Fourier Transforms

In this chapter I provide a summary of various transform pairs. Notation varies in the literature and I present here a self-consistent treatment.

Sine Transform

Given a function $u(x)$ on the interval $[0, \ell]$, the sine transform and its inverse are given by:

$$\textit{Sine Transform:} \quad a_m = \frac{2}{\ell} \int_0^\ell u(x) \sin(\pi m x/\ell) dx \tag{1}$$

$$\textit{Inverse Sine Transform:} \quad u(x) = \sum_{m=1}^{\infty} a_m \sin(\pi m x/\ell) \tag{2}$$

To prove that these are indeed inverses of one another, one uses the *orthogonality relation* for sine functions.

$$\int_0^\ell \sin(\pi m x/\ell) \sin(\pi m' x/\ell) dx = \frac{\ell}{2} \delta_{mm'} \tag{3}$$

which is easy to prove by direct integration. One can also use the *completeness relation*:

$$\sum_{m=1}^{\infty} \sin(\pi m x/\ell) \sin(\pi m x'/\ell) = \frac{\ell}{2} \delta(x - x') \tag{4}$$

which is not easy to prove.

Discrete Sine Transform

There is a discrete form of the sine transformation, the discrete sine transform (DST). In this case we are given a set of value U_j for $j = 1, \ldots, J-1$ generally thought of as a discrete sampling of a function $u(x)$ on $[0, \ell]$, i.e. $U_j = u(jh)$, which $h = \ell/J$. For the DST the function is sampled only on the interior $j = 1, 2, \ldots, J-1$. Then the discrete sine transform (DST) and its inverse are given by:

$$\textit{DST:} \quad a_m = \frac{2}{J} \sum_{j=1}^{J-1} U_j \sin(\pi m j/J), \quad m = 1, 2, \ldots, J-1 \tag{5}$$

$$\textit{Inverse DST:} \quad U_j = \sum_{m=1}^{J-1} a_m \sin(\pi m j/J), \quad j = 1, 2, \ldots, J-1 \tag{6}$$

One may think of (5) as arising from (1) by replacing the integral by a discrete (trapezoid) approximation.

Note that if one included the cases $m = 0$ and $m = J$ in (5) one would find that $a_0 = a_J = 0$. Hence there is not reason to include these. Similarly, U_0 and U_J would necessarily be zero by (6). Hence it is not possible to recover non-zero values of U_0 and U_J via the inverse DST and these are excluded.

The proof that the above pairs are inverses is easy given the orthogonality relation:

$$\sum_{j=1}^{J-1} \sin(\frac{\pi m j}{J}) \sin(\frac{\pi m' j}{J}) = \frac{J}{2} \delta_{mm'} \tag{7}$$

Note that m and j play exactly the same role in $\sin(\frac{\pi m j}{J})$ so that the discrete analog of the completeness relation (4) holds as a direct consequence of (7) by relabelling j and m:

$$\sum_{m=1}^{J-1} \sin(\frac{\pi m j}{J}) \sin(\frac{\pi m j'}{J}) = \frac{J}{2} \delta_{jj'} \tag{8}$$

Equations (7) and (8) are the same thing.

Cosine Transform

Given a function $u(x)$ on the interval $[0, \ell]$, the cosine transform and its inverse are given by:

$$\textit{Cosine Transform:} \quad a_m = \frac{2}{\ell} \int_0^\ell u(x) \cos(\pi m x/\ell) dx \tag{9}$$

$$\textit{Inverse Cosine Transform:} \quad u(x) = \frac{a_0}{2} + \sum_{m=1}^{\infty} a_m \cos(\pi m x/\ell) \tag{10}$$

$$= \sum_{m=0}^{\infty} w_m a_m \cos(\pi m x/\ell) \tag{11}$$

where the weight function w_m is given by:

$$w_j = \begin{cases} \frac{1}{2} & \text{if } j = 0 \\ 1 & \text{if } j > 0 \end{cases} \tag{12}$$

The proof that these are indeed inverses uses the orthogonality relation for cosine functions.

$$\int_0^\ell \cos(\pi m x/\ell) \cos(\pi m' x/\ell) dx = \frac{\ell}{2} \delta_{mm'} \tag{13}$$

which again is easy to prove by direct integration. These is also a difficult to prove completeness relation:

$$\sum_{m=0}^{\infty} w_m \cos(\pi m x/\ell) \cos(\pi m x'/\ell) = \frac{\ell}{2} \delta(x - x') \tag{14}$$

Discrete Cosine Transform

For the discrete cosine transform (DST) we consider a set of value U_j for $j = 0, \ldots, J$ again generally thought of as a discrete sampling of a function $u(x)$ on $[0, \ell]$. This time however we sample the function at the end points $j = 0$ and $j = J$. Then the discrete cosine transform (DST) and its inverse are given by:

$$\textit{DCT:} \quad a_m = \frac{2}{J} \sum_{j=0}^{J} w_j U_j \cos(\pi m j / J) \tag{15}$$

$$\textit{Inverse DCT:} \quad U_j = \sum_{m=0}^{J} w_m a_m \cos(\pi m j / J) \tag{16}$$

$$\tag{17}$$

where now the weight function is:

$$w_j = \begin{cases} \frac{1}{2} & \text{if } j = 0, J, \\ 1 & \text{if } 0 < j < J \end{cases} \tag{18}$$

One may think of (15) as arising from (9) by replacing the integral by a trapezoid approximation. The actual proof that the above pairs are inverses comes from the following identity.

$$\sum_{j=0}^{J} w_j w_m \cos(\frac{\pi m' j}{J}) \cos(\frac{\pi m j}{J}) = \frac{J}{2} \delta_{mm'} \tag{19}$$

Note that as the the DST, m and j play exactly the same role in this equation so that the discrete analog of the completeness relation (14) holds as a direct consequence of (19) by relabelling j and m.

Fourier Transform

Given a function $u(x)$ on the interval $[0, \ell]$, the Fourier transform (FT) and its inverse are given by:

$$\textit{Fourier Transform:} \quad a_m = \frac{1}{\ell} \int_0^\ell u(x) e^{-i2\pi m x/\ell} dx \tag{20}$$

$$\textit{Inverse Fourier Transform:} \quad u(x) = \sum_{m=-\infty}^{\infty} a_m e^{i2\pi m x/\ell} \tag{21}$$

In this case the function $u(x)$ is often allowed to be complex, but we shall consider only the case of real $u(x)$.

One has the following orthogonality relation for complex exponentials:

$$\int_0^\ell e^{i2\pi m x/\ell} e^{-i2\pi m' x/\ell} dx = \ell \delta_{mm'} \tag{22}$$

which as usual is easy to prove by direct integration.

(Notice that here we have $\int_0^\ell \phi_m(x) \phi^*_{m'}(x) dx = \ell \delta_{mm'}$, where $\phi_m(x) = e^{i2\pi m x/\ell}$.)

One can also use the completeness relation:

$$\sum_{m=-\infty}^{\infty} e^{i2\pi m x/\ell} e^{-i2\pi m x'/\ell} = \ell \delta(x - x') \tag{23}$$

which is not easy to prove.

In the case which $u(x)$ is a real-valued function, the a_m will still be complex, but the following symmetry holds:

$$a_{-m} = a^*_m \tag{24}$$

where $*$ denotes complex conjugation. Proof:

$$a^*_m = \left[\frac{1}{\ell} \int_0^\ell u(x) e^{-i2\pi m x/\ell} dx \right]^* = \frac{1}{\ell} \int_0^\ell u^*(x) e^{i2\pi m x/\ell} dx = \frac{1}{\ell} \int_0^\ell u(x) e^{-i2\pi(-m)x/\ell} dx = a_{-m}$$

Using this fact, it is possible to re-write the FT pair in such a way that all quantities are real:

$$\textit{Inverse Fourier Transform:} \quad u(x) = \frac{A_0}{2} + \sum_{m=1}^{\infty} \left(A_m \cos(2\pi m x/\ell) + B_m \sin(2\pi m x/\ell) \right)$$

with:

$$\textit{Fourier Transform:} \quad A_m = 2a^r_m = \frac{2}{\ell} \int_0^\ell u(x) \cos(2\pi m x/\ell) dx$$

$$B_m = -2a^i_m = \frac{2}{\ell} \int_0^\ell u(x) \sin(2\pi m x/\ell) dx$$

where a^r_m and a^i_m are the real and imaginary parts of a_m: $a_m = a^r_m + i a^i_m$.

Hence the FT of a real-valued function is equivalent to taking both sine and cosine transforms on the interval $[0, \ell]$ but with argument $2\pi m x/\ell$ rather than $\pi m x/\ell$.

The details are left for the reader, but the first few lines of the derivation are:

$$\begin{aligned} u(x) &= a_0 + \sum_{m=1}^{\infty} a_m e^{i2\pi m x/\ell} + \sum_{m=-1}^{-\infty} a_m e^{i2\pi m x/\ell} \\ &= a_0 + \sum_{m=1}^{\infty} a_m e^{i2\pi m x/\ell} + \sum_{m=1}^{\infty} a_{-m} e^{-i2\pi m x/\ell} \\ &= a_0 + \sum_{m=1}^{\infty} \left(a_m e^{i2\pi m x/\ell} + a^*_m e^{-i2\pi m x/\ell} \right) \end{aligned}$$

Discrete Fourier Transform

For the discrete Fourier transform (DFT) we again consider a set of value U_j for $j = 0, \ldots, J$ thought of as a discrete sampling of a function $u(x)$ on $[0, \ell]$ including end points $j = 0$ and $j = J$. In this case we are

going to consider several forms for the DFT. The first is given by:

$$\textit{DFT:}\quad a_m = \frac{1}{J}\sum_{j=0}^{J-1} U_j e^{-i2\pi mj/J}, \quad m = -J/2+1, -J/2+2, \ldots, J/2, \tag{25}$$

$$\textit{Inverse DFT:}\quad U_j = \sum_{m=-J/2+1}^{J/2} a_m e^{i2\pi mj/J} \quad j = 0, 1, \ldots, J-1, \tag{26}$$

where we have assumed J is even. This is the form we shall consider most frequently in this course.

The proof that the above pairs are inverses comes from the orthogonality of the complex exponentials:

$$\sum_{j=0}^{J-1} e^{-i2\pi mj/J} e^{i2\pi m'j/J} = J\delta_{mm'} \tag{27}$$

As with the other discrete transforms, this is also equivalent to the completeness relation in the continuous case.

Other standard and non-standard forms of the DFT can be derived by noting that if (25) is used to define a_m for all integer m, then the a_m are periodic with period J. Similarly, (26) gives periodic U_j with period J.

$$a_{m+kJ} = a_m \quad \text{and} \quad U_{j+kJ} = U_j$$

for all integer k. Proof:

$$\begin{aligned} a_{m+kJ} &= \frac{1}{J}\sum_{j=0}^{J-1} U_j e^{-i2\pi(m+kJ)j/J} \\ &= \frac{1}{J}\sum_{j=0}^{J-1} U_j e^{-i2\pi mj/J} e^{-i2\pi kj} \\ &= \frac{1}{J}\sum_{j=0}^{J-1} U_j e^{-i2\pi mj/J} = a_m \end{aligned}$$

Using this, the following is an equivalent form for the DFT and its inverse:

$$\textit{DFT:}\quad a_m = \frac{1}{J}\sum_{j=0}^{J} w_j U_j e^{-i2\pi mj/J}, \quad m = -J/2, -J/2+1, \ldots, J/2, \tag{28}$$

$$\textit{Inverse DFT:}\quad U_j = \sum_{m=-J/2}^{J/2} \hat{w}_m a_m e^{i2\pi mj/J} \quad j = 0, 1, \ldots, J-1, \tag{29}$$

where:

$$w_j = \begin{cases} \frac{1}{2} & \text{if } j = 0, J, \\ 1 & \text{if } 0 < j < J \end{cases} \tag{30}$$

and

$$\hat{w}_m = \begin{cases} \frac{1}{2} & \text{if } m = -J/2, J/2, \\ 1 & \text{if } -J/2 < m < J/2 \end{cases} \tag{31}$$

In effect, since $U_0 = U_J$, U_0 in (25) can been replaced by $(U_0 + U_J)/2$. Similarly for (29) since $a_{-J/2} = a_{J/2}$.

This is a non-standard form for the DFT. I give it here because it is the form one might guess from the continuous FT (20)-(21).

Probably the most common form for the DFT is given by using the periodicity in a_m to shift the sum in (26) and obtain:

$$\textit{DFT:}\quad a_m = \frac{1}{J}\sum_{j=0}^{J-1} U_j e^{-i2\pi mj/J}, \quad m = 0, 1, \ldots, J-1, \tag{32}$$

$$\textit{Inverse DFT:}\quad U_j = \sum_{m=0}^{J-1} a_m e^{i2\pi mj/J} \quad j = 0, 1, \ldots, J-1, \tag{33}$$

This form has the advantage that both sums are over the same range. However, it has the disadvantages that (1) it is not a natural choice given the continuous FT, (2) more importantly, we will consider real U_j and this form does not make it as obvious that U_j obtained by the inverse transformation will be real.

Most libraries that perform discrete Fourier transforms say they compute transforms of type (32)-(33). They are all equivalent of course.

Finally, we consider the sine/cosine forms of the DFT. As in the continuous case: U_j real implies $a_{-m} = a_m^*$. Then (26) gives:

$$U_j = \sum_{m=-J/2+1}^{J/2} a_m e^{i2\pi mj/J} \tag{34}$$

$$= a_0 + \sum_{m=1}^{J/2-1} a_m e^{i2\pi mj/J} + \sum_{m=-1}^{-(J/2-1)} a_m e^{i2\pi mj/J} + a_{J/2} e^{i2\pi (J/2)j/J} \tag{35}$$

$$= a_0 + \sum_{m=1}^{J/2-1} \left(a_m e^{i2\pi mj/J} + a_m^* e^{-i2\pi mj/J} \right) + a_{J/2} e^{i\pi j} \tag{36}$$

$$= a_0 + \sum_{m=1}^{J/2-1} \left(2a_m^r \cos(2\pi mj/J) + (-2a_m^i) \sin(2\pi mj/J) \right) + a_{J/2} \cos(\pi mj/(J/2)) \tag{37}$$

$$= \frac{A_0}{2} + \sum_{m=1}^{J/2-1} \left(A_m \cos(2\pi mj/J) + B_m \sin(2\pi mj/J) \right) + \frac{A_{J/2}}{2} \cos(\pi mj/(J/2)) \tag{38}$$

So

$$\textit{Inverse Fourier Transform:}\quad U_j = \sum_{m=0}^{J/2} \left(w_m A_m \cos(2\pi mj/J) + B_m \sin(2\pi mj/J) \right)$$

where

$$w_m = \begin{cases} \frac{1}{2} & \text{if } m = 0, J/2, \\ 1 & \text{if } 0 < m < J/2 \end{cases} \tag{39}$$

and A_m and B_m are given by:

$$\textit{Fourier Transform:}\quad A_m = 2a_m^r = \frac{2}{J}\sum_{j=0}^{J} w_j U_j \cos(2\pi mj/J)$$

$$B_m = -2a_m^i = \frac{2}{J}\sum_{j=1}^{J-1} U_j \sin(2\pi mj/J)$$

Final remark on counting. It is important to realize that the DFT is simply a change of bases. In the case of complex U_j it is a change of bases in C^J. In the case of real U_j, our case, it is equivalent to a change of bases in R^J. That the counting is correct for this real case (so called real-to-complex-transform) can be seen by the as follows. Referring to (25), given J real values U_j, $j = 0, 1, \ldots, J-1$, we obtain J complex amplitudes a_m, $m = -J/2+1, -J/2+2, \ldots, J/2$. However, there is a symmetry in the a_m, namely $a_{-m} = a_m^*$. Hence the negative half of the spectrum: $a_{-1}, a_{-2}, \ldots, a_{-J/2+1}$ is known given the positive half and thus is should not be counted. This leaves us with the $J/2+1$ values $a_0, a_1, \ldots, a_{J/2}$. However, a_0 is necessarily real because $a_0^* = a_{-0} = a_0$. Similarly $a_{J/2}$ is necessarily real (try to show this). Hence there are precisely J distinct real quantities in the a_m $m = -J/2+1, -J/2+2, \ldots, J/2$. These are $a_0^r, a_1^r, a_1^i, \ldots, a_{J/2-1}^r, a_{J/2-1}^i, a_{J/2}^r$.

Two and More Space Dimensions

The heat equation in two space dimensions is:

$$\frac{\partial u}{\partial t} = \nabla^2 u$$

where

$$\begin{aligned} \nabla^2 u &= \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) u, \quad \text{in Cartesian coordinates} \\ &= \left(\frac{\partial^2}{\partial r^2} + \frac{1}{r}\frac{\partial}{\partial r} + \frac{1}{r^2}\frac{\partial^2}{\partial \theta^2} \right) u, \quad \text{in polar coordinates} \end{aligned}$$

The finite-difference approach to such an equation is, in principle, very similar to that used in one space dimension:

Discretize u and spatial operator(s): $u(x,y) \to U_{jk}$, $\nabla^2 \to \mathbf{L}$

Discretize time: $\frac{\partial}{\partial t} \to$ FE, BE, CN, ...

Solve resulting algebraic equations efficiently

New difficulties:

1. Geometry and Boundary conditions
2. Grids
3. Solving large linear systems of equations.

Comments on Testing and Software Design

Suggestions for Testing

- **Test parts of your program that cannot possibly be wrong**.
- **Test scaling**.

  Even without exact solutions one can test convergence rates. Use a very high resolution solution as "exact" even if it may not be so.
- **If possible test linear and nonlinear part of the code separately**.
- **Linear Operators**.

  Generally straightforward but here are some hints.

  - Build up your operators in pieces that can be tested independently if necessary.
  - It is usually easy to choose some test functions v and verify that, for example, $(\mathbf{I} + \triangle t\mathbf{L})\mathbf{V}$ converges to $(1 + \triangle t\mathcal{L})v$ with the correct scaling.
  - Verify that all linear operators are linear.
  - Verify matrix inverses. Note that $(\mathbf{I} + \triangle t\mathbf{L})$ and $(\mathbf{I} - \triangle t\mathbf{L})^{-1}$ are inverses of one another apart from the sign of $\triangle t$. A forward Euler time step of size $\triangle t$ followed by a backward Euler time step with $(-\triangle t)$ should be the identity:

    $$\mathbf{U} = (\mathbf{I} - (-\triangle t)\mathbf{L})^{-1}\,(\mathbf{I} + \triangle t\mathbf{L})\mathbf{U}$$

    for any $\mathbf{U}$. This will verify that the explicit and implicit parts of a code are at least consistent. *I strongly recommend this test.*
- **Exact solutions**.

  Exact solutions are great, but even without one you can (usually) test in at least two ways.

  - Change the problem.
    Given a nonlinear equation

    $$\frac{\partial u}{\partial t} = \mathcal{L}u + \mathcal{N}(u),$$

    choose a suitable test function u_T for which one can analytically compute

    $$g = -(\mathcal{L}u_T + \mathcal{N}(u_T)).$$

    Add this *constant forcing function* to the right-hand-side and solve the PDE:

    $$\frac{\partial u}{\partial t} = \mathcal{L}u + \mathcal{N}(u) + g.$$

    u_T will necessarily be an exact steady solution to this equation.

  - Use your pencil or your favorite algebraic manipulation package.
    Choose a suitable function v and work out the effect of one time step on this function. Verify that the code does as expected over one time step.
    Often very trivial functions such as linear or constant functions can provide limited testing. These are surprisingly good at finding mistakes in boundary conditions.

  These are powerful, very common, yet largely un-publicized method of testing.
- **Benchmarks**.

  Test against benchmarks and other code if available.
- **Test parts of your program that cannot possibly be wrong**.
- **Leave testing code in place**.

  You will need if more often than you think. Keep in a separate file if you wish.

  For testing strive for complete clarity, not cleverness. Computational cost is not an issue for testing. Produce a documented test suit which can be easily re-run frequently.

Software Issues

Some thought to consider in writing programs to solve PDEs.

- Speed vs readability/maintainability.

  Computational speed may be sacrificed for readability, reliability, maintainability, and development time. This is especially true during initial development. Start with good readability/maintainability. Strive for efficiency in algorithms, but do not necessarily strive for the absolute minimum number of floating point operations per time step.

  Later when computation speed becomes a real limit (particularly for "production" runs) you can improve efficiency and reduce computations to the minimum number of operations. Improvement can usually made in a working code. The slow-but-sure code then can serve as a benchmark for the improved code.

  It is a good idea to comment known inefficiencies while you write code.

  Note, speed in not simply proportional to the number of computations performed. Memory access is a major issue.
- My rule of thumb: a factor of 2 in speed is not worth worrying about if it means a large sacrifice in any of the other factors: readability, reliability, maintainability.
- Generality vs readability/maintainability.

  Do not try solve all problems with one code (especially if you write object oriented code). In particular do not try to solve problems you will never encounter.
- Libraries and portability: when to use BLAS/LAPACK libraries and when to write vs self-contained codes.

  Using local libraries (libraries specific the hardware) has the advantage of speed. Furthermore using the libraries in general can reduce errors and also reduce debugging – there are no bugs in BLAS and you need not look there for errors.

  The advantage of self-contained codes written in standard C++ is that you are guaranteed portability and porting is generally easier.

