AMath 590
Autumn Quarter, 2013
R. J. LeVeque

Fourier Transforms

This is a summary of some key facts about Fourier integrals, series, sums, and transforms, and the manner in which these concepts relate to one another. Many different notations are used in the literature and in software for Fourier transforms, so it is important to make sure you understand the notation and scaling used in the paper/book/software you are using.

In the "physical domain" I will use $u(x)$ to denote a function of x defined for some interval of x values (possibly all real x) and u_j to denote a discrete set of values over some set (finite or infinite) of integer indices j.

In the "Fourier domain" I will use $\hat{u}(\xi)$ to denote a function of ξ defined for some interval of wave number values (possibly all real ξ) and $\hat{u}_k$ to denote a discrete set of wave number values over some set (finite or infinite) of integer indices k.

There are four cases to consider:

Case 1. The data is an L_2 function $u(x)$ defined for all real x.

Case 2. The data is a function $u(x)$ defined on a finite interval $0 < x < 2\pi$ or $-\pi < x < \pi$ (or a periodic function defined on one period).

Case 3. The data is a discrete grid function u_j on an infinite grid, at points $x_j = jh$ for $j = 0,\ \pm 1,\ \pm 2,\ \ldots$ and some $h > 0$.

Case 4. The data is a discrete grid function u_j on a grid with $N = 2\pi/h$ points in the interval $[-\pi, \pi]$ (or a periodic grid function defined on one period).

The Fourier transform $\hat{u}$ has a different form in each case, as listed below for each of case. Note the duality: the data in Case 2 has the same form as the transform in Case 3 and vice versa. Cases 1 and 4 are self-dual; the data and transform have the same general form.

Case 1. The Fourier transform $\hat{u}(\xi)$ is an L_2 function defined for all real ξ.

Case 2. The Fourier transform $\hat{u}_k$ is a set of discrete values defined on an infinite grid, for wave numbers $\xi_k = k = 0,\ \pm 1,\ \pm 2,\ \ldots$.

Case 3. The Fourier transform $\hat{u}(\xi)$ is a function defined on a finite interval $[-\pi/h,\ \pi/h]$.

Case 4. The Fourier transform $\hat{u}_k$ is a set of N discrete values for wave numbers $\xi_k = k = -N/2 + 1,\ \ldots,\ N/2$. This grid covers the interval $[-\pi/h,\ \pi/h]$.

The formulas in each case are given on the next page. One particular normalization has been chosen, more about this below.

Fourier transform and inverse transform formulas:

Case 1. The data is an L_2 function $u(x)$ defined for all real x. The Fourier transform $\hat{u}(\xi)$ is an L_2 function defined for all real ξ.

Forward Transform:

$$\hat{u}(\xi) = \int_{-\infty}^{\infty} e^{-i\xi x}\, u(x)\, dx \qquad \text{for } -\infty < \xi < \infty. \tag{1a}$$

Inverse transform:

$$u(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{i\xi x}\, \hat{u}(\xi)\, d\xi \qquad \text{for } -\infty < x < \infty. \tag{1b}$$

Case 2. The data is a function $u(x)$ defined on a finite interval $0 < x < 2\pi$ (or a periodic function defined on one period). The Fourier transform $\hat{u}_k$ is a set of discrete values defined on an infinite grid, for wave numbers $\xi_k = k = 0,\ \pm 1,\ \pm 2,\ \ldots$. This grid covers the interval $[-\pi/h,\ \pi/h]$.

Forward Transform:

$$\hat{u}_k = \int_0^{2\pi} e^{-ikx}\, u(x)\, dx \qquad \text{for } k = 0,\ \pm 1,\ \pm 2,\ \ldots. \tag{2a}$$

Inverse transform:

$$u(x) = \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} e^{ikx}\, \hat{u}_k \qquad \text{for } 0 < x < 2\pi. \tag{2b}$$

Case 3. The data is a discrete grid function u_j on an infinite grid, at points $x_j = jh$ for $j = 0,\ \pm 1,\ \pm 2,\ \ldots$ and some $h > 0$. The Fourier transform $\hat{u}(\xi)$ is a function defined on a finite interval $[-\pi/h,\ \pi/h]$.

Forward Transform:

$$\hat{u}(\xi) = \sum_{j=-\infty}^{\infty} e^{-i\xi x_j}\, u_j \qquad \text{for } -\frac{\pi}{h} < \xi < \frac{\pi}{h}. \tag{3a}$$

Inverse transform:

$$u_j = \frac{1}{2\pi} \int_{-\pi/h}^{\pi/h} e^{i\xi x_j}\, \hat{u}(\xi)\, d\xi \qquad \text{for } j = 0,\ \pm 1,\ \pm 2,\ \ldots. \tag{3b}$$

Case 4. The data is a discrete grid function u_j on a grid with $N = 2\pi/h$ points in the interval $[0, 2\pi]$ (or a periodic grid function defined on one period). The grid points are $x_j = jh$ for $j = 1,\ 2,\ \ldots,\ N$, where $h = 2\pi/N$. The Fourier transform $\hat{u}_k$ is a set of N discrete values for wave numbers $\xi_k = k = -N/2+1,\ \ldots,\ N/2$.

Forward Transform:

$$\hat{u}_k = h \sum_{j=1}^{N} e^{-ikx_j}\, u_j \qquad \text{for } k = -\frac{N}{2} + 1,\ \ldots,\ \frac{N}{2}. \tag{4a}$$

Inverse transform:

$$u_j = \frac{1}{2\pi} \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} e^{ikx_j}\, \hat{u}_k \qquad \text{for } j = 1,\ 2,\ \ldots,\ N. \tag{4b}$$

Some things to note and help motivate the different forms:

- If the data is a function $u(x)$, then this data may contain arbitrarily high wave number oscillations, and so the transform $\hat{u}$ may be nonzero for arbitrarily large values of $|\xi|$ (in Case 1) or k (in Case 2).
- If the data consists of discrete values then there is a limit to how high a wave number can be represented on the grid, and so in Cases 3 and 4 the Fourier transform is "band limited" and the wave number does not exceed π/h in magnitude.
- If the data $u(x)$ or u_j is given only on a finite interval (which is viewed as one period of periodic data), as in Cases 2 and 4, then the Fourier transform $\hat{u}_k$ takes discrete values since $e^{i\xi x}$ satisfies the periodicity requirement only for integer wave numbers $\xi_k = k$.
- If the data $u(x)$ or u_j is given over an infinite interval (non-periodic), as in Cases 1 and 3, then there is no periodicity requirement constraining ξ to integers and the Fourier transform $\hat{u}(\xi)$ is defined over an interval of ξ values.

Normalization.

In each case you can modify the definition of $\hat{u}$ by multiplying by a constant C and then the formula for $u(x)$ must be multiplied by $1/C$ to cancel this factor out. In particular, if the formulas above for the Fourier transform $\hat{u}$ are multiplied by $\frac{1}{2\pi}$, then the inverse transform formulas for u will be multiplied by 2π, with the result that the factor $\frac{1}{2\pi}$ will appear in the formulas for for u rather than $\hat{u}$. This alternative is often seen in the literature.

Another possibility is to use the factor $C = 1/\sqrt{2\pi}$, with the result that both the formulas for u and the formulas for $\hat{u}$ will contain a factor $1/\sqrt{2\pi}$. This is sometimes used, particularly in Case 1, since it gives a more symmetric and self-dual formula. However, it also leads to constantly writing $1/\sqrt{2\pi}$ in every formula.

Discrete Fourier transforms.

Case 4 is often used computationally: a finite set of N data values u_j is transformed into a finite set of N values $\hat{u}_k$. This is often called the *discrete Fourier transform* (DFT). The formulas for the DFT and inverse DFT (IDFT) each involve finite sums and can be evaluated directly (unlike the other cases, which involve integrals that one may not be able to evaluate exactly).

The formula for each $\hat{u}_k$ is a sum of N terms, so computing this value for one k requires $\mathcal{O}(N)$ floating point operations. Hence it appears that computing all N components of the vector $\hat{u}$ will require N times as much work, or $\mathcal{O}(N^2)$ operations. Note that the process of going from a vector u of N data values to the vector $\hat{u}$ of N transform values can be interpreted as simply multiplying by an $N \times N$ matrix R with (k, j) element equal to he^{-ikjh}. Since this is a dense matrix, this product computed in the standard way requires $\mathcal{O}(N^2)$ operations.

However, this matrix has a very special structure that can be exploited to compute this particular matrix-vector product in far fewer operations. This works best if N is a power of 2, or more generally of small primes. The fast algorithm can be derived in various ways. The main idea is that if $N = 2^s$ then the matrix R can be written as the product of $s = \log_2 N$ other matrices, each of which is very sparse: $R = R_s R_{s-1} \cdots R_1$. We can compute the product Ru by first multiplying u by R_1, then multiplying the resulting vector by R_2, etc. Each of these matrix-vector products only requires $\mathcal{O}(N)$ operations, because of the sparsity, and so computing Ru in this way only requires $\mathcal{O}(N \log N)$ operations. This algorithm is called the *Fast Fourier Transform* (FFT).

For some applications, very large values of N are used and use of the FFT results in huge savings (by nearly a factor N over the naive algorithm). For spectral methods this is usually not such an issue since relatively small values of N are often used, but it still makes sense to use FFT software rather than using the slow algorithm. Not only is it faster, it is also often more stable numerically.

For the DFT (including FFT software) somewhat different forms of Case 4 above are often seen. In particular, redefining $\hat{u}_k$ by multiplying the formula (4a) by $1/h$ and the formula (4b) for u_j by h gives the alternative formulation

Case 4 (Alternative scaling).

Forward Transform:

$$\hat{u}_k = \sum_{j=1}^{N} e^{-ikx_j}\, u_j \qquad \text{for } k = -\frac{N}{2}+1,\ \ldots,\ \frac{N}{2}. \tag{5a}$$

Inverse transform:

$$u_j = \frac{1}{N} \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} e^{ikx_j}\, \hat{u}_k \qquad \text{for } j = 1,\ 2,\ \ldots,\ N. \tag{5b}$$

where we have used $h/2\pi = 1/N$ to simplify the factor in the transform (5a).

Another form often used is a variant of this:

Case 4 (Alternative indexing of wave numbers).

Forward Transform:

$$\hat{u}_k = \sum_{j=0}^{N-1} e^{-ikx_j}\, u_j \qquad \text{for } k = 0,\ 1,\ \ldots,\ N-1. \tag{6a}$$

Inverse transform:

$$u_j = \frac{1}{N} \sum_{k=0}^{N-1} e^{ikx_j}\, \hat{u}_k. \qquad \text{for } j = 0,\ 1,\ \ldots,\ N-1. \tag{6b}$$

This form has the virtue of maximum simplicity, and is used in the Matlab `fft` routine, for example. Recall that in these formulas

$$x_j = jh = \frac{2\pi j}{N}.$$

Note that essentially the same formula as in (5a) is used for the transform, but for a different set of k values. By the assumed periodicity, $u_0 = u_N$ and the change in the limits of summation in (6a) is made just to be consistent with (6b). The change of indices from (5b) to (6b) appears more substantial, but in fact $\hat{u}_{-k} = \hat{u}_{N-k}$ so this is just a relabelling. This relation follows from

$$\begin{aligned}
\hat{u}_{-k} &= \sum_j \exp(-i(-k)2\pi j/N) u_j \\
&= \sum_j \exp(-i(N-k)2\pi j/N) \exp(iN2\pi j/N) u_j \\
&= \sum_j \exp(-i(N-k)2\pi j/N) u_j \\
&= \hat{u}_{N-k}
\end{aligned}$$

since $\exp(iN2\pi j/N) = (e^{2\pi i})^j = 1$. So the following two vectors of transforms are equivalent:

$$\begin{aligned}
&\hat{u}_0,\ \ \hat{u}_1,\ \ \ldots,\ \ \hat{u}_{N/2},\ \ \hat{u}_{N/2+1},\ \ \ldots,\ \ \hat{u}_{N-1} \\
&\hat{u}_0,\ \ \hat{u}_1,\ \ \ldots,\ \ \hat{u}_{N/2},\ \ \hat{u}_{-N/2+1},\ \ \ldots,\ \ \hat{u}_{-1}.
\end{aligned}$$

The FFTW package (often used in Fortran, C, Python, etc.) computes unnormalized versions of (6), with no factor of $1/N$ in either term. So applying the FFT and then IFFT with this software gives the original sequence multiplied by a factor N.