Chapter 7

Approximating Functions

This chapter discusses two related problems. One is finding simple approximate representations for known functions. The other is interpolation and extrapolation, estimating unknown function values from known values at nearby points. On one hand, interpolation of smooth functions gives accurate approximations. On the other hand, we can interpolate and extrapolate using our approximating functions.

Some useful interpolating functions are polynomials, splines, and trigonometric polynomials. Interpolation by low order polynomials is simple and ubiquitous in scientific computing. Ideas from Chapters 3 and 4 will let us understand its accuracy. Some simple tricks for polynomial interpolation are the Newton form of the interpolating polynomial and Horner's rule for evaluating polynomials.

Local polynomial interpolation gives different approximating functions in different intervals. A spline interpolant is a single globally defined function that has many of the approximation properties of local polynomial interpolation. Computing the interpolating spline from n data points requires us to solve a linear system of equations involving a symmetric banded matrix, so the work is proportional to n.

The order of accuracy of polynomial or spline interpolation is $p + 1$, where p is the degree of polynomials used. This suggests that we could get very accurate approximations using high degree polynomials. Unfortunately, high degree polynomial interpolation on uniformly spaced points leads to linear systems of equations that are exponentially ill conditioned, $\kappa \sim e^{cp}$, where p is the degree and κ is the condition number. The condition number grows moderately as $p \to \infty$ only if the interpolation points cluster at the ends of the interval in a very specific way.

High accuracy interpolation on uniformly spaced points can by done using trigonometric polynomial interpolation, also called Fourier interpolation. More generally, Fourier analysis for functions defined at n uniformly spaced points can be done using the discrete Fourier transform, or DFT. The fast Fourier transform, or FFT, is an algorithm that computes the DFT of n values in $O(n \log(n))$ time. Besides trigonometric interpolation, the FFT gives highly accurate solutions to certain linear partial differential equations and allows us to compute large discrete convolutions, including the convolution that defines the time lag covariance function for a time series.

7.1 Local polynomial interpolation

Given points $x_0, \ldots, x_d$ and values $f_0, \ldots, f_d$, there is a unique *interpolating* polynomial of degree d,

$$p(x) = p_0 + p_1 x + \cdots + p_d x^d \,, \tag{7.1}$$

so that

$$p(x_k) = f_k \quad \text{for } k = 0, 1, \ldots, d. \tag{7.2}$$

We give several proofs of this, each one illustrating a different aspect of polynomial interpolation. The reader may wish to skip the proofs at first reading.

7.1.1 vanderMonde theory

The most direct approach uses the vanderMonde matrix. The equations (7.19) and (7.2) form a set of linear equations that determine the $d+1$ unknown coefficients, p_j, from the $d+1$ given function values, f_k. The k^{th} equation is

$$p_0 + x_k p_1 + x_k^2 p_2 + \cdots + x_k^d p_d = f_k \ ,$$

which we write abstractly as

$$Vp = f \ , \tag{7.3}$$

where

$$V = \begin{pmatrix} 1 & x_0 & . & . & . & x_0^d \\ 1 & x_1 & . & . & . & x_1^d \\ \vdots & \vdots & & & & \vdots \\ 1 & x_d & . & . & . & x_d^d \end{pmatrix} \ , \tag{7.4}$$

$p = (p_0, \ldots, p_d)^*$, and $f = (f_0, \ldots, f_d)^*$. The equations (7.3) have a unique solution if and only if $\det(V) \neq 0$. We show $\det(V) \neq 0$ using the following famous formula:

Theorem 2 *Define $D(x_0, \ldots, x_d) = \det(V)$ as in (7.4). Then*

$$D(x_0, \ldots, x_d) = \prod_{j<k} (x_k - x_j) \ . \tag{7.5}$$

The reader should verify directly that $D(x_0, x_1, x_2) = (x_2 - x_0)(x_1 - x_0)(x_2 - x_1)$. It is clear that $D = 0$ whenever $x_j = x_k$ for some $j \neq k$ because $x_j = x_k$ makes row j and row k equal to each other. The formula (7.5) says that D is a product of factors coming from these facts.

Proof: The proof uses three basic properties of determinants. The first is that the determinant does not change if we perform an elimination operation on rows or columns. If we subtract a multiple of row j from row k or of column j from column k, the determinant does not change. The second is that if row k or column k has a common factor, we can pull that factor out of the determinant. The third is that if the first column is $(1, 0 \ldots, 0)^*$, then the determinant is the determinant of the $d \times d$ matrix got by deleting the top row and first column.

We work by induction on the number of points. For $d = 1$ (7.5) is $D(x_0, x_1) = x_1 - x_0$, which is easy to verify. The induction step is the formula

$$D(x_0, \ldots, x_d) = \left(\prod_{k=1}^{d} (x_k - x_0) \right) \cdot D(x_1, \ldots, x_d) \ . \tag{7.6}$$

We use the easily checked formula

$$x^k - y^k = (x - y)(x^{k-1} + x^{k-2} y + \cdots + y^{k-1}) \ . \tag{7.7}$$

To compute the determinant of V in (7.4), we use Gauss elimination to set all but the top entry of the first column of V to zero. This means that we replace row j by row j minus row 1. Next we find common factors in the columns. Finally we perform column operations to put the $d \times d$ matrix back into the form of a vanderMonde matrix for $x_1, \ldots, x_d$, which will prove (7.6).

Rather than giving the argument in general, we give it for $d = 2$ and $d = 3$. The general case will be clear from this. For $d = 2$ we have

$$\begin{aligned} \det \begin{pmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \end{pmatrix} &= \det \begin{pmatrix} 1 & x_0 & x_0^2 \\ 0 & x_1 - x_0 & x_1^2 - x_0^2 \\ 0 & x_2 - x_0 & x_2^2 - x_0^2 \end{pmatrix} \\ &= \det \begin{pmatrix} x_1 - x_0 & x_1^2 - x_0^2 \\ x_2 - x_0 & x_2^2 - x_0^2 \end{pmatrix} . \end{aligned}$$

The formula (7.7) with $k = 2$ gives $x_1^2 - x_0^2 = (x_1 - x_0)(x_1 + x_0)$, so $(x_1 - x_0)$ is a common factor in the top row. Similarly, $(x_2 - x_0)$ is a common factor of the bottom row. Thus:

$$\begin{aligned} \det \begin{pmatrix} x_1 - x_0 & x_1^2 - x_0^2 \\ x_2 - x_0 & x_2^2 - x_0^2 \end{pmatrix} &= \det \begin{pmatrix} x_1 - x_0 & (x_1 - x_0)(x_1 + x_0) \\ x_2 - x_0 & x_2^2 - x_0^2 \end{pmatrix} \\ &= (x_1 - x_0) \det \begin{pmatrix} 1 & (x_1 + x_0) \\ x_2 - x_0 & x_2^2 - x_0^2 \end{pmatrix} \\ &= (x_1 - x_0)(x_2 - x_0) \det \begin{pmatrix} 1 & x_1 + x_0 \\ 1 & x_2 + x_0 \end{pmatrix} . \end{aligned}$$

The final step is to subtract x_0 times the first column from the second column, which does not change the determinant:

$$\begin{aligned} \det \begin{pmatrix} 1 & x_1 + x_0 \\ 1 & x_2 + x_0 \end{pmatrix} &= \det \begin{pmatrix} 1 & x_1 + x_0 - x_0 * 1 \\ 1 & x_2 + x_0 - x_0 * 1 \end{pmatrix} \\ &= \det \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \end{pmatrix} \\ &= D(x_1, x_2) . \end{aligned}$$

This proves (7.6) for $d = 2$.

For $d = 3$ there is one more step. If we subtract row 1 from row k for $k > 1$ and do the factoring using (7.7) for $k = 2$ and $x = 3$, we get

$$\det \begin{pmatrix} 1 & x_0 & x_0^2 & x_0^3 \\ 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \end{pmatrix} =$$

$$(x_1 - x_0)(x_2 - x_0)(x_3 - x_0) \det \begin{pmatrix} 1 & x_1 + x_0 & x_1^2 + x_1 x_0 + x_0^2 \\ 1 & x_2 + x_0 & x_2^2 + x_2 x_0 + x_0^2 \\ 1 & x_3 + x_0 & x_3^2 + x_3 x_0 + x_0^2 \end{pmatrix} .$$

We complete the proof of (7.6) in this case by showing that

$$\det \begin{pmatrix} 1 & x_1 + x_0 & x_1^2 + x_1 x_0 + x_0^2 \\ 1 & x_2 + x_0 & x_2^2 + x_2 x_0 + x_0^2 \\ 1 & x_3 + x_0 & x_3^2 + x_3 x_0 + x_0^2 \end{pmatrix} = \det \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{pmatrix} .$$

For this we first subtract x_0 times the first column from the second column, then subtract x_0^2 times the first column from the third column, then subtract x_0 times the second column from the third column. This completes the proof of Theorem 2 and shows that one can find the coefficients of the interpolating polynomial by solving a linear system of equations involving the vanderMonde matrix.

7.1.2 Newton interpolation formula

The Newton interpolation formula is a simple and insightful way to express the interpolating polynomial. It is based on repeated divided differences, done in a way to expose the leading terms of polynomials. These are combined with a specific basis for the vector space of polynomials of degree k so that in the end the interpolation property is obvious. In some sense, the Newton interpolation formula provides a formula for the inverse of the vanderMonde matrix.

We begin with the problem of estimating derivatives of $f(x)$ using a number of function values. Given nearby points, x_1 and x_0, we have

$$f' \approx \frac{f(x_1) - f(x_0)}{x_1 - x_0} .$$

We know that the divided difference is particularly close to the derivative at the center of the interval, so we write

$$f'\left(\frac{x_1 + x_0}{2}\right) \approx \frac{f(x_1) - f(x_0)}{x_1 - x_0} , \tag{7.8}$$

with an error that is $O\left(|x_1 - x_0|^2\right)$. If we have three points that might not be uniformly spaced, the second derivative estimate using (7.8) could be

$$\begin{aligned} f'' &\approx \frac{f'\left(\frac{x_2+x_1}{2}\right) - f'\left(\frac{x_1+x_0}{2}\right)}{\frac{x_2 + x_1}{2} - \frac{x_1 + x_0}{2}} \\ f'' &\approx \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{\frac{1}{2}(x_2 - x_0)} . \end{aligned} \tag{7.9}$$

As we saw in Chapter 3, the approximation (7.9) is consistent (converges to the exact answer as $x_2 \to x$, $x_1 \to x$, and $x_0 \to x$) if it is exact for quadratics, $f(x) = ax^2 + bx + c$. Some algebra shows that both sides of (7.9) are equal to $2a$.

The formula (7.9) suggests the right way to do repeated divided differences. Suppose we have $d+1$ points[1] $x_0, \ldots, x_d$, we define $f[x_k] = f(x_k)$ (exchange round parentheses for square brackets), and the first order divided difference is:

$$f[x_k, x_{k+1}] = \frac{f[x_{k+1}] - f[x_k]}{x_{k+1} - x_k} \ .$$

More generally, the *Newton divided difference* of order $k+1$ is a divided difference of divided differences of order k:

$$f[x_j, \cdots, x_{k+1}] = \frac{f[x_{j+1}, \cdots, x_{k+1}] - f[x_j, \cdots, x_k]}{x_{k+1} - x_j} \ . \tag{7.10}$$

The denominator in (7.10) is the difference between the extremal x values, as (7.9) suggests it should be. If instead of a function $f(x)$ we just have values $f_0, \ldots, f_d$, we define

$$[f_j, \cdots, f_{k+1}] = \frac{[f_{j+1}, \cdots, f_{k+1}] - [f_j, \cdots, f_k]}{x_{k+1} - x_j} \ . \tag{7.11}$$

It may be convenient to use the alternative notation

$$D_k(f) = f[x_0, \cdots, x_k] \ .$$

If $r(x) = r_k x^k + \cdots + r_0$ is a polynomial of degree k, we will see that $D_k r = k! \cdot r_k$. We verified this already for $k = 1$ and $k = 2$.

The interpolation problem is to find a polynomial of degree d that satisfies the interpolation conditions (7.2). The formula (7.1) expresses the interpolating polynomial as a linear combination of pure monomials x^k. Using the monomials as a basis for the vector space of polynomials of degree d leads to the vander-Monde matrix (7.4). Here we use a different basis, which might be called the *Newton monomials* of degree k (although they strictly speaking are not monomials), $q_0(x) = 1$, $q_1(x) = x - x_0$, $q_2(x) = (x - x_1)(x - x_0)$, and generally,

$$q_k(x) = (x - x_{k-1}) \cdot \cdots \cdot (x - x_0) \ . \tag{7.12}$$

It is easy to see that $q_k(x)$ is a polynomial of degree k in x with leading coefficient equal to one:

$$q_k(x) = x^k + a_{k-1} x^{k-1} + \cdots \ .$$

Since this also holds for q_{k-1}, we may subtract to get:

$$q_k(x) - a_{k-1} q_{k-1}(x) = x^k + b_{k-2} x^{k-2} + \cdots \ .$$

Continuing in this way, we express x^k in terms of Newton monomials:

$$x^k = q_k(x) - a_{k,k-1} q_{k-1}(x) - b_{k,k-2} - \cdots \ . \tag{7.13}$$

[1] The x_k must be distinct but they need not be in order. Nevertheless, it helps the intuition to think that $x_0 < x_1 < \cdots < x_d$.

This shows that the $q_k(x)$ are linearly independent and span the same space as the monomial basis.

The connection between repeated divided differences (7.10) and Newton monomials (7.12) is

$$D_k q_j = \delta_{kj} \ . \tag{7.14}$$

The intuition is that $D_k f$ plays the role $\frac{1}{k!}\partial_x^k f(0)$ and $q_j(x)$ plays the role of x_j. For $k > j$, $\partial_k x^j = 0$ because differentiation lowers the order of a monomial. For $k < j$, $\partial_x^k x^j = 0$ when evaluated at $x = 0$ because monomials vanish when $x = 0$. The remaining case is the interesting one, $\frac{1}{k!}\partial_x^k x^k = 1$.

We verify (7.14) by induction on k. We suppose that (7.14) holds for all $k < d$ and all j and use that to prove it for $k = d$ and all j, treating the cases $j = d$, $j < d$, and $j > d$ separately. The base case $k = 1$ explains the ideas. For $j = 1$ we have

$$D_1 q_1(x) = \frac{q_1(x_1) - q_1(x_0)}{x_1 - x_0} = \frac{(x_1 - x_0) - (x_0 - x_0)}{x_1 - x_0} = 1 \ , \tag{7.15}$$

as claimed. More generally, any first order divided difference of q_1 is equal to one,

$$q_1[x_{k+1}, x_k] = \frac{q_1(x_{k+1}) - q_1(x_k)}{x_{k+1} - x_k} = 1 \ ,$$

which implies that higher order divided differences of q_1 are zero. For example,

$$q_1[x_2, x_3, x_4] = \frac{q_1[x_3, x_4] - q_1[x_2, x_3]}{x_4 - x_2} = \frac{1 - 1}{x_4 - x_2} = 0 \ .$$

This proves the base case, $k = 1$ and all j.

The induction step has the same three cases. For $j > d$ it is clear that $D_d q_j = q_j[x_0, \ldots, x_d] = 0$ because $q_j(x_k) = 0$ for all the x_k that are used in $q_j[x_0, \ldots, x_d]$. The interesting case is $q_k[x_0, \ldots, x_k] = 1$. From (7.10) we have that

$$q_k[x_0, \ldots, x_k] = \frac{q_k[x_1, \ldots, x_k] - q_k[x_0, \ldots, x_{k-1}]}{x_k - x_0} = \frac{q_k[x_1, \ldots, x_k]}{x_k - x_0} \ ,$$

because $q_k[x_0, \ldots, x_{k-1}] = 0$ (it involves all zeros). The same reasoning gives $q_k[x_1, \ldots, x_{k-1}] = 0$ and

$$q_k[x_1, \ldots, x_k] = \frac{q_k[x_2, \ldots, x_k] - q_k[x_1, \ldots, x_{k-1}]}{x_k - x_1} = \frac{q_k[x_2, \ldots, x_k]}{x_k - x_1} \ .$$

Combining these gives

$$q_k[x_0, \ldots, x_k] = \frac{q_k[x_1, \ldots, x_k]}{x_k - x_0} = \frac{q_k[x_2, \ldots, x_k]}{(x_k - x_0)(x_k - x_1)} \ ,$$

and eventually, using the definition (7.12), to

$$q_k[x_0, \ldots, x_k] = \frac{q_k(x_k)}{(x_k - x_0) \cdot \cdots \cdot (x_k - x_{k-1})} = \frac{(x_k - x_0) \cdot \cdots \cdot (x_k - x_{k-1})}{(x_k - x_0) \cdot \cdots \cdot (x_k - x_{k-1})} = 1 \ ,$$

as claimed. Now, using (7.13) we see that

$$D_k x^k = D_k\big(q_k - a_{k,k-1}q_{k-1} - \cdots\big) = D_k q_k = 1 \ ,$$

for any collection of distinct points x_j. This, in turn, implies that

$$q_k[x_{m+k}, \ldots, x_m] = 1 \ .$$

which, as in (7.15), implies that $D_{k+1}q_k = 0$. This completes the induction step.

The formula (7.14) allows us to verify the *Newton interpolation formula*, which states that

$$p(x) = \sum_{k=0}^{d} [f_0, \ldots, f_k]q_k(x) \ , \tag{7.16}$$

satisfies the interpolation conditions (7.2). We see that $p(x_0) = f_0$ because each term on the right $k = 0$ vanishes when $x = x_0$. The formula (7.14) also implies that $D_1 p = D_1 f$. This involves the values $p(x_1)$ and $p(x_0)$. Since we already know $p(x_0)$ is correct, this implies that $p(x_1)$ also is correct. Continuing in this way verifies all the interpolation conditions.

7.2 Discrete Fourier transform

The Fourier transform is one of the most powerful methods of applied mathematics. Its finite dimensional anologue, the *discrete Fourier transform*, or *DFT*, is just as useful in scientific computing. The DFT allows direct algebraic solution of certain differential and integral equations. It is the basis of computations with time series data and for digital signal processing and control. It leads to computational methods that have an infinite order of accuracy (which is not the same as being exact).

The drawback of DFT based methods is their geometric inflexibility. They can be hard to apply to data that are not sampled at uniformly spaced points. The multidimensional DFT is essentially a product of one dimensional DFTs. Therefore is it hard to apply DFT methods to problems in more than one dimension unless the computational domain has a simple shape. Even in one dimension, applying the DFT depends on boundary conditions.

7.2.1 Fourier modes

The simplest Fourier analysis is for periodic functions of one variable. We say $f(x)$ is *periodic* with *period* p if $f(x + p) = f(x)$ for all x. If α is an integer, then the *Fourier mode*

$$w_\alpha(x) = e^{2\pi i \alpha x/p} \tag{7.17}$$

is such a periodic function. Fourier analysis starts with the fact that Fourier modes form a basis for the vector space of periodic functions. That means that

if f has period p, then there are *Fourier coefficients*, $\widehat{f}_\alpha$, so that

$$f(x) = \sum_{\alpha=-\infty}^{\infty} \widehat{f}_\alpha e^{2\pi i \alpha x/p} \ . \tag{7.18}$$

More precisely, let V_p be the vector space of complex valued periodic functions so that

$$\|f\|_{L^2}^2 = \int_0^p |f(x)|^2 \, dx < \infty \ .$$

This vector space has an *inner product* that has the properties discussed in Section 4.2.2. The definition is

$$\langle f, g \rangle = \int_0^p \overline{f}(x) g(x) dx \ . \tag{7.19}$$

Clearly, $\|f\|_{L^2}^2 = \langle f, f \rangle > 0$ unless $f \equiv 0$. The inner product is linear in the g variable:

$$\langle f, ag_1 + bg_2 \rangle = a\langle f, g_1 \rangle + b\langle f, g_2 \rangle \ ,$$

and *antilinear* in the f variable:

$$\langle af_1 + bf_2, g \rangle = \overline{a}\langle f_1, g \rangle + \overline{b}\langle f_2, g \rangle \ .$$

Functions f and g are *orthogonal* if $\langle f, g \rangle = 0$. A set of functions is orthogonal if any two of them are orthogonal and none of them is zero. The Fourier modes (7.17) have this property: if $\alpha \neq \beta$ are two integers, then $\langle w_\alpha, w_\beta \rangle = 0$ (check this).

Any orthogonal system of functions is linearly independent. Linear independence means that if

$$f(x) = \sum_\alpha \widehat{f}_\alpha w_\alpha(x) = \sum_{\alpha=-\infty}^{\infty} \widehat{f}_\alpha e^{2\pi i \alpha x/p} \ , \tag{7.20}$$

then the $\widehat{f}_\alpha$ are uniquely determined. For orthogonal functions, we show this by taking the inner product of w_β with f and use the linearity of the inner product to get

$$\langle w_\beta, f \rangle = \sum_\alpha \widehat{f}_\alpha \langle w_\beta, w_\alpha \rangle = \widehat{f}_\beta \langle w_\beta, w_\beta \rangle \ ,$$

so

$$\widehat{f}_\beta = \frac{\langle w_\beta, f \rangle}{\|w_\beta\|^2} \ . \tag{7.21}$$

This formula shows that the coefficients are uniquely determined. Written more explicitly for the Fourier modes, (7.21) is

$$\widehat{f}_\alpha = \frac{1}{p} \int_{x=0}^p e^{-2\pi i \alpha x/p} f(x) dx \ . \tag{7.22}$$

An orthogonal family, w_α, is *complete* if any f has a representation in terms of them as a linear combination (7.20). An orthogonal family need not be complete. Fourier conjectured that the Fourier modes (7.17) are complete, but this was first proven several decades later.

Much of the usefulness of Fourier series comes from the relationship between the series for f and for derivatives f', f'', etc. When we differentiate (7.20) with respect to x and differentiate under the summation on the right side, we get

$$f'(x) = \frac{2\pi i}{p} \sum_\alpha \alpha \widehat{f}_\alpha e^{2\pi i \alpha x/p} \; .$$

This shows that the α Fourier coefficient of f' is

$$\widehat{f'}_\alpha = \frac{2\pi i \alpha}{p} \widehat{f}_\alpha \; . \tag{7.23}$$

Formulas like these allow us to express the solutions to certain ordinary and partial differential equations in terms of Fourier series. See Exercise 4 for one example.

We will see that the differentiation formula (7.23) also contains important information about the Fourier coefficients of smooth functions, that they are very small for large α. This implies that approximations

$$f(x) \approx \sum_{|\alpha| \leq R} \widehat{f}_\alpha w_\alpha(x)$$

are very accurate if f is smooth. It also implies that the DFT coefficients (see Section 7.2.2) are very accurate approximations of $\widehat{f}_\alpha$. Both of these make DFT based computational methods very attractive (when they apply) for application to problems with smooth solutions.

We start to see this decay by rewriting (7.23) as

$$\widehat{f}_\alpha = \frac{p\widehat{f'}_\alpha}{2\pi i} \cdot \frac{1}{\alpha} \; . \tag{7.24}$$

The integral formula

$$\widehat{f'}_\alpha = \frac{1}{p} \int_0^p \overline{w_\alpha(x)} f'(x) dx$$

shows that the Fourier coefficients $\widehat{f'}_\alpha$ are bounded if f' is bounded, since (for some real θ) $|w_\alpha(x)| = \left|e^{i\theta}\right| = 1$. This, and (7.24) shows that

$$\left|\widehat{f}_\alpha\right| \leq C \cdot \frac{1}{|\alpha|} \; .$$

We can go further by applying (7.24) to f' and f'' to get

$$\widehat{f}_\alpha = \frac{p^2 \widehat{f''}_\alpha}{-4\pi^2} \cdot \frac{1}{\alpha^2} \; ,$$

so that if f'' is bounded, then

$$\left|\widehat{f}_\alpha\right| \le C \cdot \frac{1}{|\alpha^2|} ,$$

which is faster decay ($1/\alpha^2 \ll 1/\alpha$ for large α). Continuing in this way, we can see that if f has N bounded derivatives then

$$\left|\widehat{f}_\alpha\right| \le C_N \cdot \frac{1}{|\alpha^N|} .$$

This shows, as we said, that the Fourier coefficients of smooth functions decay rapidly.

It is helpful in real applications to use real Fourier modes, particularly when $f(x)$ is real. The real modes are sines and cosines:

$$u_\alpha(x) = \cos(2\pi\alpha x/p) , \quad v_\alpha(x) = \sin(2\pi\alpha x/p) . \tag{7.25}$$

The u_α are defined for $\alpha \ge 0$ and the v_α for $\alpha \ge 1$. The special value $\alpha = 0$ corresponds to $u_0(x) = 1$ (and $v_0(x) = 0$). Exercise 3 shows that the u_α for $\alpha = 0, 1, \ldots$ and v_α for $\alpha = 1, 2, \ldots$ form an orthogonal family. The real Fourier series representation expresses f as a superposition of these functions:

$$f(x) = \sum_{\alpha=0}^{\infty} a_\alpha \cos(2\pi\alpha x/p) + \sum_{\alpha=1}^{\infty} b_\alpha \sin(2\pi\alpha x/p) . \tag{7.26}$$

The reasoning that led to (7.22), and the normalization formulas of Exercise **??** below, (7.40), gives

$$\left.\begin{aligned} a_\alpha &= \frac{2}{p}\int_0^p \cos(2\pi\alpha x/p) f(x)dx \ (\alpha \ge 1), \\ b_\alpha &= \frac{2}{p}\int_0^p \sin(2\pi\alpha x/p) f(x)dx \ (\alpha \ge 1), \\ a_0 &= \frac{1}{p}\int_0^p f(x)dx . \end{aligned}\right\} \tag{7.27}$$

This real Fourier (7.26) is basically the same as the complex Fourier series (7.20) when $f(x)$ is real. If f is real, then (7.22) shows that[2] $\widehat{f}_{-\alpha} = \overline{\widehat{f}}_\alpha$. This determines the $\alpha < 0$ Fourier coefficients from the $\alpha \ge 0$ coefficients. If $\alpha > 0$ and $\widehat{f}_\alpha = g_\alpha + ih_\alpha$ (g_α and h_α being real and imaginary parts), then (using $e^{i\theta} = \cos(\theta) + i\sin(\theta)$), we have

$$\begin{aligned} &\widehat{f}_\alpha e^{2\pi i\alpha x/p} + \widehat{f}_{-\alpha} e^{-2\pi i\alpha x/p} \\ &\quad = \quad (g_\alpha + ih_\alpha)\big(\cos(\cdot) + i\sin(\cdot)\big) + (g_\alpha - ih_\alpha)\big(\cos(\cdot) - i\sin(\cdot)\big) \\ &\quad = \quad 2g_\alpha \cos(\cdot) - 2h_\alpha \sin(\cdot) . \end{aligned}$$

[2]This also shows that the full Fourier series sum over positive and negative α is somewhat redundant for real f. This is another motivation for using the version with cosines and sines.

Moreover, when f is real,

$$\begin{aligned} g_\alpha &= \frac{1}{p}\mathrm{Re}\left[\int_0^p e^{-2\pi i\alpha x/p} f(x)dx\right] \\ &= \frac{1}{p}\int_0^p \cos(2\pi i\alpha x/p) f(x)dx \\ &= \frac{1}{2}a_\alpha \ . \end{aligned}$$

Similarly, $h_\alpha = \frac{-1}{2}b_\alpha$. This shows that the real Fourier series relations (7.20) and (7.22) directly follow from the complex Fourier series relations (7.26) and (7.27) without using Exercise 3.

7.2.2 The DFT

The DFT is a discrete anologue of Fourier analysis. The vector space V_p is replaced by sequences with period n: $f_{j+n} = f_j$. A periodic sequence is determined by the n entries[3]: $f = (f_0, \ldots, f_{n-1})^*$, and the vector space of such sequences is C^n. *Sampling* a periodic function of x is one way to create an element of C^n. Choose $\Delta x = p/n$, take *sample points*, $x_j = j\Delta x$, then the samples are $f_j = f(x_j)$. If the continuous f has period p, then the discrete sampled f has period n, because $f_{j+n} = f(x_{j+n})$, and $x_{j+n} = (j+n)\Delta x$, and $n\Delta x = p$.

The DFT modes come from sampling the continuous Fourier modes (7.17) in this way. That is

$$w_{\alpha,j} = w_\alpha(x_j) = \exp(2\pi i\alpha x_j/p) \ .$$

Since $x_j = jp/n$, this gives

$$w_{\alpha,j} = \exp(2\pi i\alpha j/n) = w^{\alpha j} \ , \tag{7.28}$$

where w is a *primitive root of unity*[4]

$$w = e^{2\pi i/n} \ . \tag{7.29}$$

Aliasing is a difference between continuous and discrete Fourier analysis. If $\beta = \alpha + n$ then the samplings of w_β and w_α are the same, $w_\beta(x_j) = w_\alpha(x_j)$ for all j, even though w_β and w_α are different functions of x. Figure 7.1 illustrates this with sine functions instead of complex exponentials. Aliasing implies that the discrete sampled vectors $w_\alpha \in C^n$ with components given by (7.28) are not all different. In fact, the n vectors w_α for $0 \le \alpha < n$ are distinct. After that they repeat: $w_{\alpha+n} = w_\alpha$ (this being an equality between two vectors in C^n).

These n discrete sampled modes form a basis, the DFT basis, for C^n. If f and g are two elements of C^n, the discrete inner product is

$$\langle f, g\rangle = \sum_{j=0}^{n-1} \overline{f}_j g_j \ .$$

[3]The * in $(f_0, \ldots, f_{n-1})^*$ indicates that we think of f as a column vector in C^n.

[4]*Unity* means the number 1. An n^{th} root of x is a y with $y^n = x$. An n^{th} root of unity is *primitive* if $w^n = 1$ but $w^k \neq 1$ for $0 \le k \le n$.

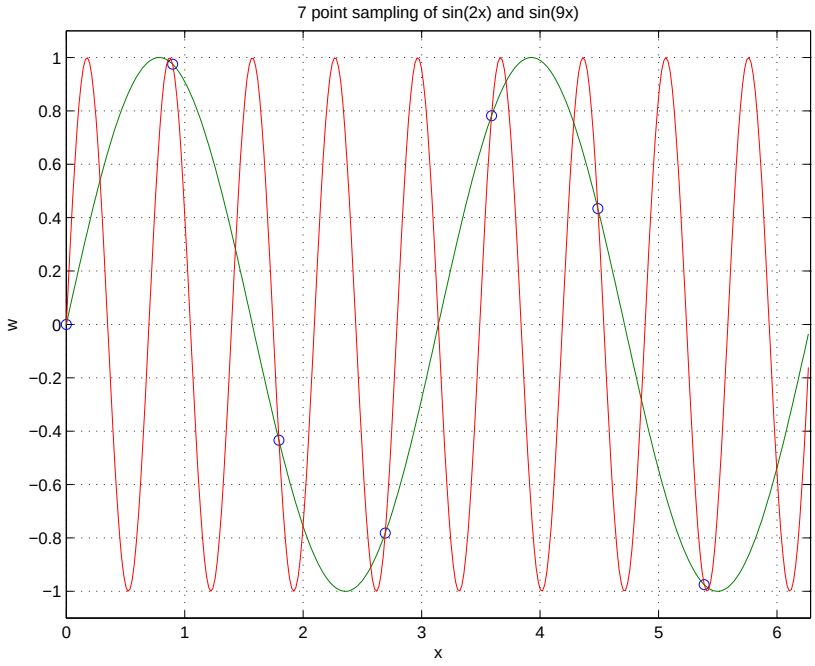

Figure 7.1: *An illustration of aliasing with* $n = 7$ *sampling points and period* $p = 2\pi$*, with modes* $\alpha = 2$ *and* $\beta = \alpha + n = 9$*. The continuous curves are* $\sin(2x)$ *and* $\sin(9x)$*. The circles are these curves sampled at the sample points* $x_j = 2\pi j/n$*. The sampled values are identical even though the functions* $\sin(2x)$ *and* $\sin(9x)$ *are not.*

This is the usual inner product on C^n and leads to the usual l^2 norm $\langle f, f\rangle = \|f\|_{l^2}^2$. We show that the discrete modes form an orthogonal family, but only as far as is allowed by aliasing. That is, if $0 \le \alpha < n$ and $0 \le \beta < n$, and $\alpha \ne \beta$, then $\langle w_\alpha, w_\beta\rangle = 0$.

Recall that for any complex number, z, $S(z) = \sum_{j=0}^{n-1} z^j$ has $S = n$ if $z = 1$ and $S = (z^n - 1)/(z - 1)$ if $z \ne 1$. Also,

$$\overline{w}_{\alpha,j} = \overline{e^{2\pi i\alpha j/n}} = e^{-2\pi i\alpha j/n} = w^{-\alpha j} \ ,$$

so we can calculate

$$\begin{aligned} \langle w_\alpha, w_\beta\rangle &= \sum_{j=0}^{n-1} w^{-\alpha j} w^{\beta j} \\ &= \sum_{j=0}^{n-1} w^{(\beta-\alpha)j} \\ &= \sum_{j=0}^{n-1} \left(w^{\beta-\alpha}\right)^j \end{aligned}$$

Under our assumptions ($\alpha \ne \beta$ but $0 \le \alpha < n$, $0 \le \beta < n$) we have $0 < |\beta - \alpha| < n$, and $z = w^{\beta-\alpha} \ne 1$ (using the fact that w is a *primitive* root of unity). This gives

$$\langle w_\alpha, w_\beta\rangle = \frac{w^{n(\beta-\alpha)} - 1}{w^{\beta-\alpha} - 1} \ .$$

Also,

$$w^{n(\beta-\alpha)} = \left(w^n\right)^{\beta-\alpha} = 1^{\beta-\alpha} = 1 \ ,$$

because w is an n^{th} root of unity. This shows $\langle w_\alpha, w_\beta\rangle = 0$. We also can calculate that $\|w_\alpha\|^2 = \sum_{j=1}^{n} |w_{\alpha,j}|^2 = n$.

Since the n vectors w_α for $0 \le \alpha < n$ are linearly independent, they form a basis of C^n. That is, any $f \in C^n$ may be written as a linear combination of the w_α:

$$f = \sum_{\alpha=0}^{n-1} \widehat{f}_\alpha w_\alpha \ .$$

By the arguments we gave for Fourier series above, the DFT coefficients are

$$\widehat{f}_\alpha = \frac{1}{n}\langle w_\alpha, f\rangle \ .$$

Expressed explicitly in terms of sums, these relations are

$$\widehat{f}_\alpha = \frac{1}{n}\sum_{j=0}^{n-1} w^{-\alpha j} f_j \ , \tag{7.30}$$

and

$$f_j = \sum_{\alpha=0}^{n-1} \widehat{f}_\alpha w^{\alpha j} \ . \tag{7.31}$$

These are the (*forward*) DFT and *inverse* DFT respectively. Either formula implies the other. If we start with the f_j and calculate the $\widehat{f}_\alpha$ using (7.30), then we can use (7.31) to recover the f_j from the $\widehat{f}_\alpha$, and the other way around. These formulas are more similar to each other than are the corresponding formulas (7.22) and (7.18). Both are sums rather than one being a sum and the other an integral.

There are many other slightly different definitions of the DFT relations. Some people define the discrete Fourier coefficients using a variant of (7.30), such as

$$\widehat{f}_\alpha = \sum_{j=0}^{n-1} w^{\alpha j} f_j \ .$$

This particular version changes (7.31) to

$$f_j = \frac{1}{n} \sum_{\alpha=0}^{n-1} w^{-\alpha j} \widehat{f}_\alpha \ .$$

Still another way to express these relations is to use the *DFT matrix*, W, which is an orthogonal matrix whose (α, j) entry is

$$w_{\alpha,j} = \frac{1}{\sqrt{n}} w^{-\alpha j} \ .$$

The adjoint of W has entries

$$w^*_{j,\alpha} = \overline{w}_{\alpha,j} = \frac{1}{\sqrt{n}} w^{\alpha j} \ .$$

The DFT relations are equivalent to $W^*W = I$. In vector notation, this $\widehat{f} = Wf$ transform differs from (7.30) by a factor of $\sqrt{n}$:

$$\widehat{f}_\alpha = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} w^{-\alpha j} f_j \ .$$

It has the advantage of making the direct and inverse DFT as similar as possible, with a factor $1/\sqrt{n}$ in both.

As for continuous Fourier series, there is a real discrete cosine and sine transform for real f. The complex coefficients $\widehat{f}_\alpha$, determine the real coefficients a_α and b_α as before. Aliasing is more complicated for the discrete sine and cosine transform, and depends on whether n is even or odd. The cosine and sine sums corresponding to (7.26) run from $\alpha = 0$ or $\alpha = 1$ roughly to $\alpha = n/2$.

We can estimate the Fourier coefficients of a continuous function by taking the DFT of its sampled values. We call the vector of samples $f^{(n)}$. It has

components $f_j^{(n)} = f(x_j)$, where $x_j = j\Delta x$ and $\Delta x = p/n$. The DFT coefficients $\widehat{f}_\alpha^{(n)}$ defined by (7.30) are rectangle rule approximations to the Fourier series coefficients (7.22). This follows from (7.28) and $\frac{1}{p}\Delta x = \frac{1}{n}$, since $\alpha j/n = \alpha x_j/p$ and

$$\begin{aligned} \widehat{f}_\alpha^{(n)} &= \frac{1}{n}\sum_{j=0}^{n-1} w^{-\alpha j} f_j^{(n)} \\ &= \frac{1}{p}\Delta x \sum_{j=0}^{n-1} e^{-2\pi i \alpha x_j/n} f(x_j) \ . \end{aligned} \tag{7.32}$$

There is a simple *aliasing formula* for the Fourier coefficients of $f^{(n)}$ in terms of those of f. It depends on aliasing when we put the continuous Fourier representation (7.22) into the discrete Fourier coefficient formula (7.33). If we rewrite (7.22) with β instead of α as the summation index, substitute into (7.33), and change the order of summation, we find

$$\widehat{f}_\alpha^{(n)} = \sum_{\beta=-\infty}^{\infty} \widehat{f}_\beta \frac{\Delta x}{p} \sum_{j=0}^{n-1} \exp\left[2\pi i \left(\beta - \alpha\right) x_j/n\right] \ .$$

We have shown that the inner sum is equal to zero unless mode β aliases to mode α on the grid, which means $\beta = \alpha + kn$ for some integer k. We also showed that if mode β does alias to mode α on the grid, then each of the summands is equal to one. Since $\Delta x/p = 1/n$, this implies that

$$\widehat{f}_\alpha^{(n)} = \sum_{k=-\infty}^{\infty} \widehat{f}_{\alpha+kn} \ . \tag{7.33}$$

This shows that the discrete Fourier coefficient is equal to the continuous Fourier coefficient (the $k = 0$ term on the right) plus all the coefficients that alias to it (the terms $k \neq 0$).

You might worry that the continuous function has Fourier coefficients with both positive and negative α while the DFT computes coefficients for $\alpha = 0, 1, \ldots, n-1$. The answer to this is aliasing. We find approximations to the negative α Fourier coefficients using $\widehat{f}_{-\alpha}^{(n)} = \widehat{f}_{n-\alpha}^{(n)}$. It may be more helpful to think of the DFT coefficients as being defined for $\alpha \approx -\frac{n}{2}$ to $\alpha \approx \frac{n}{2}$ (the exact range depending on whether n is even or odd) rather than from $\alpha = 0$ to $\alpha = n-1$. The aliasing formula shows that if f is smooth, then $\widehat{f}_\alpha^{(n)}$ is a very good approximation to $\widehat{f}_\alpha$.

7.2.3 FFT algorithm

It takes n^2 multiplies to carry out all the sums in (7.30) directly (n terms in each of n sums). The *Fast Fourier Transform*, or *FFT*, is an algorithm that

calculates the n components of $\widehat{f}$ from the n components of f using $O(n\log(n))$ operations, which is much less for large n.

The idea behind FFT algorithms is clearest when $n = 2m$. A single DFT of size $n = 2m$ is reduced to two DFT calculations of size $m = n/2$ followed by $O(n)$ work. If $m = 2r$, this process can be continued to reduct the two size m DFT operations to four size r DFT operations followed by $2 \cdot O(m) = O(n)$ operations. If $n = 2^p$, this process can be continued p times, where we arrive at $2^p = n$ trivial DFT calculations of size one each. The total work for the p levels is $p \cdot O(n) = O(n\log_2(n))$.

There is a variety of related methods to handle cases where n is not a power of 2, and the $O(n\log(n))$ work count holds for all n. The algorithm is simpler and faster for $n = 2^p$. For example, an FFT with $n = 2^{20} = 1,048,576$ should be significantly faster than with $n = 1,048,573$, which is a prime number.

Let $W_{n\times n}$ be the complex $n \times n$ matrix[5] whose (α, j) entry is $w_{\alpha,j} = w^{-\alpha j}$, where w is a primitive n^{th} root of unity. The DFT (7.30), but for the factor of $\frac{1}{n}$, is the matrix product $\widetilde{f} = W_{n\times n} f$. If $n = 2m$, then w^2 is a primitive m^{th} root of unity and an $m \times m$ DFT involves the matrix product $\widetilde{g} = W_{m\times m} g$, where the (α, k) entry of $W_{m\times m}$ is $(w^2)^{-\alpha k} = w^{-2\alpha k}$. The reduction splits $f \in C^n$ into $g \in C^m$ and $h \in C^m$, then computes $\widetilde{g} = W_{m\times m} g$ and $\widetilde{h} = W_{m\times m} h$, then combines $\widetilde{g}$ and $\widetilde{h}$ to form $\widetilde{f}$.

The elements of $\widetilde{f}$ are given by the sums

$$\widetilde{f}_\alpha = \sum_{j=0}^{n-1} w^{-\alpha j} f_j \ .$$

We split these into even and odd parts, with $j = 2k$ and $j = 2k+1$ respectively. Both of these have k ranging from 0 to $\frac{n}{2} - 1 = m - 1$. For these sums, $-\alpha j = -\alpha(2k) = -2\alpha k$ (even), and $-\alpha j = -\alpha(2k+1) = -2\alpha k - \alpha$ (odd) respectively. Thus

$$\widetilde{f}_\alpha = \sum_{k=0}^{m-1} w^{-2\alpha k} f_{2k} + w^{-\alpha} \sum_{k=0}^{m-1} w^{-2\alpha k} f_{2k+1} \ . \tag{7.34}$$

Now define $g \in C^m$ and $h \in C^m$ to have the even and odd components of f respectively:

$$g_k = f_{2k} \ , \quad h_k = f_{2k+1} \ .$$

The $m \times m$ operations $\widetilde{g} = W_{m\times m} g$ and $\widetilde{h} = W_{m\times m} h$, written out, are

$$\widetilde{g}_\alpha = \sum_{k=0}^{n-1} \left(w^2\right)^{-\alpha k} g_k \ ,$$

and

$$\widetilde{h}_\alpha = \sum_{k=0}^{n-1} \left(w^2\right)^{-\alpha k} h_k \ .$$

[5]This definition of W differs from that of Section 7.2 by a factor of $\sqrt{n}$.

Then (7.34) may be written

$$\widetilde{f}_\alpha = \widetilde{g}_\alpha + w^{-\alpha}\widetilde{h}_\alpha \ . \tag{7.35}$$

This is the last step, which reassembles $\widetilde{f}$ from $\widetilde{g}$ and $\widetilde{h}$. We must apply (7.35) for n values of α ranging from $\alpha = 0$ to $\alpha = n-1$. The computed $\widetilde{g}$ and $\widetilde{h}$ have period m ($\widetilde{g}_{\alpha+m} = \widetilde{g}_\alpha$, etc.), but the factor $w^{-\alpha}$ in front of $\widetilde{h}$ makes $\widetilde{f}$ have period $n = 2m$ instead.

To summarize, an order n FFT requires first n copying to form g and h, then two order $n/2$ FFT operations, then order n copying, adding, and multiplying. Of course, the order $n/2$ FFT operations themselves boil down to copying and simple arithmetic. As explained in Section 5.7, the copying and memory accessing can take more computer time than the arithmetic. High performance FFT software needs to be chip specific to take full advantage of cache.

7.2.4 Trigonometric interpolation

Interpolating $f(x)$ at points x_j, $j = 1, \ldots, n$, means finding another function $F(x)$ so that $F(x_j) = f(x_j)$ for $j = 1, \ldots, n$. In polynomial interpolation, $F(x)$ is a polynomial of degree $n-1$. In *trigonometric interpolation*, $F(x)$ is a *trigonometric polynomial* with n terms. Because we are interested in values of $F(x)$ off the grid, do should not take advantage of aliasing to simplify notation. Instead, we let Z_n be a set of integers as symmetric about $\alpha = 0$ as possible. This depends on whether n is even or odd

$$Z_n = \begin{cases} \{-m, -m+1, \ldots, m\} & \text{if } n = 2m+1 \text{ (i.e. } n \text{ is odd)} \\ \{-m+1, \ldots, m\} & \text{if } n = 2m \text{ (i.e. } n \text{ is even)} \end{cases} \tag{7.36}$$

The high accuracy of DFT based methods comes from the high precision of approximation. If $f(x)$ is a periodic function of x, $f(x+R) = f(x)$ for all x, and if f is analytic, as most smooth functions are in practice, then the interpolating *trigonometric polynomials* (details below):

$$\widetilde{f}(x) = \sum_{\alpha \leq N} \widetilde{c}_\alpha e^{2\pi i \alpha x/N} \ , \tag{7.37}$$

satisfies

$$\max_x \left| \widetilde{f}(x) - f(x) \right| \leq Ce^{-\mu N} \ , \tag{7.38}$$

of course with positive C and μ.

To appreciate this, consider approximating $f(x)$ using N uniformly spaced mesh points, $x_k = k\Delta x$, $\Delta x = R/N$. An approximation of order p, such as local polynomial or spline interpolation, has

$$\max_x \left| \widetilde{f}(x) - f(x) \right| = O\left(\Delta x^p\right) = O\left(N^{-p}\right) \ . \tag{7.39}$$

Although $N^{-p} \to 0$ as $N \to \infty$, no matter how large p is, the exponential in (7.38) is faster in the sense that $e^{-\mu N}/N^{-p} \to 0$ as $N \to \infty$. This is the sense

in which trigonometric interpolation is higher order than any finite order of accuracy.

At its core the DFT is an $N \times N$ unitary matrix that represents a change of basis. It would take N^2 multiplications and additions to calculate the DFT of a vector by direct matrix vector multiplication. The *fast Fourier transform*, or *FFT*, algorithm calculates the DFT in $2N \log_2(N)$ multiplications. For $N = 1024$, this is a reduction by a factor of 50.

7.3 Software

Performance tools.

7.4 References and Resources

The classical books on numerical analysis (Dahlquist and Björck, Isaacson and Keller, etc.) discuss the various facts and forms of polynomial interpolation. There are many good books on Fourier analysis. One of my favorites is by Harry Dym and my colleague Henry McKean[6] I learned the aliasing formula (7.33) from a paper by Heinz Kreiss and Joseph Oliger.

7.5 Exercises

1. Verify that both sides of (7.9) are equal to $2a$ when $f(x) = ax^2 + bx + c$.

2. One way to estimate the derivative or integral from function values is to differentiate or integrate the interpolating polynomial.

3. Show that the real Fourier modes (7.25) form an orthogonal family. This means that

   (a) $\langle u_\alpha, u_\beta \rangle = 0$ if $\alpha \neq \beta$.

   (b) $\langle v_\alpha, v_\beta \rangle = 0$ if $\alpha \neq \beta$.

   (c) $\langle u_\alpha, v_\beta \rangle = 0$ for any α and β.

   In (a), α or β may be zero. In (b), α and β start with one. In (c), α starts with zero and β starts with one. It is easy to verify these relations using complex exponentials and the formula $e^{i\theta} = \cos(\theta) + i \sin(\theta)$. For example, we can write $\cos(\theta) = \frac{1}{2}\left(e^{i\theta} + e^{-i\theta}\right)$, so that $u_\alpha = \frac{1}{2}\left(w_\alpha + w_{-\alpha}\right)$. Therefore

$$\langle u_\alpha, u_\beta \rangle = \frac{1}{4}\Big(\langle w_\alpha, w_\beta \rangle + \langle w_{-\alpha}, w_\beta \rangle + \langle w_\alpha, w_{-\beta} \rangle + \langle w_{-\alpha}, w_{-\beta} \rangle \Big) \; .$$

[6] He pronounces McKean as one would Senator McCain.

You can check that if $\alpha \geq 0$ and $\beta \geq 0$ and $\alpha \neq \beta$, then all four terms on the right side are zero because the w_α are an orthogonal family. Also check this way that

$$\|u_\alpha\|_{L^2}^2 = \langle u_\alpha, u_\alpha \rangle = \|v_\alpha\|_{L^2}^2 = \frac{p}{2} \,, \tag{7.40}$$

if $\alpha \geq 1$, and $\|u_0\|_{L^2}^2 = p$.

4. We wish to use Fourier series to solve the boundary value problem from 4.11.

   (a) Show that the solution to (4.44) and (4.45) can be written as a Fourier *sine series*

   $$u(x) = \sum_{\alpha=1}^{n-1} c_\alpha \sin(\pi\alpha x) \,. \tag{7.41}$$

   One way to do this is to define new functions also called u and f, that *extend* the given ones in a special way to satisfy the boundary conditions. For $1 \leq x \leq 2$, we define $f(x) = -f(x-1)$. This defines f in the interval $[0,2]$ in a way that makes it antisymmetric about $x = 1$. Next if $x \notin [0,2]$ there is a k so that $x - 2k \in [0,2]$. Define $f(x) = f(x-2k)$ in that case. This makes f a periodic function with period $p = 2$ that is antisymmetric about any of the integer points $x = 0$, $x = 1$, etc. Draw a picture to make this two step extension clear. If we express this extended f as a real cosine and sine series, the coefficients of all the cosine terms are zero (why?), so $f(x) = \sum_{\alpha>0} b_\alpha \sin(\pi\alpha x)$. (Why π instead of 2π?) Now determine the c_α in terms of the b_α so that u satisfies (4.44). Use the sine series for u to show that $u = 0$ for $x = 0$ and $x = 1$.

   (b) Write a program in Matlab that uses the Matlab `fft` function to calculate the discrete sine coefficients b_α for a sampled function $f^{(n)}$. The simplest way to program this is to extend f to the interval $[0,2]$ as described, sample this extended f at $2n+1$ uniformly spaced points in the interval $[0,2]$, compute the complex DFT of these samples, then extract the sine coefficients as described in Section 7.2.1.

   (c) Write a Matlab program that takes the b_α and computes $\widetilde{f}(x) = \sum_{\alpha=1}^{n-1} b_\alpha \sin(\pi\alpha x)$ for an arbitrary x value. On one graph, make a plot of $f(x)$ and $\widetilde{f}(x)$ for $x \in [0,1]$. On another plot the error $f(x) - \widetilde{f}(x)$. Check that the error is zero at the sampling points, as it should be. For plotting, you will have to evaluate $f(x)$ and $\widetilde{f}(x)$ at many more than the sampling points. Do this for $n = 5, 20, 100$ and $f(x) = x^2$ and $f(x) = \sin(3\sin(x))$. Comment on the accuracy in the two cases.

   (d) Use the relation between b_α and c_α to calculate the solution to (4.44) and (4.45) for the two functions f in part (c). Comment on the difference in accuracy.

(e) Show that the eigenvectors of the matrix A in Exercise 4.11 are discrete sine modes. Use this to describe an algorithm to express any vector, F, in terms as a linear combination of eigenvectors of A. This is more or less what part (a) does, only stated in a different way.

(f) Use part (e) to develop a *fast algorithm* (i.e. as fast as the FFT rather than the direct method) to solve $Au = F$. Write a Matlab code to do this. Compare the accuracy of the second order method from Exercise 4.11 to the DFT based algorithm of part (c).