The DFT and the FFT

John Kerl

February 8, 2008

Abstract

Riemann sums and counting measure are used to motivate an intuitive derivation of the discrete Fourier transform and the fast Fourier transform. This paper serves as a gentle introduction to the applied-math notion of discrete Fourier transforms, directed toward a pure-math audience.

Contents

1 Fourier transforms and Fourier series

Faris [**Faris**] points out that the general context of Fourier analysis is an abelian group and its dual group. (The duality is in the Pontryagin sense, the details of which are beyond the scope of this paper. See [**Wik**] for a lucid discussion.) The elements of the group are typically thought of as time or space variables; the elements of the dual group are thought of as angular-frequency or wave-number variables. The following may be shown to be dual pairs of abelian groups:

(I) The real line and itself. In this case, we have Fourier transforms: given $f \in L^2(\mathbb{R}, dx)$, we compute $\hat{f} \in L^2(\mathbb{R}, dk/2\pi)$.

(II) The unit circle and the integers $\mathbb{Z}$. In this case, we have Fourier series: given $f \in L^2(T, dx)$, we compute $\mathbf{c} \in \ell^2(\mathbb{Z})$.

(III) The integers $\mathbb{Z}$ and the unit circle. This is the case for using Fourier series to reconstruct the original function.

(IV) A finite cyclic group of order N and itself. Here, we have discrete Fourier transforms.

This paper elaborates on transforms of type IV. One may define the discrete Fourier transform by fiat. Instead, though, we consider two motivating constructions.

(Note: This paper is similar to my *Poisson summation and the discrete Fourier transform*, of which the present work is a rough draft. The Poisson-summation paper omits information about the fast Fourier transform which is included here, so I have retained this draft.)

2 The discrete Fourier transform via Riemann sums

First, consider an approximation to a transform of type II. Define T_L to be the circle of circumference L. Let $f(x)$ be square-integrable on T_L, i.e. let $f(x)$ be a particular representative for $L^2(T_L, dx)$. Equivalently, think of $f(x)$ as being periodic on the real line, with period L.

The basis functions are of the form e^{ikx}, for some values of k. To see what these values must be, look at $x = L$. For e^{ikx} to have period L, kL must be an integer multiple of 2π. That is, we take

$$k = \frac{2\pi\ell}{L}$$

for $\ell \in \mathbb{Z}$.

The Fourier coefficients $\hat{f}(k)$ are

$$\hat{f}(k) = \langle e^{ikx}, f(x) \rangle = \frac{1}{L} \int_0^L e^{-ikx} f(x) dx.$$

Form a Riemann sum with N evenly spaced mesh points, deferring for a moment the well-definedness of the sum on $L^2(T_L, dx)$. We have

$$x_j = j\Delta x$$

where

$$\Delta x = \frac{L}{N}.$$

Then the integral is approximated by

$$\frac{1}{L}\int_0^L e^{-ikx} f(x)dx \approx \frac{1}{L}\sum_{j=0}^{N-1} e^{-ikx_j} f(x_j)\Delta x.$$

Now recall that the angular frequencies k were also evenly spaced, namely, with

$$k = \ell\Delta k \quad \text{and} \quad \Delta k = \frac{2\pi}{L}.$$

Thus the Δx and Δk are not unrelated. In fact,

$$\Delta k \Delta x = \frac{L}{N}\cdot\frac{2\pi}{L} = \frac{2\pi}{N}.$$

Making substitutions, we now have for the estimated integral

$$\begin{aligned}
\hat{f}(k) &\approx \frac{1}{L}\sum_{j=0}^{N-1} e^{-ikx_j} f(x_j)\Delta x \\
&= \frac{1}{L}\sum_{j=0}^{N-1} e^{-i\ell\Delta k j\Delta x} f(x_j)\Delta x \\
&= \frac{1}{L}\sum_{j=0}^{N-1} e^{-i\ell\frac{2\pi}{L}j\frac{L}{N}} f(x_j)\frac{L}{N} \\
&= \frac{1}{N}\sum_{j=0}^{N-1} e^{-i2\pi\ell j/N} f(x_j).
\end{aligned}$$

Observe that now there are N input points, $f(x_0)$ through $f(x_{N-1})$. Also observe that, when restricted to the mesh $x = j\Delta x$, the basis functions

$$e^{ikx} = e^{i2\pi\ell j/N}$$

have identical values when ℓ is shifted by multiples of N. Thus the periodicity of the coefficients arises naturally. There are N input points and N output points. Phrased differently, a pure sinusoidal input of frequency 1 is indistinguishable from a pure sinusoidal input of frequency $N+1$ when the input data are sampled only at the N mesh points. (This fact has significant repercussions in the implementation of signal-processing systems. See [**Lyons**] for more information.)

An advantage of this derivation of the discrete Fourier transform is that it uses elementary and easily visualizable Riemann sums. However, it is intuitive only: it is not well-defined on $L^2(T_L, dx)$. If f and g differ on a set of measure zero with respect to Lebesgue measure, then they will have the same Fourier coefficients on $L^2(T_L, dx)$. However, if that measure-zero set includes one of the mesh points, then the discrete Fourier transform as presented here will produce different results.

3 The DFT via counting measure

Here is a second derivation of the discrete Fourier transform. Define a counting measure χ on T_L via

$$\chi(f) = \frac{1}{N}\sum_{j=0}^{N-1} f(x_j)$$

using the same mesh $x = j\Delta x$ as before. Then for

$$f \in L^2(T_L, d\chi)$$

we have

$$\begin{aligned}
\hat{f}(k) &= \langle e^{ikx}, f(x) \rangle \\
&= \int_0^L e^{-ikx} f(x) d\chi \\
&= \frac{1}{N} \sum_{j=0}^{N-1} e^{-ikx_j} f(x_j) \\
&= \frac{1}{N} \sum_{j=0}^{N-1} e^{-i2\pi\ell j/N} f(x_j).
\end{aligned}$$

Note that due to the finiteness of the sums, the spaces $L^p(T_L, d\chi)$, $p \in [1, +\infty]$, are all the same.

This derivation of the DFT has the advantage that it is well-defined on $L^2(T_L, d\chi)$. The maximal set of measure zero is precisely

$$T_L \setminus \{0, \Delta x, \ldots, (N-1)\Delta x\}.$$

Thus if f and g differ on a set of measure zero, they must agree only at the mesh points; they may differ everywhere else. This is the direct opposite of the case when Lebesgue measure is used, where it is the set of mesh points, rather than its complement, which has measure zero. This makes sense: in applications, we take continuous electromagnetic or acoustic phenomena, then acquire only a finite number of data samples for analysis.

4 The DFT as a linear transformation on $\mathbb{C}^N$

For brevity of notation, let $\mathbf{b}$ and $\mathbf{c}$ be given by

$$b_j = f(x_j) \quad \text{and} \quad c_\ell = \hat{f}(k) = \hat{f}(2\pi\ell/L).$$

For the DFT as derived above, we have N input data points, namely $b_0, b_1, b_2, \ldots, b_{N-1}$. We also have N output data points, namely $c_0, c_1, c_2, \ldots, c_{N-1}$, given by

$$c_\ell = \frac{1}{N} \sum_{j=0}^{N-1} e^{-i2\pi\ell j/N} b_j$$

Then the input data form a vector of length N. We may take $\mathbf{b} \in \mathbb{R}^N$, but it might as well be in $\mathbb{C}^N$. Now, define the matrix W, the *Fourier matrix*, by

$$W_{\ell j} = e^{-i2\pi\ell j/N}.$$

All N Fourier coefficients, for $\ell = 0, 1, 2, \ldots, N-1$, are collected together in nothing more than the matrix product

$$\mathbf{c} = \frac{1}{N} W\mathbf{b}.$$

This makes sense: when we multiply a matrix times a vector, the ℓth slot of the output vector is the inner product of the ℓth row of the matrix with the input vector. The inner product on functions, via

$\langle f, g\rangle = \int \overline{f} g dx$, is replaced by the standard inner product on $\mathbb{C}^N$. The first vector in the inner product is $e^{i2\pi\ell j/N}$, for fixed ℓ; the second is $\mathbf{b}$. This discrete Fourier transform is then seen to be a linear transformation from $\mathbb{C}^N$ to $\mathbb{C}^N$.

For example, take $N = 4$. Then the matrix product is

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Note, for future reference, that although there are $N^2 = 16$ entries in the matrix W, there are only $N = 4$ distinct values in it.

5 The inverse DFT

Since the DFT is nothing more than multiplication by a particular matrix, its inverse must be multiplication by the inverse matrix. Recall that W is given by

$$(W)_{\ell j} = e^{-i2\pi\ell j/N}.$$

One may guess from symmetry considerations that the analogous matrix $\overline{W}$, given by

$$(\overline{W})_{\ell j} = e^{i2\pi\ell j/N},$$

is somehow involved. The matrix product is

$$(W\overline{W})_{\ell j} = \sum_{m=0}^{N-1} (W)_{km}(\overline{W})_{mj} = \sum_{m=0}^{N-1} e^{-i2\pi km/N} e^{i2\pi mj/N} = \sum_{m=0}^{N-1} e^{i2\pi m(j-k)/N}.$$

For $j \neq k$, we have a sum of $(j-k)$th roots of unity evenly spaced around the circle, which by center-of-mass considerations sum to 0. For $j = k$, on the other hand, we have 1 added to itself N times. Thus

$$W\overline{W} = NI$$

where I is the $N \times N$ identity matrix. Therefore,

$$W^{-1} = \frac{1}{N}\overline{W}.$$

6 Scaling for the DFT

There are two predominant conventions for scaling of the DFT, neither of which matches $1/N$ scaling obtained by Riemann sums or counting measure.

Recall that for the Fourier transform on the real line, we measure functions in the time domain using dx, whereas we measure functions in the frequency domain using $dk/2\pi$. That is, functions in the two domains are held up to different rulers.

For the DFT, on the other hand, we have the Euclidean norm which is a canonical way to measure vectors. Take the usual Euclidean norm on $\mathbb{C}^N$, namely,

$$\|\mathbf{b}\| = \sum_{j=0}^{N-1} |b_j|^2.$$

To see what happens to vector norms using the DFT, take the $N = 4$ example as above, and let $\mathbf{b} = (1, 0, 0, 0)^t$. Then we have

$$\frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}$$

The Euclidean norm of the input vector is

$$\|\mathbf{b}\| = \sqrt{1^2 + 0^2 + 0^2 + 0^2} = 1,$$

whereas the Euclidean norm of the output vector is

$$\|\mathbf{c}\| = \sqrt{1/16 + 1/16 + 1/16 + 1/16} = \sqrt{1/4} = 1/2.$$

If we instead take the DFT to be scaled by $1/\sqrt{N}$, then we have

$$\frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} . = \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{bmatrix}$$

Now the Euclidean norm of the output vector is

$$\|\mathbf{c}\| = \sqrt{1/4 + 1/4 + 1/4 + 1/4} = \sqrt{1} = 1.$$

Since we have $W\overline{W} = NI$, it is apparent that implementing a forward DFT using $W/\sqrt{N}$ and implementing the reverse DFT using $\overline{W}/\sqrt{N}$ recovers the input. That is, $W/\sqrt{N}$ and $\overline{W}/\sqrt{N}$ are inverses of one another. (In fact, both are unitary matrices.) This is a common convention for DFTs in engineering practice. The other common convention is to do a forward DFT using W, and to do the reverse by multiplication by $\overline{W}/N$.

There is another pair of conventions: one sometimes sees $\overline{W}$ used for the forward DFT and W used for the reverse. Thus the lack of a single standard in mathematical circles extends outside of academia.

In summary, desired properties for the DFT scaling are as follows:

- Forward DFT followed by reverse must recover the original.
- The Parseval identity: $\|\mathbf{c}\| = \|\mathbf{b}\|$. This requires $\sqrt{N}$ scaling for forward and reverse transformations, as long as we insist on holding input and output vectors up to the same Euclidean measure.

7 Input and output folding

It was seen above that the DFT output frequencies k are taken mod $2\pi N/L$. For example, frequency $2\pi/L$ is indistinguishable from $2\pi(N+1)/L$. Commonly one takes the frequencies to be from 0 to $N-1$. However, one may wish to have the frequences be zero-centered, i.e. from $-N/2$ to $(N/2) - 1$. Clearly one may accomplish this by performing the DFT, then interchanging the first $N/2$ elements of the output vector with the last $N/2$ samples. This bulk data movement is called *output folding*.

Likewise, when performing an inverse DFT, if the frequencies have been folded, then the processing should account for that. That is, suppose a time-domain vector has been forward-transformed and then output-folded. One may undo these operations by interchanging the first and second halves of the frequency vector, then performing the inverse DFT. From the point of view of the inverse DFT, then input needs to be folded.

It is perhaps less apparent that *output folding* may be accomplished by negating every other sample of the *input*, and vice versa. (In fact, for certain implementations, these negations are more efficient than the data movement required for bulk folding.) This claim needs some justification.

First, see what happens when every other sample of the input is negated.

$$\begin{aligned}
c_\ell &= \sum_{j=0}^{N-1} e^{-i2\pi\ell j/N} b_j \\
\sum_{j=0}^{N-1} e^{-i2\pi\ell j/N} (-1)^j b_j &= \sum_{j=0}^{N-1} e^{-i2\pi\ell j/N} e^{-i\pi j} b_j \\
&= \sum_{j=0}^{N-1} e^{-i2\pi\ell j/N} e^{-i2\pi j(N/2)/N} b_j \\
&= \sum_{j=0}^{N-1} e^{-i2\pi(\ell j + j(N/2))/N} b_j \\
&= \sum_{j=0}^{N-1} e^{-i2\pi(\ell+(N/2))j/N} b_j \\
&= c_{\ell+N/2}
\end{aligned}$$

where the ℓ index is taken mod N.

Second, suppose every other sample of the output is negated.

$$\begin{aligned}
(-1)^\ell c_\ell &= (-1)^\ell \sum_{j=0}^{N-1} e^{-i2\pi\ell j/N} b_j \\
&= e^{-i\pi\ell} \sum_{j=0}^{N-1} e^{-i2\pi\ell j/N} b_j \\
&= e^{-i\pi\ell(N/2)/N} \sum_{j=0}^{N-1} e^{-i2\pi\ell j/N} b_j \\
&= \sum_{j=0}^{N-1} e^{-i2\pi\ell(j+N/2)/N} b_j
\end{aligned}$$

where the j index is taken mod N.

Thus the folding claims are proved.

8 The fast Fourier transform

This section is purely expository. Here I motivate, for the pure-math reader, the derivation of the fast Fourier transform. There is little mathematical content in this section, and in fact that is the point: the FFT is nothing more than a clever trick for obtaining precisely the same end result as the DFT, using less computation time. When doing theoretical analysis, one uses the DFT rather than the FFT, since the DFT has an elegant expression as a matrix product.

Note that multiplying an $N \times N$ matrix by a vector of length N requires on the order of N^2 arithmetic operations. To see this, observe that the jth element of the output vector is nothing more than the dot product of the jth row of the matrix with the input vector. There are N of these dot products, and each of them contains N products and N sums. Thus one often hears that the DFT is $O(N^2)$.

The fast Fourier transform is a trick that was originally discovered by Gauss, and has been rediscovered at least half a dozen times over the intervening centuries. The rediscovery which stuck in the minds of the technical community was that of Cooley and Tukey in 1965, at which point computing hardware was available to make practical use of such trickery. More historical information may be found in [**NR**]. The trick is illustrated by example.

Again take $N = 4$. Writing out the matrix product in full detail, we have

$$\begin{array}{rcrcrcrcr} c_0 & = & b_0 & + & b_1 & + & b_2 & + & b_3 \\ c_1 & = & b_0 & - & ib_1 & - & b_2 & + & ib_3 \\ c_2 & = & b_0 & - & b_1 & + & b_2 & - & b_3 \\ c_3 & = & b_0 & + & ib_1 & - & b_2 & - & ib_3 \end{array}$$

Not all of the 16 sums and 16 products appear to the eye; for this small value of N, often we are multiplying by 1.

Note that the sum $b_0 + b_2$ is computed twice, as is the difference $b_0 - b_2$. Re-ordering the sums, we see

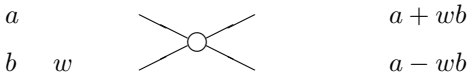
$$\begin{array}{rcrcrcrcr} c_0 & = & (b_0 & + & b_2) & + & (b_1 & + & b_3) \\ c_1 & = & (b_0 & - & b_2) & - & i(b_1 & - & b_3) \\ c_2 & = & (b_0 & + & b_2) & - & (b_1 & + & b_3) \\ c_3 & = & (b_0 & - & b_2) & + & i(b_1 & - & b_3) \end{array}$$

Here we have the four sums $(b_0 \pm b_2)$ and $(b_1 \pm b_3)$. These are followed by four products, namely multiplying by ± 1 and $\pm i$, and four more sums to finish up. Thus there are now 8 sums and 4 products, down from 16 of each. This in the essence of the fast Fourier transform.

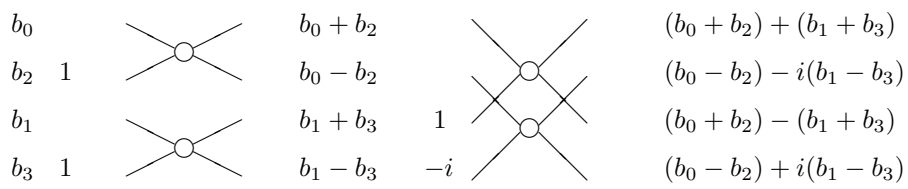
To generalize the concept beyond $N = 4$, define the *two-point butterfly operation* (this is a standard term) as follows. This is formally a map from $\mathbb{C}^3 \to \mathbb{C}^2$, given by

$$(a, b, w) \mapsto (a + wb, a - wb).$$

One writes



Using this notation, one may diagram the 4-point FFT as follows:



The general implementation of a fast Fourier transform for N a power of 2 follows this model. The following computational steps are required:

- Re-ordering the input. For $N = 4$, we changed b_0, b_1, b_2, b_3 to b_0, b_2, b_1, b_3. In general, one writes the input in *bit-reversed order*. (See [**NR**] for full details. A discussion of bits and bytes is tempting, but outside the scope of this paper.) There are no sums or products here, only data movement and/or fancy indexing of arrays.
- Each column of butterflies is called a *stage*.
- Each stage requires on the order of N sums and products.
- There are $\log_2(N)$ stages.
- At each stage, one accesses Nth roots of unity. One does not need a W matrix with N^2 elements; there are only N distinct complex numbers in the matrix. Furthermore, it may be shown that only the first $N/2$ of them are actually used during the butterfly operations. These values may be precomputed and stored in a table; the technical term is *twiddle table*.

As a result of the above facts, the FFT, for N a power of 2, has computational complexity $O(N \log_2 N)$. This is a vast improvement on the $O(N^2)$ of the DFT. For example, take $N = 1024$. Then $N^2 = 1,048,576$ whereas $N \log_2 N = 10240$, i.e. a 1024-point FFT may be executed in a hundredth the time required for a 1024-point DFT.

For N a power of 2, scaling by $1/\sqrt{N}$ may be accomplished by multiplying intermediate resultss by $1/\sqrt{2}$ at each stage, say, after the butterfly operations. Likewise, scaling by $1/N$ may be done by scaling by $1/2$ at each stage.

Last, suppose N is not a power of two. One may perform a derivation as given above, but using, say, 3-point butterflies when 3 divides N. In fact, some simplification may be done using *any* non-trivial factorization of N. When N is prime, only a DFT is possible; the more small factors N has, the better. Some software packages will factor N and do the best-possible DFT; other packages permit N to only be a power of 2, or perhaps 3 times a power of two, etc.

Full information about implementations of FFTs may be found in [**NR**]. See also [**Kerl**] for a particular power-of-two implementation.

9 The FFT via factoring the Fourier matrix

Implementations of the FFT use bit-reverse, twiddle tables, and butterfly operations as described in the previous section; an implementor needs nothing more. Yet one may wish to provide some mathematical justification, beyond a worked example for a specific value of N.

To motivate the theory, examine the 4-point FFT from the previous section. Writing the bit-reverse, stage-1 and stage-2 operations as matrices, we have

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -i \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & i \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

That is, the Fourier matrix has been factored into sparser components.

Now let N be an arbitrary even positive integer. Separating even- and odd-numbered inputs, we may write the DFT as

$$\begin{aligned} c_\ell &= \sum_{j=0}^{N-1} e^{-i2\pi\ell j/N} b_j \\ &= \sum_{j=0}^{N/2-1} e^{-i2\pi\ell 2j/N} b_{2j} + \sum_{j=0}^{N/2-1} e^{-i2\pi\ell(2j+1)/N} b_{2j+1} \\ &= \sum_{j=0}^{N/2-1} e^{-i2\pi\ell 2j/N} b_{2j} + e^{-i2\pi\ell/N} \sum_{j=0}^{N/2-1} e^{-i2\pi\ell 2j/N} b_{2j+1} \end{aligned}$$

Observe that we now have, for each ℓ, a pair of sums, each corresponding to a DFT of half the size as the original. This process may be applied recursively for each power of 2 dividing N; the recursion ends with a DFT as described in previous sections. Optimally, when N is a power of two, the final DFT is a 1-point DFT. The construction just outlined is known as the *Danielson-Lanczos lemma*.

10 Higher-dimensional DFTs

Let $T^d_{\mathbf{L}}$ be a d-dimensional torus, where $\mathbf{L} = (L_1, \ldots, L_d)$ for positive real L_i's. Write $\mathbf{x}$ and $\mathbf{k}$ for $(x_1, \ldots, x_d)$ and $(k_1, \ldots, k_d)$, respectively. To find the possible values of $\mathbf{k}$, we again require

$$k_r = \frac{2\pi\ell_r}{L_r}$$

for $\ell_r \in \mathbb{Z}$. Then for $f(\mathbf{x}) \in L^2(T^N_{\mathbf{L}}, d\mathbf{x})$ we have the d-dimensional type-II transform (i.e. Fourier series)

$$\hat{f}(\mathbf{k}) = \int_{T^d_{\mathbf{L}}} e^{-i\mathbf{k}\cdot\mathbf{x}} f(\mathbf{x}) d\mathbf{x}$$

If in addition $f(\mathbf{x}) \in L^1(T^N_{\mathbf{L}}, d\mathbf{x})$, by Fubini's theorem we may iterate the integrals to obtain

$$\begin{aligned} \hat{f}(k_1, \ldots, k_d) &= \int_0^{L_1} \cdots \int_0^{L_d} e^{-i(k_1x_1 + \ldots + k_dx_d)} f(x_1, \ldots, x_d) dx_d \cdots dx_1 \\ &= \int_0^{L_1} e^{-ik_1x_1} \left[\cdots \left[\int_0^{L_d} e^{-ik_dx_d} f(x_1, \ldots, x_d) dx_d \right] \cdots \right] dx_1. \end{aligned}$$

Now form a discrete mesh

$$\mathbf{x_j} = (j_1\Delta x_1, \ldots, j_d\Delta x_d)$$

with, for $r = 1, 2, \ldots, d$,

$$L_r = N_r \Delta x_r.$$

Define a counting measure χ via

$$\chi(f(\mathbf{x})) = \frac{1}{\prod_{r=1}^d N_d} \sum_{j_1=0}^{N_1-1} \cdots \sum_{j_d=0}^{N_d-1} f(x_{j_1}, \ldots, x_{j_d}).$$

As in the 1-dimensional case, due to the finiteness of the sums the spaces $L^p(T^N_{\mathbf{L}}, d\chi)$, $p \in [1, +\infty]$, are all the same.

One may again form a Riemann sum, or take the integral using the counting measure. One obtains

$$\begin{aligned}\hat{f}(\mathbf{k}) &= \int_{T^d_{\mathbf{L}}} e^{-i\mathbf{k}\cdot\mathbf{x}} f(\mathbf{x}) d\chi \\ &= \sum_{j_1=0}^{N_1} \frac{e^{-i2\pi\ell_1 j_1/N_1}}{N_1} \cdots \sum_{j_d=0}^{N_d} \frac{e^{-i2\pi\ell_d j_d/N_d}}{N_d} f(x_{j_1}, \ldots, x_{j_d})\end{aligned}$$

11 Decomposition of higher-dimensional DFTs

Take the DFT as in the previous section to be a linear transformation on $\mathbb{C}^{N_1} \times \cdots \times \mathbb{C}^{N_d}$. For $r = 1, 2, \ldots, d$ and $j_r = 0, 1, 2, \ldots, N_r - 1$ write

$$b_{j_1,\ldots,j_d} = f(x_{j_1}, \ldots, x_{j_d}) \qquad \text{and} \qquad c_{\ell_1,\ldots,\ell_d} = \hat{f}(k_{\ell_1}, \ldots, k_{\ell_d}).$$

Then

$$c_{\ell_1,\ldots,\ell_d} = \sum_{j_1=0}^{N_1} \frac{e^{-i2\pi\ell_1 j_1/N_1}}{N_1} \cdots \sum_{j_d=0}^{N_d} \frac{e^{-i2\pi\ell_d j_d/N_d}}{N_d} b_{j_1,\ldots,j_d}.$$

As before, each of the j_r's and ℓ_r's are taken mod N_r. There are $\prod_{r=1}^d N_r$ output values $c_{\ell_1,\ldots,\ell_d}$, and for each of them there are on the order of $\prod_{r=1}^d N_r$ arithmetic operations to evaluate the nested sum. Thus, it would appear that the d-dimensional DFT has complexity $O(N_1^2 \cdots N_d^2)$. In the particular case when all the N_r's are the same, say $N_r = N$, this is $O(N^{2d})$.

However, one may decompose the d-dimensional DFT as a sequence of 1-dimensional DFTs, avoiding some redundant computation. The key is that the inner sum is invariant under the outer sums, and thus does not need to be recomputed, and so on for each sum. Let

$$\omega_m = e^{-i2\pi/m}.$$

Form the sequence $I^{(0)}, \ldots, I^{(d)}$ of d-dimensional arrays via

$$\begin{aligned} I^{(0)}_{j_1,\ldots,j_d} &= b_{j_1,\ldots,j_d} \\ I^{(1)}_{\ell_1,j_2,\ldots,j_d} &= \sum_{j_1=0}^{N_1} \frac{\omega_{N_1}^{\ell_1,j_1}}{N_1} I^{(0)}_{j_1,\ldots,j_d} \\ I^{(2)}_{\ell_1,\ell_2,j_3,\ldots,j_d} &= \sum_{j_2=0}^{N_2} \frac{\omega_{N_2}^{\ell_2,j_2}}{N_2} I^{(1)}_{\ell_1,j_2,\ldots,j_d} \\ &\vdots \\ I^{(d)}_{\ell_1,\ldots,\ell_d} &= \sum_{j_d=0}^{N_d} \frac{\omega_{N_d}^{\ell_d,j_d}}{N_d} I^{d-1}_{\ell_1,\ldots,\ell_{d-1},j_d} \\ &= b_{\ell_1,\ldots,\ell_d} \end{aligned}$$

Each sum visibly consists of $O(N_d)$ operations, and there are $\prod_{r=1}^d N_r$ elements in each of the $I^{(r)}$ arrays. Thus the total complexity is

$$O(N_1^2 N_2 \cdots N_d + N_1 N_2^2 \cdots N_d + \ldots + N_1 \cdots N_{d-1} N_d^2).$$

In the particular case when all $N_r = N$, this is $O(dN^{d+1})$.

Another special case is when $d = 2$. Here, the input data form a $N_1 \times N_2$ matrix, and the decomposition reduces to doing 1-dimensional DFTs along each row of the matrix, then doing 1-dimensional DFTs along each column. (For this reason, decomposition of the multi-dimensional DFT into 1-dimensional DFTs is often referred to as the *row-column algorithm*.) That is, there are N_1 1-dimensional DFTs with complexity $O(N_2^2)$, followed by N_2 1-dimensional DFTs with complexity $O(N_1^2)$.

The 1-dimensional DFTs may be replaced with FFTs, with optimal performance when each of the N_r's is a power of two. In this case, the complexity becomes

$$\begin{aligned} & O(N_1 \log_2(N_1) N_2 \cdots N_d + N_1 N_2 \log_2(N_2) \cdots N_d + \ldots + N_1 \cdots N_{d-1} N_d \log_2(N_d)) \\ = \quad & O\left([N_1 \cdots N_d]\left[\log_2(N_1) + \ldots + \log_2(N_d)\right]\right) = O\left(\left[\prod_{r=1}^{d} N_d\right]\left[\sum_{r=1}^{d} \log_2(N_d)\right]\right) \\ = \quad & O\left(\left[\prod_{r=1}^{d} N_d\right]\left[\log_2\left(\prod_{r=1}^{d} N_d\right)\right]\right) = O(M \log_2(M)) \end{aligned}$$

if we define $M = \prod_{r=1}^{d} N_r$. Thus, the performance benefit of the one-dimensional fast Fourier transform extends to higher dimensions.

References

[**Faris**] Faris, W. *Real Analysis*. http://math.arizona.edu/~faris/realweb/realS06.html.

[**Kerl**] Kerl, J. *A radix-2 1D FFT*. http://math.arizona.edu/~kerl/python/pyfft_m.py.

[**Lyons**] Lyons, R. *Understanding Digital Signal Processing* (2nd ed.). Prentice Hall, 2004.

[**NR**] Press, W. et al. *Numerical Recipes* (2nd ed.). Cambridge, 1992.

[**Wik**] Articles on *Quadrature* and *Pontryagin duality*. http://en.wikipedia.org.