Discrete Fourier Transforms

1 Introduction and Definition

Fourier transforms are one of the workhorses of physics, both computational and otherwise. On a computer, we in generaly carry out *discrete Fourier transforms* or DFTs, where we sum (rather than integrate) over a discrete set of function points, and evaluate the DFT at a discrete set of k's. Because of this, the DFTs can differ in some subtle ways from analytic Fourier transforms.

The fundamental definition of the DFT as usually carried out on a computer is:

$$\mathrm{F}(k) = \sum_{x=0}^{N-1} f(x) \exp(-2\pi ikx/N$$

If we have N input data points, then x ranges from 0 to $N-1$, and we evaluate for $k=0$ to $N-1$. This is one of the few formulas you should just memorize, numerical factors and all. There will come a day when you will be grateful for having done so.

2 DFTs as Matrix Multiplies

If you look at the definition of the DFT, you'll see it's a matrix multiply. We have an input vector of length N, and an output vector of length n, and each output element is a sum over coefficients times input elements. The $(x,k)^{th}$ entry is $\exp(-2\pi ikx/N)$. The matrix is symmetric, since we can swap x and k without changing coefficients. Less obviously, columns (and hence rows) of the matrix are orthogonal to each other. The dot product of the k and k' columns is

$$\sum_{x=0}^{N-1} \exp(-2\pi ikx/N) \exp(2\pi ik'x/N) = \sum_{x=0}^{N-1} \exp(-2\pi i(\delta_k)x/N)$$

where $\delta_k \equiv k - k'$. The sign flip comes about because when we take the dot product of complex vectors, we standardly conjugate one of them. Note that this dot product is now a geometric series, so we can evaluate it by taking the sum from $x=0$ to infinity, and subtracting the sum from $x=N$ to infinity. Let $x' \equiv x+N$, and we have:

$$\sum_{x=0}^{\infty} \exp(-2\pi i(\delta_k)x/N) - \sum_{x=N}^{\infty} \exp(-2\pi i(\delta_k)x/N) = \sum_{x=0}^{\infty} \exp(-2\pi i(\delta_k)x/N) - \sum_{x'=0}^{\infty} \exp(-2\pi i(\delta_k)(x'-N)/N)$$

We can pull the $-N$ factor out of the exponential, and decide to relabel x' to x (which we are allowed to do since we are just summing over it), leaving the second term:

$$\exp(2\pi i\delta_k) \sum_{x=0}^{\infty} \exp(-2\pi i(\delta_k)(x)/N)$$

Except for the coefficient, this is now identical to the original sum, so we have the dot product between columns is:

$$(1 - \exp(-2\pi i\delta_k) \sum_0^\infty \exp(-2\pi i\delta_k x/N)$$

Because we restrict k to be an integer, δ_k must also be an integer, and so the leading factor must be zero. The only exception is when $\exp(-2\pi i(\delta_k)x/N)$ is a constant. This happens when $\delta_k = 0$, since the input to the exponent is always zero, $\exp(0) = 1$, and so we have $\sum_{x=0}^{N-1} 1 = N$. However, note that whenever $\delta_k = mN$, then the $N's$ cancel, and we're left with an integral multiple of 2π for the input to the exponential, and we again have the exponential evaluate to 1 for all values of x. While this is not normally an issue since k is restricted to be in $[0, N-1]$, we will see this behavior become important in the context of aliasing.

Since the columns are orthogonal, we can pretty trivially write down the inverse matrix. All we have to do is flip the sign of the exponent, and divide by N. In fact, in principle we have a lot of freedom about where we put the $1/N$, and various conventions are used in analytic Fourier transforms. For the DFT, however, the convention that the forward transform gets the negative, and the inverse transform gets $1/N$ is universal as far as your instructor knows. The inverse DFT is then

$$f(x) = \frac{1}{N} \sum_{k=0}^{N-1} F(k) \exp(2\pi ikx/N)$$

To recap, the DFT rotates a vector into a new space where the basis vectors are sines and cosines. The rotation is invertible, and we can get back to where we started by carrying out the same rotation, slightly modified by a complex conjugate and normalization factor.

3 Some DFT PRoperties

Many of the propertis of the DFT carry over directly from their continuous analogs. These include:

The Fourier transform of a δ-function is a constant. In discrete mathematics, the Dirac $\delta function$ becomes the Kronecker δ, where $\delta(x) = 0$, except $\delta(0) = 1$. We then have

$$F(k) = \sum \delta(x) \exp(-2\pi ikx/N) = \exp(-2\pi ik0/N) = 1$$

for all k. The DFT of $f(x) = 1$ similarly is $F(k) = N\delta(k)$. One way to see this is that $f(x) = 1$ is just the basis vector for $k = 0$, which we know is orthogonal to all non-zero k basis vectors, so the only non-zero term has to be the $k = 0$ value.

3.1 Shifted DFT

If we know the DFT of a function, we also can work out the DFT of a shifted version of the function.

$$\sum f(x-m)\exp(-2\pi ikx/N) = \sum f(x)\exp(-2\pi ik(x+m)/N) = \exp(-2\pi ikm/N)\sum(f(x)\exp(-2\pi kx/N)$$

In short, we apply a phase ramp with slope $\exp(-2\pi im/N)$ to the original Fourier transform.

3.2 Flipped DFT

If we know the DFT of a function, the DFT of the flipped version $f(-x)$ is:

$$F(K) = \sum f(-x)\exp(-2\pi ikx/N) = \sum f(x)\exp(2\pi ikx/N) = (\sum f^*(x)\exp(-2\pi ikx/N))^*$$

This is the conjugate of the DFT of the conjugate of $f(x)$. For the special (but common) case that $f(x)$ is real, then the DFT of $f(-x)$ is the complex conjugate of the DFT of $f(x)$.

3.3 Aliasing

If we have $F(k)$, what is $F(k+mN)$ for integer m ? Well, that is

$$\sum f(x)\exp(-2\pi i(k+mN)x/N) = \sum f(x)\exp(-2\pi i(kx/N))\exp(-2\pi imx)$$

The second term is always 1 since m, x are integers, so $F(k+mN) = F(k)$. Similarly, if we are taking the inverse DFT, $IDFT(x+mN) = IDFT(x)$. While we usually only think about a single period of the DFT, in practice they repeat infinitely and you would do well to keep that in mind. In particular, a jump from the right edge of our function to the left edge looks looks just like a jump in the middle of our function - one way to see this is to apply a phase ramp from the shifting theorem to move the edge of the domain into the middle.

Similarly, if we have high frequency components in our data, where $k > N$, to the DFT those components are indistinguishable from components where $0 \leq k < N$. This can be a huge problem when applying DFTs to signal processing, and usually one uses analog (not digital) filters to make sure that the high frequency components are gone before the digital system ever sees the signals.

3.4 DFT of Real Functions

Applying the flip theorem to $F(k)$, we have $F(-k) = (DFT(f(x)^*))^*$. When $f(x)$ is real, then this reduces to $F(-k) = F(k)^*$. However, from aliasing, we know that $F(-k) = F(N-k)$, so for real data, we have

$$F(N-k) = F(k)^*$$

i.e. the second half of the DFT is just the flip of the conjugate of the first half. This is such a common case that Fourier transform libraries universally support the transforms of real data, and only return the first half of the DFT. You would generally do well to keep this aliasing in mind, and think of the highest frequencies in the DFT as the lowest negative frequencies.

3.5 Sampling/Nyquist Theorem

We now have enough bits in place to understand one of the fundamental theorems in digital signal processing. If we have a signal with maximum frequency ν, how fast do we have to measure that signal to capture all its information? Naively, you might think if you had a signal say with frequency up to 1 MHz, that you would have to measure the signal one million times per second. This isn't right, because really the signal generally has components from -1 MHz to 1 MHz, for a total width of 2 MHz. That means we had better sample 2 million times per second to capture all the information in the signal. If we sample faster than that, we don't learn anything new because the new frequencies we are measuring have zero amplitude by construction. We can shift the signal around in frequency space by multiplying by a real-space phase ramp (which is sin/cos waves). The power to shift means it's the total bandwidth of the signal that matters and not the center frequency, which leads to the Nyquist (or sampling) theorem. *If we have a band-limited signal of bandwidth ν, we get complete information about the signal if we sample at rate $\frac{1}{2\nu}$.* This is referred to as *Nyquist sampling.* Another way to see the Nyquist theorem is that N independent time samples leads to $N/2$ independent complex frequency amplitudes. So, you'd better have twice as many time samples as the number of frequency measurements you want. This checks out from a purely information-theory view since it takes $2N$ real numbers to describe N complex numbers.

4 Discrete Convolution Theorem

Convolutions arise naturally in a wide range of physical systems, such as optics, electronics, and even Poisson's equation. The one-dimensional continuous definition is

$$h(t) = f \circledast g \equiv \int f(\tau)g(t-\tau)d\tau$$

The way this is usually interpreted is there is some underlying signal $f(\tau)$. Our system responds to a delta function signal at the origin with $g(t)$, and so will respond at time t to a signal at time τ with $g(t-\tau)$ times the signal amplitude. The total signal we see is then the integral over all incoming signals, which gives us the definition for h.

In the discrete limit, the convolution becomes

$$h(x) = f \circledast g \equiv \sum f(\chi)g(x-\chi)$$

If we swap in the inverse Fourier transforms of the functions on the right hand side, something remarkable happens. Recall that

$$f(\chi) = \frac{1}{N}\sum F(k)\exp(2\pi i k\chi/N)$$

and

$$g(x-\chi) = \frac{1}{N}\sum G(k')\exp(2\pi i k'(x-\chi)/N)$$

Swapping these in, we have

$$h(x) = \frac{1}{N^2}\sum_{\chi=0}^{N-1}\sum_{k} F(k)\exp(2\pi i k\chi/N)\sum_{k'} G(k)\exp(2\pi i k'(x-\chi)/N)$$

We are free to reorder the sums, so we'll move the sum over χ to the right, and put all terms containing χ to the right of that summation, and all other terms to the left. That gives:

$$h(x) = \frac{1}{N^2}\sum_{k}\sum_{k'} F(k)G(k')\exp(2\pi i k'x/N)\sum_{\chi}\exp(2\pi i(k-k')x/N)$$

Now, the sum of χ is zero, unless $k=k'$, in which case it is N. That means the only terms that contribute are when $k=k'$, so we can replace the double sum with a single sum over k, plus cancel out one factor of N. This gives:

$$h(x) = \frac{1}{N}\sum_{k} F(k)G(k)\exp(2\pi i kx/N)$$

This is just the inverse DFT of F times G, so we have the rather remarkable result that

$$H(k) = F(k)G(k)$$

If we want to convolve two function in real space, we just need to multiply them in Fourier space. This is true both for discrete and continuous functions.

4.1 Cross-correlations

Another common class of integrals is the cross-correlation of two functions, $\int f(\tau)g(t+\tau)d\tau$. In the discrete case, it becomes $\sum f(\chi)g(x+\chi)$. We can proceed exactly as in the convolution case, but the interior sum becomes $\sum \exp(2\pi i(k+k')\chi/N$, which requires $k=-k'$. This gives a sign flip:

$$corr(f,g) = \frac{1}{N}\sum_{k} F(-k)G(k)\exp(2\pi i kx/N)$$

If f is real, then we know that $F(-k) = F^*(k)$, and we have:

$$corr(f,g) = IDFT(F^*G)$$

i.e. we take the complex conjugate of one of the Fourier transforms.

5 Fast Fourier Transform

If we look at the definition of the DFT, it looks like we need to do n complex operations at each value of k, and since there are n values of k, we need to do a total of n^2 multiplies/adds. For large values of n, this can be quite slow. However, we can do better than this. The classic description (usually credited to Cooley/Tukey, but going back to Gauss) breaks up the DFT into even/odd values of x:

$$F(k) = \sum_{x even} f(x)\exp(-2\pi ikx/N) + \sum_{x_odd} f(x)\exp(-2\pi ikx/N)$$

Let f_e be the even entries of f and f_o be the odd entries. Let $M = N/2$, and let $x = 2x'$ in the first sum, and $x = 2x' + 1$ in the second. The DFT then becomes

$$F(k) = \sum_{x'} f_e(x')\exp(-2\pi ikx'/M) + \sum_{x'} f_o(x')\exp(-2\pi ikx'/M)\exp(-2\pi ik/N)$$

The sums are now just the Fourier transforms of the even/odd entries, so this gives

$$F(k) = F_e(k) + \exp(-2\pi ik/N)F_o(k)$$

We have replaced the original Fourier transform with two half-length Fourier transforms, with some phase factors added in (conventionally known as *twiddle factors*). The Fourier transforms F_e and F_o just repeat for $k > N/2$, and the twiddle factors change sign for $k > N/2$, so we can construct the full Fourier transform $F(k)$ where the first half is $F_e + twidF_o$ and the second half is $F_e - twidF_o$.

If you look at the DFT definition, the DFT of a single number is just that number. We can now outline the fast Fourier transform (FFT). Assume our vector has length equal to a power of 2. Take the DFT of the even entries and the DFT of the odd entries, and reconstruct the DFT using the twiddle factors. How do we take the DFT of the even/odd halves? Well, just split them into their even/odd halves as well and repeat, until we get to the DFT of a single number.

How much work is this? Well, at the first stage, we have to combine 2 Fourier transforms of length $N/2$, so order N complex multiplies/adds. To take those Fourier transforms requires splitting the even/odd parts into 4 pieces of length $N/4$, so the work required to reconstruct them is again order N. The third pass takes 8 Fourier transforms of length $N/8$, so once again N operations, and so on. We stop when the length gets to one, and since we halve the length at each pass, the total number of passes is $\log_2(N)$. Since each pass requires N operations, the total operation count goes like $N\log(N)$. For large N, this is very much smaller than N^2. To take the Fourier transform of a vector of a million numbers, the run-time of the recursive algorithm is order 50,000 times faster than the naive implementation. For a billion numbers, we win by a factor of 30 *million.* These speedups are so huge the algorithm (along with related ones) is known as the fast Fourier transform or FFT. In practice the usage of the FFT is so dominant that DFT and FFT (which is one implementation of the DFT) are in practice used interchangably.

The FFT is most simply written for vectors with lengths equal to a power of 2. This has lead many people to believe the data vector length *must* be a power of 2. They are mistaken. If you look at the DFT, you could split the data into every third entry and derive a similar recursive algorithm where you combine three sub-DFTs with two sets of twiddle factors. This gives an algorithm where we do $2N$ work at each pass, but only do $\log_3(N)$ passes, yielding a constant $\sim$25% slower factor. Similar efficient tricks exist for other prime factors. On physical computers, memory bandwidth is an important constraint as well, and so in practice as long as the length is a product of "small" prime factors, the FFT will run efficiently and is often even faster for non-power of 2 lengths. As the prime factors get larger, the efficiency drops off, though even prime numbers can be Fourier transformed in $N\log(N)$ time (with a large pre-factor). If you are Fourier transforming lots of data, I suggest truncating to a length that's powers of say [2,3,5,7,11,13]. For large data sets, this usually amounts to trimming (much) less than one percent of the data.