Chapter 2

The Finite Element Method

In this chapter we wish to discuss methods for approximating solutions to a two-point boundary value problem. We shall give a quick overview of the finite difference method, which is developed from the strong form of the TPBVP, and then move on to the finite element method, which is derived from the weak form of the TPBVP. We will reserve discussion of the use of the finite element method in the minimization form of the problem until we deal with nonlinear equations later in the course.

2.1 The Finite Difference Method

We begin by giving a brief description of the finite difference method with no accompanying mathematical analysis. Although this course focuses on the finite element method, the finite difference method was in widespread use before the finite element method gained acceptance, and it is still a good method for use in some applications. Its use and mathematical analysis become problematic, however, when it is used on problems with complicated geometry or boundary conditions–problems which arise particularly often in engineering applications and for which the finite element method is particularly well-suited.

As we stated above, the finite difference method is developed from the strong form of the TPBVP. We begin with the definition of a derivative

$$u'(x) = \lim_{h \to 0} \frac{u(x+h) - u(x)}{h}.$$

Actually, it works better if we begin with a centered difference instead of a right-hand difference:

$$u'(x) = \lim_{h \to 0} \frac{u(x+h/2) - u(x-h/2)}{h}.$$

Instead of taking a limit, however, we approximate the derivative of u at x by some finite difference:

$$u'(x) \approx \frac{u(x+h/2) - u(x-h/2)}{h}.$$

We next introduce a shorthand for this difference quotient. Let

$$D_h u(x) = \frac{u(x+h/2) - u(x-h/2)}{h}.$$

In order to set up the finite difference method for a two-point boundary value problem, we first introduce a regular grid of points. Let $n > 0$ be an integer, and let $x_i = \frac{i}{n}$, $i = 0, ..., n$. Thus $x_0 = 0$ and $x_n = 1$. Let also $h = \frac{1}{n}$, so that $x_{i+1} - x_i = h$ for all i. We then discretize the problem

$$\begin{aligned} -(au')' &= f \text{ in } (0,1), \\ u(0) &= u(1) = 0 \end{aligned} \tag{2.1}$$

by replacing derivatives with difference quotients:

$$\begin{aligned} -D_h(aD_hu)(x_i) &\approx f(x_i) \text{ for all } i = 1, ..., n-1, \\ u(0) &= u(1) = 0. \end{aligned} \tag{2.2}$$

In order to complete the discretization of (2.1), we replace the quantities $u(x_i)$ with unknowns U_i^h, $i = 0, ..., n-1$, that is, we seek to find U_i^h such that $u(x_i) \approx U_i^h$. Obviously, we should let $U_0^h = U_n^h = 0$. We next rewrite the first portion of (2.2) as

$$\begin{aligned} -D_h(a(x_i)\tfrac{u(x_i+h/2)-u(x_i-h/2)}{h}) &= -\tfrac{a(x_i+h/2)(u(x_i+h)-u(x_i))-a(x_i-h/2)(u(x_i)-u(x_i-h))}{h^2} \\ &= -\tfrac{a(x_i+h/2)(u(x_{i+1})-u(x_i))-a(x_i-h/2)(u(x_i)-u(x_{i-1}))}{h^2} \approx f(x_i). \end{aligned} \tag{2.3}$$

We then substitute U_i^h for $u(x_i)$ in (2.3) and set the resulting expression equal to $f(x_i)$ to obtain

$$-\frac{a(x_i + \frac{h}{2})(U_{i+1}^h - U_i^h) - a(x_i - \frac{h}{2})(U_i^h - U_{i-1}^h)}{h^2} = f(x_i), \quad i = 1, ..., n-1.$$

Thus the finite difference method results in $n-1$ equations in $n-1$ unknowns U_i^h, $i = 1, ..., n$ (recall that we have fixed U_0^h and U_n^h already). We thus may rewrite the problem in vector-matrix form. Let $U = [U_1^h; U_2^h; ...; U_{n-1}^h]$ be the column vector of unknowns, and let $F = [f(x_1); f(x_2); ...; f(x_{n-1})]$. Also, let $a_i = a(x_i + \frac{h}{2})$, and let A be the matrix

$$\frac{1}{h^2}\begin{bmatrix} (a_0+a_1) & -a_1 & 0 & 0 & 0 \\ -a_1 & (a_1+a_2) & -a_2 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & -a_{n-3} & (a_{n-3}+a_{n-2}) & -a_{n-2} \\ 0 & 0 & 0 & -a_{n-2} & (a_{n-2}+a_{n-1}) \end{bmatrix}$$

Then we may find the unknown vector U by solving the linear system $AU = F$. The unknowns U_i then serve as an approximation to $u(x_i)$, $i = 1, ..., n-1$.

2.2 Galerkin's Method

2.2.1 Fourier expansions

The finite element method is a special type of what is known as a Galerkin method, so called because it was invented by the Russian engineer Galerkin about 1915. The basic idea of Galerkin's method is to replace an infinite-dimensional vector space of functions ($\mathcal{A}$ in our case) with a finite-dimensional vector space of functions which is contained in the original infinite-dimensional space.

One may think of solving the weak form of a TPBVP as something like solving an infinite set of linear equations (a fact we demonstrate below), and Galerkin approximations may be computed by solving a finite set of linear equations. Thus Galerkin's method may somewhat imprecisely be thought of as approximating the solution of an infinite set of linear equations with the solution of a finite set of linear equations.

We use Fourier analysis to demonstrate how a TPBVP may be thought of as an infinite set of linear equations. Let $u \in \mathcal{A}$. Then u has a Fourier expansion $u(x) = u_0 + \sum_{i=1}^{\infty}(a_i \sin \pi i x + b_i \cos \pi i x)$. We consider the weak Neumann problem: Find $u \in \mathcal{A}$ such that

$$\mathcal{L}(u, v) = (f, v), \quad v \in \mathcal{A}. \tag{2.4}$$

Any $v \in \mathcal{A}$ has a Fourier expansion of the form $v(x) = v_0 + \sum_{i=1}^{\infty}(c_i \sin \pi i x + d_i \cos \pi i x)$. For the sake of convenience, we let $\psi_0 = 1$, $\psi_{2i-1}(x) = \sin \pi i x$, $\psi_{2i}(x) = \cos \pi i x$, $v_{2i-1} = c_i$, and $v_{2i} = d_i$. Then we may rewrite the Fourier expansion of v as $v(x) = \sum_{i=0}^{\infty} v_i \psi_i(x)$. Roughly speaking, then, the set $\{\psi_i\}_{i=0,1,2...}$ is a basis for $\mathcal{A}$. Thus if u satisfies

$$\mathcal{L}(u, \psi_j) = (f, \psi_j), \quad j \geq 0, \tag{2.5}$$

we find that

$$\mathcal{L}(u, v) = \mathcal{L}(u, \sum_{j=0}^{\infty} v_j \psi_j) = \sum_{j=0}^{\infty} v_j \mathcal{L}(u, \psi_j) = \sum_{j=0}^{\infty} v_j (f, \psi_j) = (f, \sum_{j=0}^{\infty} v_j \psi_j) = (f, v).$$

Thus (2.5) is (at least formally) equivalent with the weak form (2.4). Writing $u = \sum_{i=0}^{\infty} u_i \psi_i$, we rewrite (2.5) as

$$\mathcal{L}(u, v) = \sum_{i=0}^{\infty} u_i \mathcal{L}(\psi_i, \psi_j) = (f, \psi_j), \quad j \geq 0. \tag{2.6}$$

(2.6) is an infinite set of linear equations in an infinite number of unknowns u_i, $i \geq 0$. We note that if we can find the coefficients u_i, $i \geq 0$, we have found u (or more precisely, we have found the Fourier coefficients of u) and have thus solved the weak form of our equation. Thus the weak solution of a two-point boundary value problem may essentially be found by solving an infinite set of linear equations in an infinite number of unknowns.

2.2.2 A more general TPBVP

We shall now consider a TPBVP with lower-order terms, or more precisely, we consider the problem

$$-(a(x)u'(x))' + b(x)u'(x) + c(x) = f(x) \text{ in } (0, 1)$$

with Neumann or Dirichlet (or mixed) boundary conditions. We now write

$$\mathcal{L}(u, v) = \int_0^1 au'v' \, dx + \int_0^1 bu' \, dx + \int_0^1 cuv \, dx$$

and

$$\mathcal{L}_{NEU}(u, v) = \mathcal{L}(u, v) - \nu_0 v(0) + \nu_1 v(1), \tag{2.7}$$

where $\nu_0 = a(0)u'(0)$ and $\nu_1 = a(1)u'(1)$. Then a weak Dirichlet problem may be written as: Find $u \in \mathcal{A}_{Dir}$ such that

$$\mathcal{L}(u, v) = (f, v) \text{ for all } v \in \mathcal{A}_0,$$

while a weak Neumann problem may be written as: Find $u \in \mathcal{A}$ such that

$$\mathcal{L}_{NEU}(u,v) = (f,v) \text{ for all } v \in \mathcal{A}.$$

As long as $b = 0$, we may, as in Section 1.4, consider $\mathcal{L}$ to be an inner product on A (or $\mathcal{A}_0$). If b is nonzero, $\mathcal{L}(u,v) \neq \mathcal{L}(v,u)$ in general, so L is no longer an inner product. It still has most of the same properties, however, and we shall basically treat it as an inner product. Similarly, with some restrictions on the coefficients a, b, and c, $\|u\|_{eng} = \sqrt{\mathcal{L}(u,u)}$ is a norm on $\mathcal{A}_0$ (or $\mathcal{A}$) as before.

2.2.3 Galerkin's method

Galerkin's method involves approximating the infinite-dimensional space of admissible functions in a weak problem ($\mathcal{A}$, $\mathcal{A}_0$, or $\mathcal{A}_{Dir}$ in our case) with an N-dimensional space S_N, then rewriting the weak problem with the finite-dimensional space taking the place of the infinite-dimensional space. This leads to a finite-dimensional set of linear equations which may then be solved. We consider a Neumann problem to start with. Let $S_N \subset \mathcal{A}$ be a finite-dimensional vector space having dimension N. Then Galerkin's method is to find $u_N \in S_N$ such that

$$\mathcal{L}(u_N, v_N) = (f, v_N) \text{ for all } v_N \in S_N. \tag{2.8}$$

(Note that this is a Neumann problem with boundary conditions $u'(0) = u'(1) = 0$.)

We then let $\{\phi_i\}$, $i = 1, ..., N$, be a basis for S_N. We assert that $u_N \in S_N$ satisfies (2.8) if and only if $u_N \in S_N$ satisfies

$$\mathcal{L}(u_N, \phi_j) = (f, \phi_j), \quad j = 1, ..., N. \tag{2.9}$$

If u_N satisfies (2.8), it solves (2.9) as $\phi_j \in S_N$, $i = 1, ..., N$. We wish to show that if u_N satisfies (2.9), then it also satisfies (2.8). Let $v_N \in S_N$. Since $\{\phi_i\}_{i=1,...,N}$ is a basis for S_N, $v_N = \sum_{j=1}^{N} v_j \phi_j$. Then using this representation for v_N and (2.9), we find

$$\mathcal{L}(u_N, v_N) = \mathcal{L}(u_N, \sum_{j=1^N} v_j \phi_j) = \sum_{j=1}^{N} v_j \mathcal{L}(u_N, \phi_j) = \sum_{j=1}^{N} v_j (f, \phi_j) = (f, \sum_{j=1}^{N} v_j \phi_j) = (f, v_N).$$

We next show that finding u_N is equivalent to solving an $N \times N$ set of linear equations. Clearly (2.9) represents a set of N linear equations. Also, we may represent our unknown function u_N as $u_N = \sum_{i=1}^{N} u_i \phi_i$, so that (2.9) may be rewritten as: Find a set of coefficients u_i, $i = 1, ..., N$, such that

$$\sum_{i=1}^{N} u_i \mathcal{L}(\phi_i, \phi_j) = (f, \phi_j), \quad j = 1, ..., N. \tag{2.10}$$

This is a set of N equations in the N unknowns $\{u_i\}_{i=1,...,N}$. We also may rewrite (2.10) in matrix-vector notation, as we did in the case of the finite difference method. We let S be the matrix having entries $S_{ij} = \mathcal{L}(\phi_j, \phi_i)$. S is called the *stiffness matrix*. We also define the column vectors $U = [u_1; u_2; ...; u_N]$ and $F = [(f, \phi_1); (f, \phi_2); ...; f(\phi_N)]$. Then we seek to solve the matrix equation

$$SU = F$$

for the vector unknown U. We note that nothing we have said indicates that this equation has a solution. (We do note that since it is a square system, any solutions that exist will be unique.) In fact, a solution may or may not exist, that is, the matrix S may or may not be singular.

We give two examples here of possible finite-dimensional spaces S_N. We save our most important example, piecewise polynomial spaces, for our discussion of the finite element method.

Example 1: Trigonometric Polynomials. We previously considered Fourier series expansions. Instead of considering full Fourier expansions, here we construct a finite-dimensional space of dimension $2N+1$ by taking $S_{2N+1} = span\{1, \sin \pi x, \sin 2\pi x, ..., \sin N\pi x, \cos \pi x, \cos 2\pi x, ..., \cos N\pi x\}$. Thus members of S_{2N+1} will have the form $v_{2N+1} = v_0 + \sum_{i=1}^{N} a_i \sin \pi i x + b_i \cos \pi i x$, that is, they appear as truncated Fourier series. S_{2N+1} is said to be a space of *trigonometric polynomials.*

Example 2: Polynomials. We may simply take $S_{N+1} = span\{1, x, ..., x^N\}$, so that members of S_{N+1} have the form $V_N = \sum_{i=0}^{N} v_i x^i$.

One could certainly think of other finite-dimensional spaces of functions; these are just two of the more basic examples, with their most obvious bases stated.

2.2.4 Error analysis of Galerkin's method

We next consider a basic error estimate for Galerkin's method, that is, we wish to say something about the size of the error $u - u_N$. Both the assumptions we make in the statement of our estimate and the proof will be rather imprecise as our goal is to give a flavor of the main ideas involved, not to give a completely rigorous proof.

Before stating an error estimate, we first comment further on the bilinear form $\mathcal{L}$. It sometimes happens that $\mathcal{L}(u,u) = 0$ for some u in the admissible class appropriate for a given problem even though $u \neq 0$. Consider, for example, the case where $\mathcal{L}(u,u) = \int_0^1 (u')^2 \, dx$, with Neumann boundary conditions so that the appropriate admissible class is all of $\mathcal{A}$. Then for $u_0 \equiv 1$, $\mathcal{L}(u,u) = 0$ but $u \neq 0$. In this case, $\mathcal{L}$ is not an inner product according to the definition given in exercise 2 of chapter 1 (property (e) is not satisfied). In terms of solving differential equations, this means that if $\mathcal{L}(u,v) = (f,v)$ for all $v \in \mathcal{A}$, it is also true that $\mathcal{L}(u+u_0,v) = \mathcal{L}(u,v) + \mathcal{L}(u_0,v) = (f,v)$ for all $v \in \mathcal{A}$. That is, the differential equation $\mathcal{L}(u,v) = (f,v)$ for all $v \in \mathcal{A}$ does not have a unique solution. This means that we can't expect Galerkin's method to work well (which solution will be approximated?), so we should try to rule this situation out. We shall do so by requiring that $\mathcal{L}(\cdot,\cdot)$ be an inner product.

We also make a few comments about projections. We consider projections from a vector space V having an inner product $(\cdot,\cdot)$ onto a subspace $X \subset V$. Then the projection of a point $x \in V$ onto X is taken to be the unique point Px satisfying $(x - Px, y) = 0$ for all $y \in X$. That is, Px is the point in X such that $x - Px$ is orthogonal (or perpendicular) to X. We also note that projections are distance-minimizing in the norm $\|x\| = \sqrt{(x,x)}$:

$$\|x - Px\| = \min_{y \in X} \|x - y\|,$$

or stated slightly differently,

$$\|x - Px\| \leq \|x - y\| \text{ for all } y \in X. \tag{2.11}$$

We leave the proof of this fact as an exercise.

We note that if

$$\mathcal{L}(u,v) = (f,v)$$

for all $v \in \mathcal{A}$ and

$$\mathcal{L}(u_N, v_N) = (f, v_N)$$

for all $v_N \in S_N \subset \mathcal{A}$, then we may subtract the above two equations to find that $\mathcal{L}(u - u_N, v_N) = 0$ for all $v_N \in S_N$. Thus if $\mathcal{L}(\cdot, \cdot)$ is an inner product, u_N is the projection of u onto S_N with respect to the inner product $\mathcal{L}(\cdot, \cdot)$. (Even if $\mathcal{L}(\cdot, \cdot)$ is not an inner product, however, we still call u_N the *Galerkin projection* of u.) These observations lead us to the following theorem.

Theorem 2.2.1 *Assume that $\mathcal{L}(\cdot, \cdot)$ is an inner product on some admissible class $\bar{\mathcal{A}}$ of functions (one may think of $\bar{\mathcal{A}} = \mathcal{A}$ or $\bar{\mathcal{A}} = \mathcal{A}_0$), and let $S_N \subset \bar{\mathcal{A}}$ be a finite-dimensional subspace of $\bar{\mathcal{A}}$. Also, assume that u_N is the Galerkin projection of some $u \in \bar{\mathcal{A}}$ onto S_N. Then*

$$\|u - u_N\|_{eng} = \min_{\chi \in S_N} \|u - \chi\|_{eng}. \tag{2.12}$$

Proof. Since $\mathcal{L}(\cdot, \cdot)$ is an inner product, u_N is a projection of u onto S_N. Since projections are distance-minimizing in the corresponding norm, (2.12) must hold.

Remark. We note that we have not covered here the case of nonsymmetric forms, i.e., of forms where $\mathcal{L}(u, v) \neq \mathcal{L}(v, u)$. A similar theorem holds in this case, but we do not state it. We also note that we have been rather lax in our treatment of projections. In particular, we have left out some technicalities necessary to deal with the fact that $\bar{A}$ is an infinite-dimensional space. Our goal here was to emphasize that Galerkin's method involves finding a projection of a function contained in an infinite dimensional space onto a finite dimensional space and to explore the consequences of this fact while not overburdening the reader with technical details.
Remark. In (2.12), we have used the energy norm to measure the error $u - u_N$. Notice that the energy norm really measures the *average* (or the average of the square) of the error function $u - u_N$ over the interval $(0, 1)$. This leaves open the possibility that $u - u_N$ will be very large over a small subinterval of $(0, 1)$ while being very small elsewhere (if you want 100 numbers to have an average of 1, you can either choose them all to be 1, or you can choose 1 to be 99.01 and the rest to be .01; the same applies to functions). Rather than knowing that the average (energy) error is small, it would be better to know that the error is small everywhere. However, it is theoretically much more convenient to measure the error in the energy norm, and we restrict ourselves to doing so.

2.3 The Finite Element Method

(2.12) tells us that the Galerkin approximation u_N of u from a space S_N is the best approximation as measured in the energy norm. We now discuss the task of choosing a space S_N which is able to do a good job of approximating functions in $\mathcal{A}$ (or $\mathcal{A}_0$) and which also is relatively easy to compute with (our final aim, after all, is the construction of a computer code to implement Galerkin's method).

2.3.1 Piecewise Linear Spaces

The spaces S_N which we shall employ are alternatively known as piecewise polynomial or finite element spaces. In the present 1-dimensional case, they also are called spline spaces. We begin with the simplest example (and one which is widely used in practice), that of continuous piecewise linear functions, a "typical" example of which is displayed in Figure 2.1.

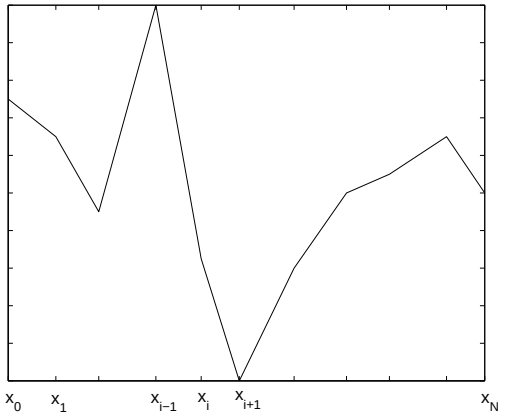

Figure 2.1: A "typical" piecewise linear function

We may describe the space of piecewise linear functions as follows. Let $0 = x_0 < x_1 < x_2 < ... < x_{N-1} < x_N = 1$. We shall call the x_i's *mesh points*. Also, let $h_i = x_i - x_{i-1}$, $i = 1, ..., N$ (so that h_i is the length of the subinterval $[x_{i-1}, x_i]$). We shall also denote by $h(x)$ the function which is h_i when $x \in (x_{i-1}, x_i)$ (it isn't defined at the points x_i, but this doesn't matter). Then we define the space S_h to be the functions which are continuous and which are linear on each subinterval $[x_i, x_{i+1}]$ (note that instead of indexing the space S by its dimension N, we are now indexing it by the mesh size function h).

We next note that if we fix the values of a function $v_h \in S_h$ at each of the $N+1$ mesh points x_i, $i = 0, ..., N$, then we have determined the function v_h. Thus we conjecture that S_h is an $N+1$-dimensional space. To confirm this fact, we exhibit a basis for S_h containing $N+1$ members. We let ψ_i, $i = 0, ..., N$ be the "hat" function in S_h which is 1 at x_i and 0 at x_j if $j \neq i$ (such a basis function is pictured in Figure 2.2). Then if $v_h \in S_h$, it is easy to verify that $v_h = \sum_{i=0}^{N} v_h(x_i)\psi_i$. Also, the ψ_i's are clearly linearly independent, so they form a basis with $N+1$ members.

We next comment on the smoothness properties of the space S_h. We have hitherto always said that our class $\mathcal{A}$ is composed of "smooth enough" functions, and it is clear that members of S_h are not very smooth. In fact, they have piecewise constant derivatives which are undefined at mesh points (since members of S_h have corners at the mesh points). In Figure 2.3 we picture the derivative of a basis function ψ_i. Why, then, do we consider S_h to be a subspace of $\mathcal{A}$? We do not give a full answer here, but we do note that members of S_h are just smooth enough to use in the weak form of the equation. While v_h' is not defined everyplace for $v_h \in S_h$, v_h' is defined at all but a few points, and if we take ϕ to be a smooth function, the integration by parts formula

$$\int_0^1 v_h \phi' \, dx = v_h \phi|_0^1 - \int_0^1 v_h' \phi \, dx \tag{2.13}$$

holds. Secondly, we note that the energy norm that we have used in (2.12) to measure the error in the Galerkin projection only involves first derivatives–which we again note are well-defined for members of S_h. Thus we use functions which have only one derivative, and then not even at every

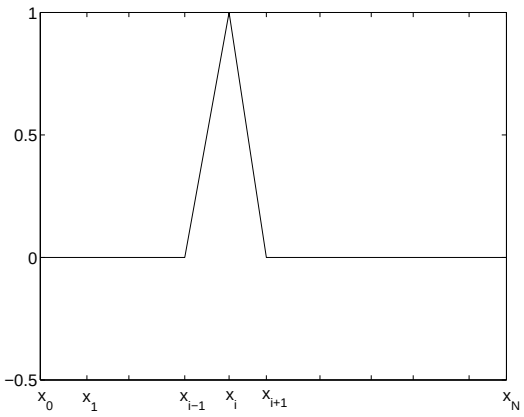

Figure 2.2: The piecewise linear basis function ψ_i

point, to approximate solutions of differential equations which in their strong form are required to have two derivatives. This may seem like a slightly paradoxical situation, but as we shall see, it nonetheless works quite well.

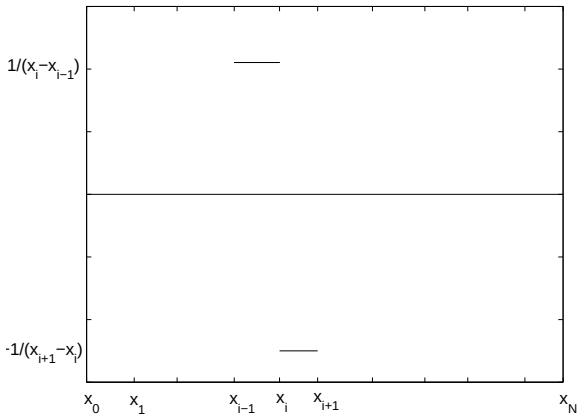

Figure 2.3: The derivative ψ_i' of a basis function

We next discuss some of the mechanics of implementing the finite element method using the piecewise linear space S_h. We recall from (2.10) that the stiffness matrix S for a Galerkin method on a space with basis ψ_i is defined by $S_{ji} = \mathcal{L}(\psi_i, \psi_j)$, where $\mathcal{L}(u,v) = \int_0^1 au'v'\,dx + \int_0^1 bu'\,dx + \int_0^1 cuv\,dx$. We also introduce some terminology: we define the support of a function v to be the closure of the set of values for which v is nonzero, or $supp(v) = \overline{\{x : v(x) \neq 0\}}$. Next we note that since $supp(\psi_i) \cap supp(\psi_j) = \emptyset$ if $|j-i| > 1$, we have $S_{ji} = \mathcal{L}(\psi_i, \psi_j) = 0$ if $|j-i| > 1$. Thus S is a

tridiagonal matrix having form

$$\begin{bmatrix} \ddots & \ddots & & 0 \\ \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots \\ 0 & & \ddots & \ddots \end{bmatrix} \tag{2.14}$$

just as in the finite difference method. More generally, we say that S is *banded* with bandwidth 1 because all of its nonzero entries lie within 1 diagonal row of the main diagonal. We shall be able to exploit the fact that S is a banded matrix when solving the linear system arising from this finite element method. We leave the discussion of the details for later. We do note, however, that efficient implementation of finite element methods requires careful attention to the structure of the matrices which arise. Gaussian elimination is a computationally expensive algorithm, with the number of operations required growing with the cube of the number of unknowns, and it is generally not feasible to naively apply Gaussian elimination. In the present case of 1-dimensional problems we will be able to arrange things so that we need only solve linear systems with tightly banded matrices. Stiffness matrices arising in higher dimensional problems are not generally tightly banded, however, so other approaches are necessary there.

2.3.2 General Spline Spaces

One may modify the setup given above for piecewise linear spaces in two different ways. The first is to raise the order k of polynomials used while maintaining the same degree of smoothness (that is, functions in the finite element space will be continuous only; their derivatives will not be continuous across mesh points). We will denote by S_h^k the space of functions which are polynomials of at most degree k on each mesh interval $[x_i, x_{i+1}]$ and which are continuous on $[0,1]$. Thus the piecewise linear functions are denoted S_h^1, the piecewise quadratic functions S_h^2, and so forth.

More generally, we may require that members of a finite-dimensional space are in $C^\mu([0,1])$ and are polynomials of degree k when restricted to each mesh subinterval. More precisely, we define $S_h^{k,\mu} = \{\chi \in C^\mu([0,1]) \text{ such that } \chi|_{[x_i,x_{i+1}]} \in P^k\}$. Thus the piecewise linear space S_h from above may be denoted $S_h^{1,0}$, and the piecewise quadratics may be denoted by $S_h^{2,0}$. We note that the space $S_h^{1,1}$ is trivial: in order for a piecewise linear function to have a continuous first derivative, it must be linear globally, that is, a line on the whole interval $[0,1]$. It similarly does not work well to require piecewise quadratic functions to have continuous first derivatives because, in the language of the trade, they do not possess enough degrees of freedom to do so while maintaining the ability to approximate well.

The "smallest" (or lowest order) piecewise polynomial space which one may efficiently require to have continuous derivatives is the Hermit cubic spline space $S_h^{3,1}$. The Hermite cubics and other spline spaces are heavily used in practice, and not only in the context of finite element methods. We shall construct a basis for the Hermite cubics. First, however, we determine appropriate degrees of freedom for piecewise cubic polynomials. We note that C^1 functions must have continuous function values and continuous derivatives, so it seems reasonable to specify the function values and first derivatives of $\chi \in S_h^{3,1}$ at each mesh point x_i. The following lemma tells us that doing so uniquely determines a piecewise cubic function.

Proposition 2.3.1 *Assume that $g(x)$ is a cubic polynomial on $[x_0, x_1]$, where $x_1 > x_0$. Then the values $g_0 = g(x_0)$, $g_1 = g'(x_0)$, $g_2 = g(x_1)$, and $g_3 = g'(x_1)$ determine the polynomial g uniquely.*

Proof. We write $g(x) = c_3(x - x_0)^3 + c_2(x - x_0)^2 + c_1(x - x_0) + c_0$. Computing $g(x_0)$, $g'(x_0)$, $g(x_1)$, and $g'(x_1)$, we obtain the following set of four linear equations for c_i, $i = 0, ..., 3$:

$$\begin{aligned} c_0 &= g_0, \\ c_1 &= g_1, \\ c_3(x_1 - x_0)^3 + c_2(x_1 - x_0)^2 + c_1(x - x_0) + c_0 &= g_2, \\ 3c_3(x_1 - x_0)^2 + 2c_2(x_1 - x_0) + c_1 &= g_3. \end{aligned}$$

We recall the fundamental theorem of linear algebra, which states that a square system of linear equations has a unique solution if and only if it has a trivial null space, that is, if and only if the only time each equation is 0 is when all coefficients are 0. Thus we assume $g_i = 0$, $i = 0, ..., 3$. The first two equations above yield $c_0 = c_1 = 0$. The third equation then reduces to $c_3(x_1 - x_0)^3 + c_2(x_1 - x_0)^2 = 0$, and we may solve for c_3 to find that $c_3 = -\frac{c_2}{x_1 - x_0}$. Substituting this expression into the fourth equation above while recalling taht $c_1 = 0$ yields $5c_2(x_1 - x_0) = 0$, which implies that $c2 = 0$. Thus $c_3 = 0$, and we finally find that $g(x) = 0$. Thus our system of equations has a unique solution, and g_i, $i = 0, ..., 3$, uniquely determine g.

We next turn to the problem of determining a basis for $S_h^{3,1}$. We have shown that our degrees of freedom should be the function values and first derivatives at the mesh points because determining these at two mesh points determines a cubic polynomial in between the mesh points. We shall first determine four cubic polynomials on $[0, 1]$, one satisfying $g(0) = 1$, $g'(0) = 0$, $g(1) = 0$, $g'(1) = 0$, one satisfying $g'(0) = 1$ with the other three degrees of freedom being 0, and so on. We will then construct our basis by transforming and scaling these four "building block" functions.

Our proof of the above proposition suggest that we can determine our building block functions by solving four sets of linear equations, and we do so. We let $\tilde{v}_1(\tilde{x}) = c_3\tilde{x}^3 + c_2\tilde{x}^2 + c_1\tilde{x} + c_0$ be the unique cubic polynomial satisfying $\tilde{v}_1(0) = 1$, $\tilde{v}_1'(0) = 0$, $\tilde{v}_1(1) = 1$, and $\tilde{v}_1'(1) = 0$. This leads to a set of four linear equations in four unknowns:

$$\begin{aligned} c_0 &= 1, \\ c_1 &= 0, \\ c_3 + c_2 + c_1 + c_0 &= 0, \\ 3c_3 + 2c_2 + c_1 &= 0. \end{aligned}$$

Solving this set of equations gives us $\tilde{v}_1(\tilde{x}) = 2\tilde{x}^3 - 3\tilde{x}^2 + 1$. Similarly, letting $\tilde{v}_2(\tilde{x})$ satisfy $\tilde{v}_2'(0) = 1$ and $\tilde{v}_2(0) = \tilde{v}_2(1) = \tilde{v}_2'(1) = 0$, we find that $\tilde{v}_2(\tilde{x}) = \tilde{x}^3 - 2\tilde{x}^2 + \tilde{x}$. Letting $\tilde{v}_3(1) = 1$ and letting the other degrees of freedom be 0, we find that $\tilde{v}_3(\tilde{x}) = -2\tilde{x}^3 + 3\tilde{x}^2$. Finally, letting $\tilde{v}_4'(1) = 1$ and letting the other degrees of freedom be 0, we find $\tilde{v}_4(\tilde{x}) = \tilde{x}^3 - \tilde{x}^2$.

Given a mesh $0 = x_0 < x_1 < ... < x_N = 1$, we will have $2(N + 1)$ degrees of freedom, so we need $2(N + 1)$ basis functions. We will construct our basis $\psi_0, ..., \psi_{2N+1}$ so that $\psi_{2i+1}(x_j) = \delta_{ij}$, $\psi_{2i+1}'(x_j) = 0$, $\psi_{2i}(x_j) = 0$, and $\psi_{2i}'(x_j) = \delta_{ij}$, $i, j = 0, ..., N$. We consider in detail only the construction of basis functions corresponding to interior mesh points; basis functions corresponding to the boundary points are similar.

We first construct a basis function $\psi_{2i+1}(x)$ which we shall use to specify the function value at the mesh point x_i. We will construct ψ_{2i+1} so that $supp(\psi_{2i+1}) = [x_{i-1}, x_{i+1}]$. Our strategy is to paste together a transformed version of $\tilde{v}_3$ from x_{i-1} to x_i and a transformed version of $\tilde{v}_1$ from

x_i to x_{i+1}. To transform $\tilde{v}_3(\tilde{x})$ to $[x_{i-1}, x_i]$, we use the change of variables $\tilde{x} = \frac{x-x_{i-1}}{x_i-x_{i-1}} = \frac{x-x_{i-1}}{h_i}$. Similarly, to transform $\tilde{v}_1(\tilde{x})$ to $[x_i, x_{i+1}]$, we use the change of variables $\tilde{x} = \frac{x-x_i}{h_{i+1}}$. Thus we may define

$$\psi_{2i+1}(x) = \begin{cases} \tilde{v}_3(\frac{x-x_{i-1}}{h_i}), & x_{i-1} \le x < x_i, \\ \tilde{v}_1(\frac{x-x_i}{h_{i+1}}), & x_i \le x \le x_{i+1}, \\ 0 \text{ otherwise} \end{cases} .$$

The basis function ψ_{2i+1} is pictured in figure 2.4.

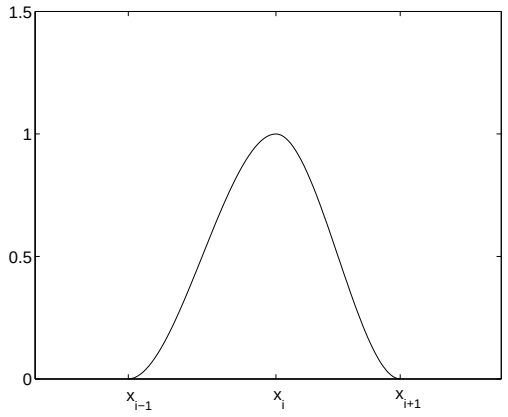

Figure 2.4: The Hermite cubic basis function ψ_{2i+1}

Our first instinct is to define the basis function ψ_{2i} so that $\psi'_{2i}(x_i) = 1$, and in fact this is precisely what we initially did. We shall now backtrack a bit, however, for the following reason. We note that if we define $\psi_{2i}(x) = h_i\tilde{v}_4(\tilde{x})$ for $x_{i-1} \le x \le x_i$, then we will have $\psi'_{2i}(x_i) = h_i\tilde{v}'_4(\frac{x-x_{i-1}}{h_i})\frac{d}{dx}(\frac{x-x_{i-1}}{h_i}) = \tilde{v}'_4(1) = 1$. However, in this case ψ_{2i} would have a maximum of just $h_i max_{\tilde{x}\in[0,1]}\tilde{v}_4$, as compared with a maximum of 1 for ψ_{2i-1}. We wish to have our basis functions be approximately the same size as measured in the L_∞ norm so that the elements of our stiffness matrix won't differ too much in size, so we shall define things a little differently. Our definition is

$$\psi_{2i}(x) = \begin{cases} \frac{2h_i}{h_i+h_{i+1}}\tilde{v}_4(\frac{x-x_{i-1}}{h_i}), & x_{i-1} \le x < x_i, \\ \frac{2h_{i+1}}{h_i+h_{i+1}}\tilde{v}_2(\frac{x-x_i}{h_{i+1}}), & x_i \le x \le x_{i+1}, \\ 0 \text{ otherwise} \end{cases} .$$

ψ_{2i} is plotted in figure 2.5. We note that $\frac{d}{dx}\psi_{2i}(x_i) = \frac{2}{h_i+h_{i+1}}$. For comparison, we note that $\psi_{2i+1}(x_i) = 1$ and $\psi_{2i+1}(x_{i-1}) = 0$, so by the mean value theorem, $\frac{d}{dx}\psi_{2i+1}(\xi) = \frac{1-0}{h_i} = \frac{1}{h_i}$ for some $\xi \in (x_{i-1}, x_i)$. Thus ψ_{2i+1} and ψ_{2i} are about the same size, as are ψ'_{2i+1} and ψ'_{2i}.

We finally note that the stiffness matrix S arising from use of the Hermite cubic polynomials in the finite element method is banded, as was the stiffness matrix arising from use of the piecewise linears. However, the bandwidth in the case of Hermite cubic splines is 3 as opposed to 1 in the case of the piecewise linears.

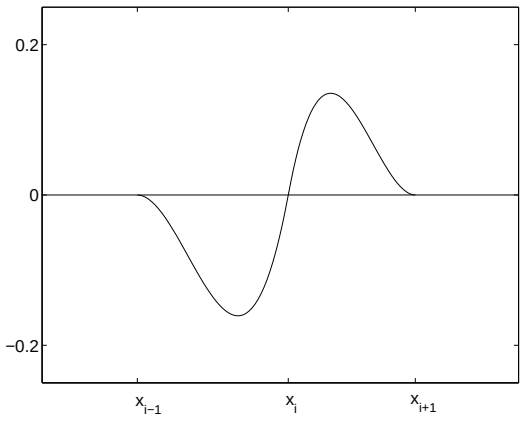

Figure 2.5: The Hermite cubic basis function ψ_{2i}

2.3.3 Boundary conditions in the finite element method

We now discuss in a little more detail how boundary conditions are handled in the finite element method. We begin with Neumann conditions. In Section 2.2.2, we noted that a general Neumann problem with boundary conditions $a(0)u'(0) = \nu_0$, $a(1)u'(1) = \nu_1$ can be written as: Find $u \in \mathcal{A}$ such that

$$\mathcal{L}_{NEU}(u,v) = \mathcal{L}(u,v) - \nu_0 v(0) + \nu_1 v(1) = (f,v) \text{ for all } v \in \mathcal{A}. \tag{2.15}$$

We shall rewrite this equation. For $v \in \mathcal{A}$, we define

$$\mathcal{F}(v) = (f,v),$$

and

$$\mathcal{F}_{Neu}(v) = (f,v) + \nu_0 v(0) - \nu_1 v(1).$$

We may thus rewrite (2.13) as: Find $u \in \mathcal{A}$ such that

$$\mathcal{L}(u,v) = \mathcal{F}_{Neu}(v)$$

for all $v \in \mathcal{A}$. The finite element method for (2.15) may be written as: Find $u_h \in S_h^{k,\mu}$ such that

$$\mathcal{L}(u_h, v_h) = \mathcal{F}_{Neu}(v_h) \text{ for all } v_h \in S_h^{k,\mu}.$$

Let ψ_i, $i = 1, ..., M$ be a basis for $S_h^{k,\mu}$. Then we may rewrite the above finite element method as: Find $u_h \in S_h^{k,\mu}$ such that

$$\mathcal{L}(u_h, \psi_i) = \mathcal{F}_{Neu}(\psi_i) \text{ for all } i = 1, ..., M.$$

We note that for most i, $\psi_i(0) = \psi_i(1) = 0$, in which case $\mathcal{F}_{Neu}(\psi_i) = \mathcal{F}(\psi_i)$. This simplifies the computations since in our code we may first calculate $\mathcal{F}(\psi_i)$ for all i, then add in the boundary

conditions for those basis functions for which it is necessary to do so (and usually there will be only two such basis functions).

We next describe how Dirichlet boundary conditions may be treated in finite element methods. We assume that our Dirichlet conditions are $u(0) = u_0$ and $u_1 = u_1$, so that our problem is: Find $u \in \mathcal{A}_{Dir} = \{v \in \mathcal{A} \text{ s.t. } v(0) = u_0, v(1) = u_1$ such that

$$\mathcal{L}(u, v) = \mathcal{F}(v) \text{ for all } v \in \mathcal{A}_0.$$

We rewrite this problem as follows. Let u_{Dir} be any function in $\mathcal{A}_{Dir}$, and let $u_0 = u - u_{Dir}$. Then our problem may be reformulated as: Find $u_0 \in \mathcal{A}_0$ such that

$$\mathcal{L}(u_0, v) = \mathcal{F}(v) - \mathcal{L}(u_{Dir}, v) \text{ for all } v \in \mathcal{A}_0.$$

We emphasize that the function u_{Dir} is fixed and known; we may choose it to be anything we want (such as, say, the linear function passing throught the points $(0, u_0)$ and $(1, u_1)$). In the finite element method, we shall similarly choose a fixed function $u_{Dir,h} \in S_h^{k,\mu}$ that satisfies $u_{Dir,h}(0) = u_0$ and $u_{Dir,h}(1) = u_1$. We then define $\overset{o}{S}{}_h^{k,\mu} = \{v_h \in S_h^{k,\mu} \text{ s.t. } v_h(0) = v_h(1) = 0\}$. Then the finite element method for the Dirichlet problem is: Find $u_h \in \overset{o}{S}{}_h^{k,\mu}$ such that

$$\mathcal{L}(u_h, v_h) = \mathcal{F}(v_h) - \mathcal{L}(u_{Dir,h}, v_h) \text{ for all } v_h \in \overset{o}{S}{}_h^{k,\mu}.$$

Rewriting this in terms of basis functions ψ_i, $i = 1, ..., M$ for $\overset{o}{S}{}_h^{k,\mu}$, we seek $u_h \in \overset{o}{S}{}_h^{k,\mu}$ such that

$$\mathcal{L}(u_h, \psi_i) = \mathcal{F}(\psi_i) - \mathcal{L}(u_{Dir,h}, \psi_i), \quad i = 1, ..., M.$$

We note that our choice of $u_{Dir,h}$ will have an impact on how convenient and expensive it will be to compute the right hand side of the above equation; in particular, we would like to compute the term $\mathcal{L}(u_{Dir,h}, \psi_i)$ for as few of the basis functions ψ_i as possible. We thus shall choose $u_{Dir,h}$ to be a linear combination of the two basis functions of $S_h^{k,\mu}$ corresponding to the point values at 0 and 1. For example, in the case of piecewise linear elements, we let $u_{Dir,h} = u_0\psi_0 + u_1\psi_N$. In the case of Hermite cubic elements, we let $u_{Dir,h} = u_0\psi_1 + u_1\psi_{2N+1}$. We finally note that in order to produce a basis for $\overset{o}{S}{}_h^{k,\mu}$, we simply delete the basis elements from $S_h^{k,\mu}$ which correspond to point values at 0 and 1. In particular, in the case of piecewise linear elements we let $\overset{o}{S}{}_h^1 = span\{\psi_1, \psi_3, ..., \psi_{N-1}\}$, and in the case of Hermite cubic elements we let $\overset{o}{S}{}_h^{3,1} = span\{\psi_0, \psi_2, \psi_3, ..., \psi_{2N-1}, \psi_{2N}\}$.

2.3.4 An overview of a finite element code

One of the great attractions of the finite element method is that it can be coded in a very modular way. While there are certain pieces of information that many or all parts of the code must have access to, there are several essentially independent functions in a finite element code. The basic structure we shall employ is the following. We begin with a program which calls the various subprograms and contains declarations of global variables, etc. As long as consistency in data structures is maintained, the following parts of the code may be written to be fairly independent of one another: a mesh generator, a subroutine which returns the value of a basis function at a point, a numerical integration routine, a routine to assemble the stiffness matrix and right-hand-side vector F, a solve

for linear systems of equations, and various types of output routines (for example, a routine to calculate various norms of the error in the finite element approximation if we are testing our code on problems with known solutions, or graphing routines).

2.4 Exercises

1. Program the finite difference method in Matlab for the problem

$$\begin{aligned} -(au')' &= f \text{ in } (0,1), \\ u(0) &= u(1) = 0. \end{aligned}$$

Make sure that your code allows you to specify different numbers of mesh points. Use your code to approximate the solution to the above problem with $a(x) = 2 + \sin 50x$ and $f(x) = -[2\pi^2 \sin \pi x - 50\pi \cos 50x \cos \pi x + \pi^2 \sin 50x \sin \pi x]$. Note that this problem has a known solution, $u(x) = \sin \pi x$ (check it out if you're not sure). Next, calculate the error in your solution at each of the mesh points, and calculate $err_N = \max_{i=1,...,N-1} u(x_i) - U_i$. Finally, use a mesh which has mesh intervals which are half the size of those in your first calculation, i.e., one with $2N+1$ mesh points. Then calculate err_{2N+1}, and compare it with err_N. Repeat this procedure a couple more times and try to detect a pattern.
2. Let $(\cdot, \cdot)$ be an inner product on a vector space V with corresponding norm $\|x\| = \sqrt{(x,x)}$. Use Schwarz's inequality

$$(u, v) \leq \|u\|\|v\|$$

to prove that if Px is the projection of a point x onto a subspace $X \subset V$, then

$$\|x - Px\| \leq \|x - y\|$$

for all $y \in X$. (Recall that the projection Px satisfies $(x - Px, y) = 0$ for all $y \in X$.)
3. Construct a basis for use with S_h^2 (that is, the piecewise quadratic polynomials). For degrees of freedom, use the function values at each mesh point x_i and at the midpoint $\frac{x_{i-1}+x_i}{2}$ of each mesh interval. You will thus need to construct two types of basis function. A basis function ψ_{2i+1} of the first type will satisfy $\psi_{2i+1}(x_j) = \delta_{ij}$ for $i, j = 0, ..., N$, and $\psi_{2i+1}(\frac{x_{j-1}+x_j}{2}) = 0$ for $j = 1, ..., N$. A basis function ψ_{2i} of the second type will satisfy $\psi_{2i}(x_j) = 0$ for $i = 1, ..., N$ and for $j = 0, ..., N$ and $\psi_{2i}(\frac{x_{j-1}+x_j}{2}) = \delta_{ij}$ for $i, j = 1, ..., N$.

2.5 Programming Project, Part I

Before giving the specific assignment for this chapter, I'll make a few general comments about the project. First of all, you're reminded that no sharing of code is allowed, although you may discuss issues that arise with classmates. Secondly, you may not use Matlab functions that perform numerical operations except those which would be present in any programming language (such as sin, cos, etc.). You may, however, use Matlab functionality pertaining to data manipulation. For example, using the diag operator to create or manipulate a matrix is fine (even preferred), as is using Matlab's vectorized notation (for example, if one wishes to apply the function f to each element of a vector x, writing $y = f(x)$ is fine–don't use a for-loop in this situation!). Thirdly, debug CAREFULLY! This project will require an error-free code, so try to do it right the first time. Also,

test every portion of your code as well as possible. I will try to give hints about tests you can use to debug as we go, but the final responsibility for constructing a correct code lies with you!

The first part of the project will involve two parts of your final code, the mesh generator and the polynomial basis functions for S_h^1 and $S_h^{3,1}$. Your mesh generation routing should output (or, return as a global variable) a vector containing the mesh points $0 = x_0 < x_1 < x_2 < ... < x_N = 1$. For now, it's fine if the mesh points are evenly spaced, but remember as your programming that you'll eventually want to output meshes which are not uniform. You should also code the polynomial basis functions for the piecewise linear and the Hermite cubic spaces. For each type (linear and cubic), you'll need two functions, one for function values and one for derivatives. In each case, your functions should accept as input a value $x \in [0, 1]$ and an index (in the piecewise linear case, the index of the mesh point at which the basis function is nonzero; you decide how to index the cubic case). The output should then simply be the value of the basis function or of its derivative at x. To test your code, make sure that plots of basis functions and their derivatives are correct for various meshes. Also, check some values by hand.