{ik(x-x_j)}\right) = \sum_{j>-N/2}^{j\le N/2} f(x_j)\delta_N(x-x_j)h \tag{30}$$

where as usual, $h = \frac{2\pi}{N}$ and $x_j = jh$, and

$$\delta_N(x) = \frac{1}{2\pi}\sum_{n>-N/2}^{n\le N/2} e^{ikx}. \tag{31}$$

We can see this as the discrete approximation of the fact that the "$\delta-$function" (evaluation at $x = 0$) is the identity with respect to convolutions, i.e., $f = f * \delta$, with δ_N a 'delta-sequence' that gives such an approximation of the identity. In the continuous case, the reconstruction aspect of the sampling theorem states that if

$$f(t) = \int_{|k|<K} \hat{f}(k)e^{ikt}\, dk \tag{32}$$

then

$$f(t) = \sum_{j=-\infty}^{\infty} f(t_j)S(K(t-t_j)) \tag{33}$$

where $h = \frac{\pi}{K}$, $t_j = jh$, and

$$S(t) = \text{sinc}(t) := \frac{\sin(t)}{t} \text{ for } t \neq 0 \text{ and } \text{sinc}(0) = 1. \tag{34}$$

The function S is known as the Whittaker cardinal function (or just the 'sinc'-function). The analogy with a Lagrange interpolation polynomial is apparent, since in addition to $\text{sinc}(0) = 1$, $\text{sinc}(n) = 0$ for any nonzero integer n. The fact that $\sum_{j=-\infty}^{\infty} f(t_j)S(K(t-t_j))$ interpolates f at each t_j is immediate from this. Of course there are infinitely many smooth functions that equal S on the integers. What is special among such functions about S is the fact that it is band limited, and in particular, it is the Fourier transform of the cutoff function

$$\hat{S}(k) = 1 \text{ for } |k| \leq \pi, \text{ and } \hat{S}(k) = 0 \text{ for } |k| > \pi. \tag{35}$$

Since the shifts $t - t_j$ correspond to phase shifts of Fourier coefficients, the functions $S(K(t-t_j))$ are uniformly band limited, as is any linear combination. By subtraction, we can state the samping theorem in the following form: *If a function vanishes on an infinite set of equispaced points, then a sufficient condition for it to be zero everywhere is that it is band limited with a bandwidth determined by interpreting the spacing of the points as a sampling frequency.*

To prove the sampling theorem, the band-limited transform $\hat{\mathbf{f}}(k)$ may be extended periodically to $\hat{\mathbf{f}}_P(k)$, then expanded as the sum of the resulting Fourier series, and the result must be identical the the (non-periodic) Fourier representation of $\hat{\mathbf{f}}(k) = \hat{S}(k)\hat{\mathbf{f}}_P(k)$ on the line. Since Fourier transform take multiplication to convolution, and vice versa, by interchanging an integration and a summation in analogy with the interchange of summations in the discrete case we can show the equivalence of the band-limited Fourier transform product

$$\int_{|k|<K} \hat{f}(k)e^{ikt}\, dk = \int_{-\infty}^{+\infty} \hat{S}(\frac{k\pi}{K})\hat{f}(k)e^{ikt}\, dk. \tag{36}$$

and the sampling convolution expansion (33).

The analogy with Lagrange polynomials mentioned above can even be extended to the (infinite) product representation

$$\text{sinc}(x) = \Pi_{n\in\mathbf{Z},\ n\neq 0}(1 - \frac{x^2}{n^2}). \tag{36}$$

Formally, the product on the right is also zero at each non-zero integer n, and one at zero. Also formally, the coefficient of x^2 on the right is

$-\sum_{n=1}^{\infty}\frac{1}{n^2}$ while the corresponding coefficient in the Taylor Maclaurin series of $\frac{1}{\pi x}\sin(\pi x)$ is $-\frac{1}{6\pi x}(\pi x)^3 = -\frac{\pi^2}{6}$. These observations can be made rigorous to show that $\sum_{n=1}^{\infty}\frac{1}{n^2} = \frac{\pi^2}{6}$, e.g., in [Marshall].

We conclude this section with a 'footnote' on the indexing of N frequencies or grid points in the form $-N/2 \leq N/2$ that is symmetric about zero, for which there are two cases. When N odd, equality is never satisfied at the upper limit, but when N is even, it is. In contrast with the asymmetric indexing from 0 to $N-1$, symmetric indexing possesses the favorable property that the mapping from the discrete basis vectors e^{inx_j} to the corresponding continuous basis functions e^{inx} is close under taking the real and imaginary parts. For the same reason, the symmetric indexing is the correct one to use when associating a discrete basis vector with an eigenvalue of a linear operator that it diagonalizes. For instance, when using a spectral-Fourier method to approximate the second derivative operator $\mathbf{A} = \frac{d^2}{dx^2}$ for which $\mathbf{A}e^{ikx} = -k^2 e^{ikx}$, and obtaining a discrete basis $\mathbf{f}_k$ by sampling e^{ikx} at the $N = 2^J$ points $x_j = 2\pi j/N$, both $e^{i(N-1)x}$ and $e^{i(-1)x}$ are represented by the same $\mathbf{f}_{N-1} = \mathbf{f}_{-1}$. So couldn't one assign it either the eigenvalue $\lambda_{N-1} = -(N-1)^2$, or $\lambda_{-1} = -(-1)^2$? But for the approximation to converge to the analytical solution, not skipping low frequency modes, the $N-1$ mode must patiently wait its turn which will come as the number of discretization points is increased.

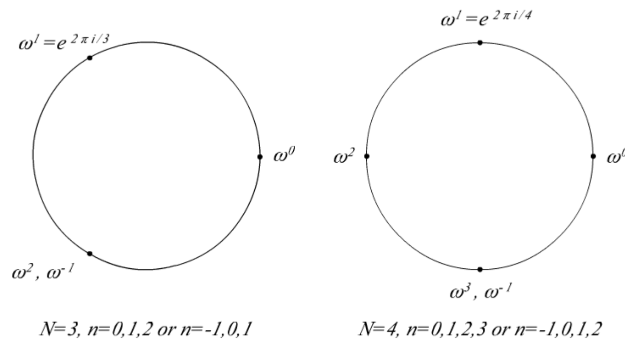
$\omega^1 = e^{2\pi i/3}$
ω^0
$\omega^2,\ \omega^{-1}$
$N=3,\ n=0,1,2$ or $n=-1,0,1$
$\omega^1 = e^{2\pi i/4}$
ω^2
ω^0
$\omega^3,\ \omega^{-1}$
$N=4,\ n=0,1,2,3$ or $n=-1,0,1,2$

Figure F.2. Indexing Symmetric and Asymmetric About $n = 0$.

$\triangleright$ **Exercise K–1.** What are the eigenvalues and eigenvectors *of* the DFT, and how are they related to eigenvalues and eigenvectors of the Fourier Transform? (See the references for discussions of this topic.)

K.2. How The FFT is Computed

The most popular and easiest to understand family of N-point FFTs are those for which N is a power of 2, say $N = 2^\ell$. We will restrict our attention to this case, and refer the reader to other treatments for variations on the FFT for general N, data that are real, even/odd symmetric, etc. In the case of interest, the algorithm is based on redundancy in the calculation of the coefficients with $k < N/2$, and those with $k \geq N/2$. This is expressed in relations among the coefficients of the matrix $\mathbf{F}_N$, involving those that multiply the components of $\mathbf{f}$ with even indices, and those that multiply the components with odd indices. In terms of the contributions from the components of $\mathbf{f}$ with even and odd indices,

$$\begin{aligned} f^e_j &:= f_{2j},\ j = 0, \ldots, \frac{N}{2} - 1, \\ f^o_j &:= f_{2j+1},\ j = 0, \ldots, \frac{N}{2} - 1. \end{aligned}$$

The pattern is expressed in the following identities:

$$\begin{aligned} \hat{f}_k &= \hat{f^e}_k + e^{-ikh}\hat{f^o}_k \\ \hat{f}_{k+N/2} &= \hat{f^e}_k - e^{-ikh}\hat{f^o}_k \end{aligned} \tag{1}$$

where $h = 2\pi/N$ and the transforms on the right hand side are $N/2$ point transforms, Collectively, these two computations are known as the butterfly, and the e^{-ikh} factors are known as 'twiddle' factors. In block matrix form in terms of the matrix $\mathbf{F}_N$,

$$\mathbf{F}_N\mathbf{P}_N = \begin{pmatrix} \mathbf{F}_{N/2} & +\mathbf{D}_N\mathbf{F}_{N/2} \\ \mathbf{F}_{N/2} & -\mathbf{D}_N\mathbf{F}_{N/2} \end{pmatrix}. \tag{2}$$

where $\mathbf{D}_N$ is a diagonal $N/2 \times N/2$ matrix of 'twiddle factors'

$$\mathbf{D}_N = \mathrm{diag}\{1, \omega, \ldots, \omega^{\frac{N}{2}-1}\}, \tag{3}$$

and right multiplication by the $N \times N$ permutation matrix

$$\mathbf{P}_N = \{p_{2i,i} = 1,\ 0 \leq i < N/2,\ p_{2(i-N/2)+1,i} = 1,\ N/2 \leq i < N\}, \tag{4}$$

permutes all columns of $\mathbf{F}_N$ with even indices in order ahead of columns with odd indices. Here it is in gory detail for the case $N = 8$:

$$\mathbf{F}_8\mathbf{P}_8 =$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & -i & \omega^3 & -1 & -\omega & i & -\omega^3 \\ 1 & \omega^2 & -1 & -\omega^2 & 1 & \omega^2 & -1 & -\omega^2 \\ 1 & \omega^3 & i & \omega & -1 & -\omega^3 & -i & -\omega \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -\omega & -i & -\omega^3 & -1 & \omega & i & \omega^3 \\ 1 & -\omega^2 & -1 & \omega^2 & 1 & -\omega^2 & -1 & \omega^2 \\ 1 & -\omega^3 & i & -\omega & -1 & \omega^3 & -i & \omega \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix} & \begin{pmatrix} 1 & 1 & 1 & 1 \\ \omega & \omega^3 & -\omega & -\omega^3 \\ \omega^2 & -\omega^2 & \omega^2 & -\omega^2 \\ \omega^3 & \omega & -\omega^3 & -\omega \end{pmatrix} \\ \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix} & \begin{pmatrix} -1 & -1 & -1 & -1 \\ -\omega & -\omega^3 & \omega & \omega^3 \\ -\omega^2 & \omega^2 & -\omega^2 & \omega^2 \\ -\omega^3 & -\omega & \omega^3 & \omega \end{pmatrix} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ \omega & \omega^3 & -\omega & -\omega^3 \\ \omega^2 & -\omega^2 & \omega^2 & -\omega^2 \\ \omega^3 & \omega & -\omega^3 & -\omega \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \omega & 0 & 0 \\ 0 & 0 & \omega^2 & 0 \\ 0 & 0 & 0 & \omega^3 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}$$
$$= \mathbf{D}_8\mathbf{F}_4$$
$$\begin{pmatrix} -1 & -1 & -1 & -1 \\ -\omega & -\omega^3 & \omega & \omega^3 \\ -\omega^2 & \omega^2 & -\omega^2 & \omega^2 \\ -\omega^3 & -\omega & \omega^3 & \omega \end{pmatrix} = - \begin{pmatrix} 1 & 1 & 1 & 1 \\ \omega & \omega^3 & -\omega & -\omega^3 \\ \omega^2 & -\omega^2 & \omega^2 & -\omega^2 \\ \omega^3 & \omega & -\omega^3 & -\omega \end{pmatrix} = -\mathbf{D}_8\mathbf{F}_4$$

The permutation can be inverted to write $\mathbf{F}_N$ as the block matrix following $\mathbf{P}_N^{-1}$, where $\mathbf{P}_N^{-1}$ permutes the components of the input vector $\mathbf{f}$ in the same way as $\mathbf{P}_N$ permutes the columns of $\mathbf{F}_N$, placing those with even indices in order ahead of those with odd indices:

$$\mathbf{P}_N^{-1} = \begin{pmatrix} 1 & 0 & 0 & \dots & & & \\ 0 & 0 & 1 & 0 & 0 & \dots & \\ 0 & 0 & 0 & 0 & 1 & 0 & \dots \\ \vdots & & & & & & \\ 0 & 1 & 0 & 0 & 0 & \dots & \\ 0 & 0 & 0 & 1 & 0 & \dots & \\ 0 & 0 & 0 & 0 & 0 & 1 & 0\dots \\ \vdots & & & & & & \end{pmatrix}$$

To confirm (1)–(4), we first check that the second half of each evenly indexed column of $\mathbf{F}_N$ repeats the first half. Using $\omega^N = 1$ we check that for column indices $2l$, $l = 0, \dots, N/2-1$, and row indices $k = 0, \dots, N/2-1$,

$$f_{k+N/2,2l} = \omega^{(k+N/2)(2l)} = \omega^{2kl+Nl} = \omega^{k(2l)} = f_{k,2l}. \tag{5}$$

Since $h = 2\pi/N$,

$$\omega_N^2 = e^{-i(2h)} = e^{-i(2(2\pi/N))} = e^{-i(2\pi/(N/2))} = \omega_{N/2} \tag{6}$$

and therefore these $N/2 \times N/2$ components $f_{k,2l}$ of these repeated half-columns are exactly the k, l entries of $\mathbf{F}_{N/2}$! This repetition of the same matrix multiplying the same components of $\mathbf{f}$ already saves us half of the work of computing half of the components of $\hat{\mathbf{f}}$.

A similar saving can be found when calculating the contributions from components of $\mathbf{f}$ with odd indices. The second half of the odd index columns of $\mathbf{F}_N$ repeat the first half except for a change of sign.

$$\begin{aligned} f_{k+N/2,2l+1} &= \omega^{(k+N/2)(2l+1)} = \omega^{k(2l+1)}\omega^{N/2(2l+1)} \\ &= f_{k,2l+1}\omega^{Nl+N/2} = f_{k,2l+1}\omega^{N/2} \\ &= -f_{k,2l+1}. \end{aligned} \tag{7}$$

And while they are not simply copies of $\mathbf{F}_{N/2}$, they only differ from the even columns that are by the diagonal matrix $\mathbf{D}_N$.

$$f_{k,2l+1} = \omega^{k(2l+1)} = \omega^{2kl}\omega^k = \omega^k f_{k,2l}. \tag{8}$$

The pattern $(1,2)$ that is at the heart of the FFT is known as the butterfly.

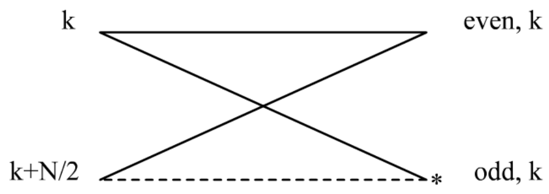

Figure F.2. The Butterfly Operation.

In chaotic dynamics, the butterfly represents the small change in initial conditions to which a system is sensitive, but here it refers to the crisscross nature of the calculation expressed by $(1,2)$. In Figure F.2, The positive contributions to the k and $k+N/2$ components of the N-point transform from the k component of the $N/2$-point transform of the components with even indices are represented by solid lines. Forming the product of the twiddle factor and the k component of the $N/2$-point transform of the components with odd indices is indicated by the asterisk (*). The positive and negative contributions of the result to the k and $k+N/2$ components of the N-point transform are indicated by solid and dashed lines, respectively.

As a consequence of $(1,2)$, *all* of the multiplications that are necessary to compute the components of $\mathbf{F}_N\mathbf{f}$ for $k \geq N/2$ have already been performed in computing the components for $k < N/2$. Furthermore, these multiplications come in the form of two DFTs with half the number or points, plus $N/2$ multiplications by the scalars in the two $\mathbf{D}_N$ matrices. Because of this, and the fact that N is a power of 2, the same decomposition may be performed to evaluate those DFTs more efficiently, and so on recursively on the decreasing sequence of half-size DFTs. In terms of number of multiplications, if $M(\ell)$ is the number of multiplications required for an FFT with $N = 2^\ell$ points, the butterfly structure shows that

$$M(\ell) = 2 * M(\ell - 1) + 2^{\ell-1}. \tag{9}$$

Since no multiplications are required to perform a two point DFT,

$$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \end{pmatrix} = \begin{pmatrix} u_0 + u_1 \\ u_0 - u_1 \end{pmatrix},$$

$M(1) = 0$, and we can confirm that

$$\begin{aligned} M(2) &= 2(0) + 2 = \ 2 = 1(2^1), \\ M(3) &= 2(2) + 4 = \ 8 = 2(2^2), \\ M(3) &= 2(8) + 8 = 24 = 3(2^3). \end{aligned}$$

In general, we can prove by induction on ℓ that

$$M(\ell) = \ell 2^{\ell-1} = (N/2)\log_2(N).$$

We have already checked several initial cases. To confirm the induction step, we check that the induction hypothesis assumption for the case $\ell - 1$

$$M(\ell-1) = (\ell-1)2^{\ell-2}$$

together with (2) imply the case ℓ:

$$M(\ell) = 2(\ell-1)2^{\ell-2} + 2^{\ell-1} = (\ell-1)2^{\ell-1} + 2^{\ell-1} = \ell 2^{\ell-1}.$$

There is approximately one addition and one subtraction associated with each twiddle factor multiplication, so there are about $\ell 2^\ell = N\log_2(N)$ additions required in an FFT of order N. When N is large, this represents a spectacular saving in comparison to the N^2 multiplications required without the FFT. For example, when $N = 2^{20} = 1,048,576$, the ratio is on the order or fifty thousand times fewer operations!

Another way to see this result is to note that ultimately, the *only* multiplications ever performed are the multiplications by the nonzero twiddle factor components of the $\mathbf{D}$ matrices at each stage. At the $\mathbf{F}_N$ stage, there are $2 * N/2$ such multiplications to combine two transforms of size $N/2$. But each of these transforms are obtained by combining two transforms of size $N/4$, so altogether four transforms of size $N/4$, so again N twiddle factor multiplications. After that, there are eight transforms of sizes $N/8$, and so on, for a total of N twiddle factor multiplications over $\log_2(N) - 1$ levels, since no multiplications are required at the $N = 2, N = 1$ level.

In practice, this is implemented by first performing the collective permutation of components to start with $N/2$ two-point DFTs, then combine these simultaneously to take advantage of common twiddle factors into $N/4$ four point transforms, and so on up the chain to the final combination of two $N/2$ point transforms into the single N point transform, $\mathbf{F}_N$. If the indices of the components are represented in binary, the first level permutation at the first level separates the components with even indices ending in 0 from those ending in 1. At the next level subdivides each of these according to whether their second to last bits are 0 or 1. Altogether, this corresponds to arranging the components in *reverse binary order*, i.e., numerical order if we read the bits of each index from the right, the least significant bit, as opposed to the usual ordering from the left, the most significant bit. We just reverse the direction of significance. In the following table, we demonstrate the subdivisions for $N = 8$ points. The first column contains the indices in forward binary order, the second column contains the even ones indices before the odd indices, and the third contains the even indices among the even indices, the odd among the even, the even among the odd, and finally the odd among the odd.

1.	2.	3.
000	000	000
001	010	100
010	100	010
011	110	110
100	001	001
101	011	101
110	101	011
111	111	111

Each pattern in the last column is the reverse of the corresponding pattern in the first column, so we can implement the necessary rearrangement of the components of $\mathbf{f}$ by implementng a reverse binary order counter $r(i)$, and transpose u_i and $u_{r(i)}$ whenever $r(i) > i$ to avoid transposing any component with itself, or two components twice. The following C code implements this first part of the FFT:

```
i=0;                                 /* binary add 1 to i in mirror */
for (j=1; j< N; j++) {               /* count forward with j */
bit=N/2;                             /* most significant */
while ( i >= bit ) {                 /* until you encounter 0,
i=i-bit;                             /* change 1s to 0s */
bit=bit/2; }                         /* next most significant */
i=i+bit;                             /* when you do, change 0 to 1 */
if (i<j) {                           /* swap distinct pairs once */
temp=c[i]; c[i]=c[j]; c[j]=temp; } } /* next j */
```

After this rearrangement, the components needed when we put transforms together appear at the correct indices. In other words, in the final column, we want to put the components of the two point transforms in the same two locations from which they were computed In the next to last column, corresponding butterfly pairs within each four point transform are also computed from values in the corresponding locations. When these values are overwritten after they are used, they are again in the proper locations to compute the butterfly pairs of the next level transform.

This portion of the algorithm is implemented with a triply nested loop. The outermost loop counts through $i = 1, \ldots, N$ stages of putting together $\frac{N}{2^i} = 2^{N-i}$ transforms of length 2^i. Each of these levels is implemented by the doubly nested loop through the 2^{i-1} butterfly pairings of points with a common twiddle factor, through the 2^{N-i} transforms. At this deepest level, the single butterfly multiplication and corresponding addition and subtraction are performed.

The following C code implements this second part of the FFT, The code calls the complex addition, subtraction, and multiplication functions Cadd, Csub, and Cmul defined in a library, complex.h, that is also linked from the webpage.

```
numtrans=N;
sizetrans=1;
while (numtrans > 1) {
numtrans=numtrans/2;                  /* at each level, do half as many */
halfsize=sizetrans;
sizetrans=sizetrans*2;                /* subtransforms of twice the size */
for (in=0; in<halfsize; in++) {       /* index in each subtransform */
twiddle=twid[in*numtrans];            /* sharing common twiddle */
for (sub=0; sub< numtrans; sub++) {   /* index of subtransform */
iplus=sub*sizetrans+in;               /* indices for butterfly */
iminus=iplus+halfsize;
but1=c[iplus];
but2=Cmul(twiddle,c[iminus]);         /* lower one gets twiddled */
c[iplus]=Cadd(but1,but2);             /* butterfly */
c[iminus]=Csub(but1,but2); } } }
```

Many implementations, for the sake of being self-contained, sacrifice considerable efficiency and/or accuracy at this point by computing the twiddle factors ‘on the fly’ in their code. Since the exponential twiddle factors must be included in self-contained code, they will be recomputed each time the code is called. This will be quite wasteful if one needs to call the FFT many times in a computation, since the same twiddle factors are used in any FFT call with the same number of points N. A remedy is to compute the twiddle factors once and for all in an array that is placed outside the FFT routine, and simply look up the proper factor as needed in the FFT. The following C code implements computing these factors using a complex exponentiation function Cexp that is also defined in the complex.h library:

```
pi=4.0*atan(1.0);
h=2.0*pi/N;
for (j=0; j< N/2; j++) twid[j]=Cexpi(-j*h);
```

As discussed earlier, these factors can be calculated without any trigonometric function calls, using only addition, multiplication, division, and square roots by constructing inscribed 2^ℓ-gons in the unit circle using repeated bisection and scaling. The following C code implements this calculation.

```
x[0]=1.0; x[1]=0.0; x[2]=-1.0; x[3]=0.0; x[4]=1.0;
y[0]=0.0; y[1]=1.0; y[2]=0.0; y[3]=-1.0; y[4]=0.0;
N=4;
for (i=0; i< logN-2; i++) {
for (j=0; j<N; i++) {
mx=x[i][j]+x[i][j+1];my=y[i][j]+y[i][j+1];
scale=1.0/pow(mx*mx+my*my,0.5);
x[i+1][2*j+1]=mx*scale; y[i+1][2*j+1] =my*scale;
x[i+1][2*j]=x[i][j]; y[i+1][2*j] =y[i][j];
x[i+1][2*N]=1.0; y[i+1][2*N] =0.0;
N+=N;
}}
```

Even if an implementation does compute the twiddle factors ‘on the fly’, this can be performed more or less efficiently. Instead of recomputing all of the twiddle factors for each subtransform level, so the 2^2-point, 2^3-point, etc., until the 2^N point twiddle factors are computed. Since each of the subtranform twiddle factors are subsets of the final 2^N point twiddle factors, it would make sense to compute these to begin with, and reduce computation by looking up subtransform factors. This will be even more significant if each of the twiddle factors is calculated as a complex exponential (two trigonometric evaluations). Most codes that calculate the twiddle factors internally initialize with trigonometric calculations once at each level, then compute the complex powers of the initial factor with multiplications and square roots within the loop. This also has the potential disadvantage of amplifying errors with the FFT each time it is iterated. For accuracy as well as efficiency, in dedicated, repeated use it is advisable to calculate the $N/2$ twiddle factors that suffice for all levels and every call to the FFT externally and accurately once and for all, and implement the code so as to address them by lookup. Placing these factors in memory locations that are rapidly accessible can accelerate performance even further. For these and many other reasons, there are computer chips dedicated to implementing this algorithm, and its core arithmetic butterfly computation as efficiently as possible at the hardware level.