Chapter 1
Piecewise Polynomial Approximation in 1D

Abstract In this chapter we introduce a type of functions called piecewise polynomials that can be used to approximate other more general functions, and which are easy to implement in computer software. For computing piecewise polynomial approximations we present two techniques, interpolation and L^2-projection. We also prove estimates for the error in these approximations.

1.1 Piecewise Polynomial Spaces

1.1.1 The Space of Linear Polynomials

Let $I = [x_0, x_1]$ be an interval on the real axis and let $P_1(I)$ denote the vector space of linear functions on I, defined by

$$P_1(I) = \{v : v(x) = c_0 + c_1 x,\ x \in I,\ c_0, c_1 \in \mathbb{R}\} \tag{1.1}$$

In other words $P_1(I)$ contains all functions of the form $v(x) = c_0 + c_1 x$ on I.

Perhaps the most natural basis for $P_1(I)$ is the monomial basis $\{1, x\}$, since any function v in $P_1(I)$ can be written as a linear combination of 1 and x. That is, a constant c_0 times 1 plus another constant c_1 times x. In doing so, v is clearly determined by specifying c_0 and c_1, the so-called coefficients of the linear combination. Indeed, we say that v has two degrees of freedom.

However, c_0 and c_1 are not the only degrees of freedom possible for v. To see this, recall that a line, or linear function, is uniquely determined by requiring it to pass through any two given points. Now, obviously, there are many pairs of points that can specify the same line. For example, $(0, 1)$ and $(2, 3)$ can be used to specify $v = x + 1$, but so can $(-1, 0)$ and $(4, 5)$. In fact, any pair of points within the interval I will do as degrees of freedom for v. In particular, v can be uniquely determined by its values $\alpha_0 = v(x_0)$ and $\alpha_1 = v(x_1)$ at the end-points x_0 and x_1 of I.

M.G. Larson and F. Bengzon, *The Finite Element Method: Theory, Implementation, and Applications*, Texts in Computational Science and Engineering 10,
DOI 10.1007/978-3-642-33287-6_1, © Springer-Verlag Berlin Heidelberg 2013

To prove this, let us assume that the values $\alpha_0 = v(x_0)$ and $\alpha_1 = v(x_1)$ are given. Inserting $x = x_0$ and $x = x_1$ into $v(x) = c_0 + c_1 x$ we obtain the linear system

$$\begin{bmatrix} 1 & x_0 \\ 1 & x_1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} \tag{1.2}$$

for c_i, $i = 1, 2$.

Computing the determinant of the system matrix we find that it equals $x_1 - x_0$, which also happens to be the length of the interval I. Hence, the determinant is positive, and therefore there exist a unique solution to (1.2) for any right hand side vector. Moreover, as a consequence, there is exactly one function v in $P_1(I)$, which has the values α_0 and α_1 at x_0 and x_1, respectively. In the following we shall refer to the points x_0 and x_1 as the nodes.

We remark that the system matrix above is called a Vandermonde matrix.

Knowing that we can completely specify any function in $P_1(I)$ by its node values α_0 and α_1 we now introduce a new basis $\{\lambda_0, \lambda_1\}$ for $P_1(I)$. This new basis is called a nodal basis, and is defined by

$$\lambda_j(x_i) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}, \quad i, j = 0, 1 \tag{1.3}$$

From this definition we see that each basis function λ_j, $j = 0, 1$, is a linear function, which takes on the value 1 at node x_j, and 0 at the other node.

The reason for introducing the nodal basis is that it allows us to express any function v in $P_1(I)$ as a linear combination of λ_0 and λ_1 with α_0 and α_1 as coefficients. Indeed, we have

$$v(x) = \alpha_0 \lambda_0(x) + \alpha_1 \lambda_1(x) \tag{1.4}$$

This is in contrast to the monomial basis, which given the node values requires inversion of the Vandermonde matrix to determine the corresponding coefficients c_0 and c_1.

The nodal basis functions take the following explicit form on I

$$\lambda_0(x) = \frac{x_1 - x}{x_1 - x_0}, \qquad \lambda_1(x) = \frac{x - x_0}{x_1 - x_0} \tag{1.5}$$

This follows directly from the definition (1.3), or by solving the linear system (1.2) with $[1,\ 0]^T$ and $[0,\ 1]^T$ as right hand sides.

1.1.2 The Space of Continuous Piecewise Linear Polynomials

A natural extension of linear functions is piecewise linear functions. In constructing a piecewise linear function, v, the basic idea is to first subdivide the domain of v into smaller subintervals. On each subinterval v is simply given by a linear function.

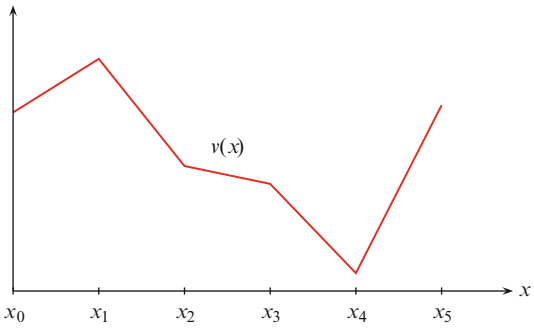

Fig. 1.1 A continuous piecewise linear function v

Continuity of v between adjacent subintervals is enforced by placing the degrees of freedom at the start- and end-points of the subintervals. We shall now formalize this more mathematically stringent.

Let $I = [0, L]$ be an interval and let the $n + 1$ node points $\{x_i\}_{i=0}^n$ define a partition

$$\mathcal{I} : 0 = x_0 < x_1 < x_2 < \ldots < x_{n-1} < x_n = L \tag{1.6}$$

of I into n subintervals $I_i = [x_{i-1}, x_i]$, $i = 1, 2 \ldots, n$, of length $h_i = x_i - x_{i-1}$. We refer to the partition $\mathcal{I}$ as to a mesh.

On the mesh $\mathcal{I}$ we define the space V_h of continuous piecewise linear functions by

$$V_h = \{v : v \in C^0(I),\ v|_{I_i} \in P_1(I_i)\} \tag{1.7}$$

where $C^0(I)$ denotes the space of continuous functions on I, and $P_1(I_i)$ denotes the space of linear functions on I_i. Thus, by construction, the functions in V_h are linear on each subinterval I_i, and continuous on the whole interval I. An example of such a function is shown in Fig. 1.1

It should be intuitively clear that any function v in V_h is uniquely determined by its nodal values

$$\{v(x_i)\}_{i=0}^n \tag{1.8}$$

and, conversely, that for any set of given nodal values $\{\alpha_i\}_{i=0}^n$ there exist a function v in V_h with these nodal values. Motivated by this observation we let the nodal values define our degrees of freedom and introduce a basis $\{\varphi_j\}_{j=0}^n$ for V_h associated with the nodes and such that

$$\varphi_j(x_i) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}, \quad i, j = 0, 1, \ldots, n \tag{1.9}$$

The resulting basis functions are depicted in Fig. 1.2.

Because of their shape the basis functions φ_i are often called hat functions. Each hat function is continuous, piecewise linear, and takes a unit value at its own node x_i, while being zero at all other nodes. Consequently, φ_i is only non-zero on the two

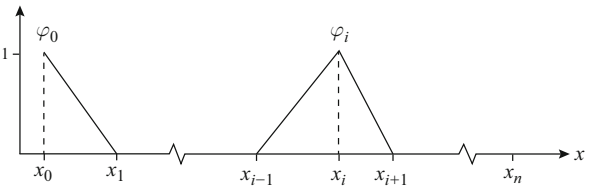

Fig. 1.2 A typical hat function φ_i on a mesh. Also shown is the "half hat" φ_0

intervals I_i and I_{i+1} containing node x_i. Indeed, we say that the support of φ_i is $I_i \cup I_{i+1}$. The exception is the two "half hats" φ_0 and φ_n at the leftmost and rightmost nodes $a = x_0$ and $x_n = b$ with support only on one interval.

By construction, any function v in V_h can be written as a linear combination of hat functions $\{\varphi_i\}_{i=0}^n$ and corresponding coefficients $\{\alpha_i\}_{i=0}^n$ with $\alpha_i = v(x_i)$, $i = 0, 1, \ldots, n$, the nodal values of v. That is,

$$v(x) = \sum_{i=0}^{n} \alpha_i \varphi_i(x) \tag{1.10}$$

The explicit expressions for the hat functions are given by

$$\varphi_i = \begin{cases} (x - x_{i-1})/h_i, & \text{if } x \in I_i \\ (x_{i+1} - x)/h_{i+1}, & \text{if } x \in I_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{1.11}$$

1.2 Interpolation

We shall now use the function spaces $P_1(I)$ and V_h to construct approximations, one from each space, to a given function f. The method we are going to use is very simple and only requires the evaluation of f at the node points. It is called interpolation.

1.2.1 Linear Interpolation

As before, we start on a single interval $I = [x_0, x_1]$. Given a continuous function f on I, we define the linear interpolant $\pi f \in P_1(I)$ to f by

$$\pi f(x) = f(x_0)\varphi_0 + f(x_1)\varphi_1 \tag{1.12}$$

We observe that interpolant approximates f in the sense that the values of πf and f are the same at the nodes x_0 and x_1 (i.e., $\pi f(x_0) = f(x_0)$ and $\pi f(x_1) = f(x_1)$).

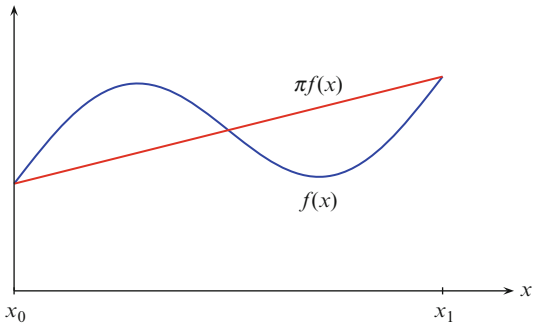

Fig. 1.3 A function f and its linear interpolant πf

In Fig. 1.3 we show a function f and its linear interpolant πf.

Unless f is linear, πf will only approximate f, and it is therefore of interest to measure the difference $f - \pi f$, which is called the interpolation error. To this end, we need a norm. Now, there are many norms and it is not easy to know which is the best. For instance, should we measure the interpolation error in the infinity norm, defined by

$$\|v\|_\infty = \max_{x \in I} |v(x)| \tag{1.13}$$

or the $L^2(I)$-norm defined, for any square integrable function v on I, by

$$\|v\|_{L^2(I)} = \left(\int_I v^2 \, dx \right)^{1/2} \tag{1.14}$$

We shall use the latter norm, since it captures the average size of a function, whereas the former only captures the pointwise maximum.

In this context we recall that the $L^2(I)$-norm, or any norm for that matter, obeys the Triangle inequality

$$\|v + w\|_{L^2(I)} \leq \|v\|_{L^2(I)} + \|w\|_{L^2(I)} \tag{1.15}$$

as well as the Cauchy-Schwarz inequality

$$\int_I vw \, dx \leq \|v\|_{L^2(I)} \|w\|_{L^2(I)} \tag{1.16}$$

for any two functions v and w in $L^2(I)$.

Then, using the L^2-norm to measure the interpolation error, we have the following results.

Proposition 1.1. *The interpolant πf satisfies the estimates*

$$\|f - \pi f\|_{L^2(I)} \leq Ch^2 \|f''\|_{L^2(I)} \tag{1.17}$$

$$\|(f - \pi f)'\|_{L^2(I)} \leq Ch \|f''\|_{L^2(I)} \tag{1.18}$$

where C is a constant, and $h = x_1 - x_0$.

Proof. Let $e = f - \pi f$ denote the interpolation error.

From the fundamental theorem of calculus we have, for any point y in I,

$$e(y) = e(x_0) + \int_{x_0}^{y} e' \, dx \tag{1.19}$$

where $e(x_0) = f(x_0) - \pi f(x_0) = 0$ due to the definition of πf.

Now, using the Cauchy-Schwarz inequality we have

$$\begin{aligned} e(y) &= \int_{x_0}^{y} e' \, dx && (1.20)\\ &\leq \int_{x_0}^{y} |e'| \, dx && (1.21)\\ &\leq \int_I 1 \cdot |e'| \, dx && (1.22)\\ &\leq \left(\int_I 1^2 \, dx \right)^{1/2} \left(\int_I e'^2 \, dx \right)^{1/2} && (1.23)\\ &= h^{1/2} \left(\int_I e'^2 \, dx \right)^{1/2} && (1.24) \end{aligned}$$

or, upon squaring both sides,

$$e(y)^2 \leq h \int_I e'^2 \, dx = h \|e'\|^2_{L^2(I)} \tag{1.25}$$

Integrating this inequality over I we further have

$$\|e\|^2_{L^2(I)} = \int_I e(y)^2 \, dy \leq \int_I h \|e'\|^2_{L^2(I)} \, dy = h^2 \|e'\|^2_{L^2(I)} \tag{1.26}$$

since the integrand to the right of the inequality is independent of y. Thus, we have

$$\|e\|_{L^2(I)} \leq h \|e'\|_{L^2(I)} \tag{1.27}$$

With a similar, but slightly different argument, we also have

$$\|e'\|_{L^2(I)} \leq h \|e''\|_{L^2(I)} \tag{1.28}$$

Hence, we conclude that

$$\|e\|_{L^2(I)} \leq h \|e'\|_{L^2(I)} \leq h^2 \|e''\|_{L^2(I)} \tag{1.29}$$

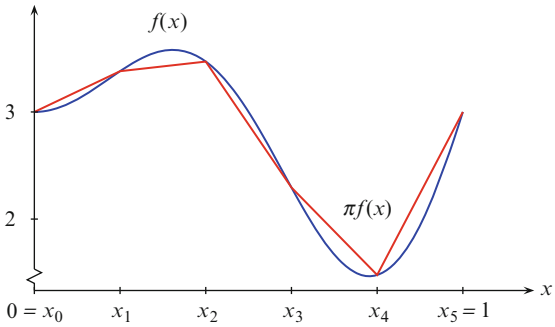

Fig. 1.4 The function $f(x) = 2x\sin(2\pi x) + 3$ and its continuous piecewise linear interpolant $\pi f(x)$ on a uniform mesh of $I = [0, 1]$ with six nodes x_i, $i = 0, 1, \ldots, 5$

from which the first inequality of the proposition follows by noting that since πf is linear $e'' = f''$. The second inequality of the proposition follows similarly from (1.26)

The difference in argument between deriving (1.27) and (1.28) has to do with the fact that we can not simply replace e with e' in (1.19), since $e'(x_0) \neq 0$. However, noting that $e(x_0) = e(x_1) = 0$, there exist by Rolle's theorem a point $\bar{x}$ in I such that $e'(\bar{x}) = 0$, which means that

$$e'(y) = e'(\bar{x}) + \int_{\bar{x}}^{y} e''\,dx = \int_{\bar{x}}^{y} e''\,dx \tag{1.30}$$

Starting instead from this and proceeding as shown above (1.28) follows. □

Examining the proof of Proposition 1.1 we note that the constant C equals unity and could equally well be left out. We have, however, chosen to retain this constant, since the estimates generalize to higher spatial dimensions, where C is not unity. The important thing to understand is how the interpolation error depends on the interpolated function f, and the size of the interval h.

1.2.2 Continuous Piecewise Linear Interpolation

It is straight forward to extend the concept of linear interpolation on a single interval to continuous piecewise linear interpolation on a mesh. Indeed, given a continuous function f on the interval $I = [0, L]$, we define its continuous piecewise linear interpolant $\pi f \in V_h$ on a mesh $\mathcal{I}$ of I by

$$\pi f(x) = \sum_{i=1}^{n} f(x_i)\varphi_i(x) \tag{1.31}$$

Figure 1.4 shows the continuous piecewise linear interpolant $\pi f(x)$ to $f(x) = 2x\sin(2\pi x) + 3$ on a uniform mesh of $I = [0, 1]$ with 6 nodes.

Regarding the interpolation error $f - \pi f$ we have the following results.

Proposition 1.2. *The interpolant πf satisfies the estimates*

$$\|f - \pi f\|^2_{L^2(I)} \leq C \sum_{i=1}^{n} h_i^4 \|f''\|^2_{L^2(I_i)} \tag{1.32}$$

$$\|(f - \pi f)'\|^2_{L^2(I)} \leq C \sum_{i=1}^{n} h_i^2 \|f''\|^2_{L^2(I_i)} \tag{1.33}$$

Proof. Using the Triangle inequality and Proposition 1.1, we have

$$\|f - \pi f\|^2_{L^2(I)} = \sum_{i=1}^{n} \|f - \pi f\|^2_{L^2(I_i)} \tag{1.34}$$

$$\leq \sum_{i=1}^{n} C h_i^4 \|f''\|^2_{L^2(I_i)} \tag{1.35}$$

which proves the first estimate. The second follows similarly. □

Proposition 1.2 says that the interpolation error vanish as the mesh size h_i tends to zero. This is natural, since we expect the interpolant πf to be a better approximation to f where ever the mesh is fine. The proposition also says that if the second derivative f'' of f is large then the interpolation error is also large. This is also natural, since if the graph of f bends a lot (i.e., if f'' is large) then f is hard to approximate using a piecewise linear function.

1.3 L^2-Projection

Interpolation is a simple way of approximating a continuous function, but there are, of course, other ways. In this section we shall study so-called orthogonal-, or L^2-projection. L^2-projection gives a so to speak good on average approximation, as opposed to interpolation, which is exact at the nodes. Moreover, in contrast to interpolation, L^2-projection does not require the function we seek to approximate to be continuous, or have well-defined node values.

1.3.1 Definition

Given a function $f \in L^2(I)$ the L^2-projection $P_h f \in V_h$ of f is defined by

$$\int_I (f - P_h f) v \, dx = 0, \quad \forall v \in V_h \tag{1.36}$$

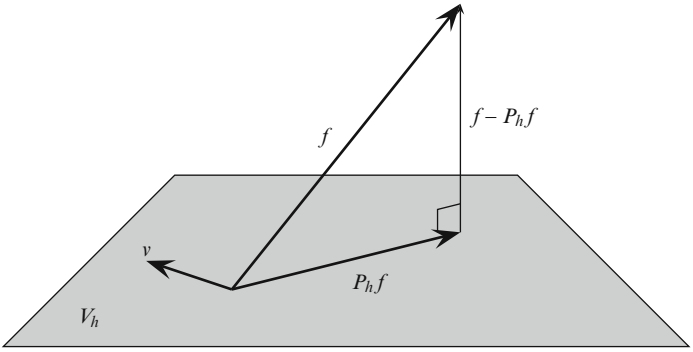

Fig. 1.5 Illustration of the function f and its L^2-projection $P_h f$ on the space V_h

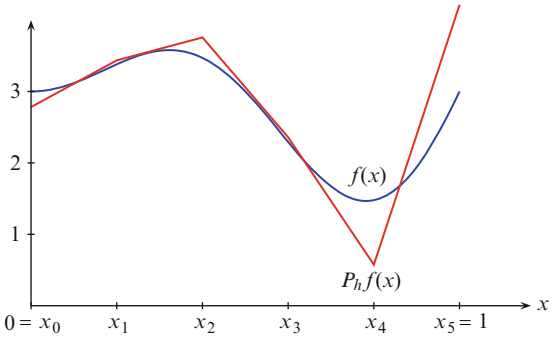

Fig. 1.6 The function $f(x) = 2x\sin(2\pi x) + 3$ and its L^2-projection $P_h f$ on a uniform mesh of $I = [0, 1]$ with six nodes, x_i, $i = 1, 2, \ldots, 6$

In analogy with projection onto subspaces of $\mathbb{R}^n$, (1.34) defines a projection of f onto V_h, since the difference $f - P_h f$ is required to be orthogonal to all functions v in V_h. This is illustrated in Fig. 1.5.

As we shall see later on, $P_h f$ is the minimizer of $\min_{v \in V_h} \|f - v\|_{L^2(I)}$, and therefore we say that it approximates f in a least squares sense. In fact, $P_h f$ is the best approximation to f when measuring the error $f - P_h f$ in the L^2-norm.

In Fig. 1.6 we show the L^2-projection of $f(x) = 2x\sin(2\pi x) + 3$ computed on the same mesh as was used for showing the continuous piecewise linear interpolant πf in Fig. 1.4. It is instructive to compare these two approximations because it highlights their different characteristics. The interpolant πf approximates f exactly at the nodes, while the L^2-projection $P_h f$ approximates f on average. In doing so, it is common for $P_h f$ to over and under shoot local maxima and minima of f, respectively. Also, both the interpolant and the L^2-projection have difficulty with approximating rapidly oscillating or discontinuous functions unless the node positions are adjusted appropriately.

1.3.2 Derivation of a Linear System of Equations

In order to actually compute the L^2-projection $P_h f$, we first note that the definition (1.36) is equivalent to

$$\int_I (f - P_h f)\varphi_i \, dx = 0, \quad i = 0, 1, \ldots, n \tag{1.37}$$

where φ_i, $i = 0, 1, \ldots, n$, are the hat functions. This is a consequence of the fact that if (1.36) is satisfied for v anyone of the hat functions, then it is also satisfied for v a linear combination of hat functions. Conversely, since any function v in V_h is precisely such a linear combination of hat functions, (1.37) implies (1.36).

Then, since $P_h f$ belongs to V_h it can be written as the linear combination

$$P_h f = \sum_{j=0}^{n} \xi_j \varphi_j \tag{1.38}$$

where ξ_j, $j = 0, 1, \ldots, n$, are $n + 1$ unknown coefficients to be determined.

Inserting the ansatz (1.38) into (1.37) we get

$$\int_I f\varphi_i \, dx = \int_I \left(\sum_{j=0}^{n} \xi_j \varphi_j \right) \varphi_i \, dx \tag{1.39}$$

$$= \sum_{j=0}^{n} \xi_j \int_I \varphi_j \varphi_i \, dx, \quad i = 0, 1, \ldots, n \tag{1.40}$$

Further, introducing the notation

$$M_{ij} = \int_I \varphi_j \varphi_i \, dx, \quad i, j = 0, 1, \ldots, n \tag{1.41}$$

$$b_i = \int_I f\varphi_i \, dx, \quad i = 0, 1, \ldots, n \tag{1.42}$$

we have

$$b_i = \sum_{j=0}^{n} M_{ij} \xi_j, \quad i = 0, 1, \ldots, n \tag{1.43}$$

which is an $(n + 1) \times (n + 1)$ linear system for the $n + 1$ unknown coefficients ξ_j, $j = 0, 1, \ldots, n$. In matrix form, we write this

$$M\xi = b \tag{1.44}$$

where the entries of the $(n+1)\times(n+1)$ matrix M and the $(n+1)\times 1$ vector b are defined by (1.41) and (1.42), respectively.

We, thus, conclude that the coefficients ξ_j, $j = 0, 1, \ldots, n$ in the ansatz (1.38) satisfy a square linear system, which must be solved in order to obtain the L^2-projection $P_h f$.

For historical reasons we refer to M as the mass matrix and to b as the load vector.

1.3.3 Basic Algorithm to Compute the L^2-Projection

The following algorithm summarizes the basic steps for computing the L^2-projection $P_h f$:

Algorithm 1 Basic algorithm to compute the L^2-projection

1: Create a mesh with n elements on the interval I and define the corresponding space of continuous piecewise linear functions V_h.

2: Compute the $(n+1)\times(n+1)$ matrix M and the $(n+1)\times 1$ vector b, with entries

$$M_{ij} = \int_I \varphi_j \varphi_i \, dx, \qquad b_i = \int_I f\varphi_i \, dx \tag{1.45}$$

3: Solve the linear system

$$M\xi = b \tag{1.46}$$

4: Set

$$P_h f = \sum_{j=0}^{n} \xi_j \varphi_j \tag{1.47}$$

1.3.4 A Priori Error Estimate

Naturally, we are interested in knowing how good $P_h f$ is at approximating f. In particular, we wish to derive bounds for the error $f - P_h f$. The next theorem gives a key result for deriving such error estimates. It is a so-called a best approximation result.

Theorem 1.1. *The L^2-projection $P_h f$, defined by (1.36), satisfies the best approximation result*

$$\|f - P_h f\|_{L^2(I)} \leq \|f - v\|_{L^2(I)}, \quad \forall v \in V_h \tag{1.48}$$

Proof. Using the definition of the L^2-norm and writing $f - P_h f = f - v + v - P_h f$, with v an arbitrary function in V_h, we have

$$\|f - P_h f\|_{L^2(I)}^2 = \int_I (f - P_h f)(f - v + v - P_h f)\, dx \tag{1.49}$$

$$= \int_I (f - P_h f)(f - v)\, dx + \int_I (f - P_h f)(v - P_h f)\, dx \tag{1.50}$$

$$= \int_I (f - P_h f)(f - v)\, dx \tag{1.51}$$

$$\leq \|f - P_h f\|_{L^2(I)} \|f - v\|_{L^2(I)} \tag{1.52}$$

where we used the definition of the L^2-projection to conclude that

$$\int_I (f - P_h f)(v - P_h f)\, dx = 0 \tag{1.53}$$

since $v - P_h f \in V_h$. Dividing by $\|f - P_h f\|_{L^2(I)}$ concludes the proof. □

This shows that $P_h f$ is the closest of all functions in V_h to f when measuring in the L^2-norm. Hence, the name best approximation result.

We can use best approximation result together with interpolation estimates to study how the error $f - P_h f$ depends on the mesh size. In doing so, we have the following basic so-called a priori error estimate.

Theorem 1.2. *The L^2-projection $P_h f$ satisfies the estimate*

$$\|f - P_h f\|_{L^2(I)}^2 \leq C \sum_{i=1}^{n} h_i^4 \|f''\|_{L^2(I_i)}^2 \tag{1.54}$$

Proof. Starting from the best approximation result, choosing $v = \pi f$ the interpolant of f, and using the interpolation error estimate of Proposition 1.1, we have

$$\|f - P_h f\|_{L^2(I)}^2 \leq \|f - \pi f\|_{L^2(I)}^2 \tag{1.55}$$

$$\leq \sum_{i=1}^{n} \|f - \pi f\|_{L^2(I_i)}^2 \tag{1.56}$$

$$\leq \sum_{i=1}^{n} C h_i^4 \|f''\|_{L^2(I_i)}^2 \tag{1.57}$$

which proves the estimate. □

Defining $h = \max_{1 \leq i \leq n} h_i$ we conclude that

$$\|f - P_h f\|_{L^2(I)} \leq C h^2 \|f''\|_{L^2(I)} \tag{1.58}$$

Thus, the L^2-error $\|f - P_h f\|_{L^2(I)}$ tends to zero as the maximum mesh size h tends to zero.

1.4 Quadrature

To compute the L_2-projection we need to compute the mass matrix M whose entries are integrals involving products of hat functions. One way of doing this is to use quadrature, or, numerical integration. To this end, f be a continuous function on the interval $I = [x_0, x_1]$, and consider the problem of evaluating, approximately, the integral

$$J = \int_I f(x)\,dx \tag{1.59}$$

A quadrature rule is a formula that is used to compute integrals approximately. It it usually derived by first interpolating the integrand f by a polynomial and then integrating the interpolant. Depending on the degree of the interpolating polynomial one obtains quadrature rules of different computational complexity and accuracy. Evaluating a quadrature rule generally involves summing values of the integrand f at a set of carefully selected quadrature points within the interval I times the interval length $h = x_1 - x_0$. We shall next describe three classical quadrature rules called the Mid-point rule, the Trapezoidal rule, and Simpson's formula, which corresponds to using polynomial interpolation of degree 0, 1, and 2 on f, respectively.

1.4.1 The Mid-Point Rule

Interpolating f with the constant $f(m)$, where $m = (x_0 + x_1)/2$ is the mid-point of I, we get

$$J \approx f(m)h \tag{1.60}$$

which is the Mid-point rule. Geometrically this means that we approximate the area under the integrand f with the area of the square $f(m)h$, see Fig. 1.7. The Mid-point rule integrates linear polynomials exactly.

1.4.2 The Trapezoidal Rule

Continuing, interpolating f with the line passing through the points $(x_0, f(x_0))$ and $(x_1, f(x_1))$ we get

$$J \approx \frac{f(x_0) + f(x_1)}{2}h \tag{1.61}$$

which is the Trapezoidal rule. Geometrically this means that we approximate the area under f with the area under the trapezoidal with the four corner points $(x_0, 0)$,

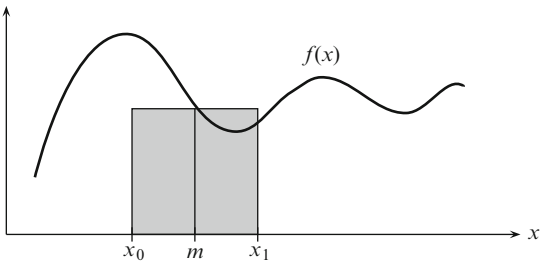

Fig. 1.7 The area of the *shaded square* approximates $J = \int_I f(x)\,dx$

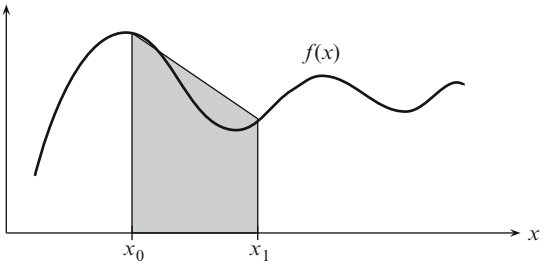

Fig. 1.8 The area of the *shaded* trapezoidal approximates $J = \int_I f(x)\,dx$

$(x_0, f(x_0))$, $(x_1, 0)$, and $(x_1, f(x_1))$, see Fig. 1.8. The Trapezoidal rule is also exact for linear polynomials.

1.4.3 Simpson's Formula

This rule corresponds to a quadratic interpolant using the end-points and the mid-point of the interval I as nodes. To simplify things a bit let $I = [0, l]$ be the interval of integration and let $g(x) = c_0 + c_1 x + c_2 x^2$ be the interpolant. Since g interpolates f at the points $(0, f(0))$, $(\frac{l}{2}, f(\frac{l}{2}))$, and $(l, f(l))$ (i.e., its graph passes trough these points) their coordinates must satisfy the equation for g. This yields the following linear system for c_0, c_1, and c_2.

$$\begin{bmatrix} 0 & 0 & 1 \\ \frac{1}{4}l^2 & \frac{1}{2}l & 1 \\ l^2 & l & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} f(0) \\ f(\frac{l}{2}) \\ f(l) \end{bmatrix} \tag{1.62}$$

Solving this one readily finds

$$c_0 = 2(f(0) - 2f(\tfrac{l}{2}) + f(l))/l^2, \quad c_1 = -(3f(0) - 4f(\tfrac{l}{2}) + f(l))/l, \quad c_2 = f(0) \tag{1.63}$$

Now, integrating g from 0 to l one eventually ends up with

$$\int_0^l g(x)\,dx = \frac{f(0) + 4f(\frac{1}{2}l) + f(l)}{6} l \tag{1.64}$$

which is Simpson's formula.

On the interval $I = [x_0, x_1]$ Simpson's formula takes the form

$$J \approx \frac{f(x_0) + 4f(m) + f(x_1)}{6} h \tag{1.65}$$

with $m = \frac{1}{2}(x_0 + x_1)$ and $h = x_1 - x_0$.

Simpson's formula is exact for third order polynomials.

1.5 Computer Implementation

1.5.1 Assembly of the Mass Matrix

Having studied various quadrature rules, let us now go through the nitty gritty details of how to assemble the mass matrix M and load vector b. We begin by calculating the entries $M_{ij} = \int_I \varphi_i \varphi_j\,dx$ of the mass matrix. Because each hat φ_i is a linear polynomial the product of two hats is a quadratic polynomial. Thus, Simpson's formula can be used to integrate M_{ij} exactly. In doing so, since the hats φ_i and φ_j lack common support for $|i - j| > 1$ only M_{ii}, M_{ii+1}, and M_{i+1i} need to be calculated. All other matrix entries are zero by default. This is clearly seen from Fig. 1.9 showing two neighbouring hat functions and their support. This leads to the observation that the mass matrix M is tridiagonal.

Starting with the diagonal entries M_{ii} and using Simpson's formula we have

$$\begin{aligned} M_{ii} &= \int_I \varphi_i^2\,dx && (1.66)\\ &= \int_{x_{i-1}}^{x_i} \varphi_i^2\,dx + \int_{x_i}^{x_{i+1}} \varphi_i^2\,dx && (1.67)\\ &= \frac{0 + 4\cdot(\frac{1}{2})^2 + 1}{6} h_i + \frac{1 + 4\cdot(\frac{1}{2})^2 + 0}{6} h_{i+1} && (1.68)\\ &= \frac{h_i}{3} + \frac{h_{i+1}}{3}, \quad i = 1, 2, \ldots, n-1 && (1.69) \end{aligned}$$

where $x_i - x_{i-1} = h_i$ and $x_{i+1} - x_i = h_{i+1}$. The first and last diagonal entry are $M_{00} = h_1/3$ and $M_{nn} = h_n/3$, respectively, since the hat functions φ_0 and φ_n are only half.

Continuing with the subdiagonal entries M_{i+1i}, still using Simpson's formula, we have

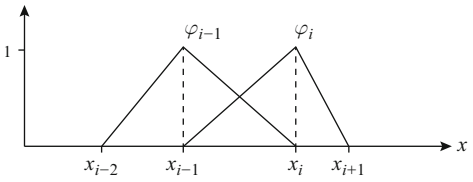

Fig. 1.9 Illustration of the hat functions φ_{i-1} and φ_i and their support

$$M_{i+1\,i} = \int_I \varphi_i \varphi_{i+1}\, dx \tag{1.70}$$

$$= \int_{x_i}^{x_{i+1}} \varphi_i \varphi_{i+1}\, dx \tag{1.71}$$

$$= \frac{1 \cdot 0 + 4 \cdot (\frac{1}{2})^2 + 0 \cdot 1}{6} h_{i+1} \tag{1.72}$$

$$= \frac{h_{i+1}}{6}, \quad i = 0, 1, \ldots, n \tag{1.73}$$

A similar calculation shows that the superdiagonal entries are $M_{i\,i+1} = h_{i+1}/6$.

Hence, the mass matrix takes the form

$$M = \begin{bmatrix} \frac{h_1}{3} & \frac{h_1}{6} & & & & & \\ \frac{h_1}{6} & \frac{h_1}{3} + \frac{h_2}{3} & \frac{h_2}{6} & & & & \\ & \frac{h_2}{6} & \frac{h_2}{3} + \frac{h_3}{3} & \frac{h_3}{6} & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \frac{h_{n-1}}{6} & \frac{h_{n-1}}{3} + \frac{h_n}{3} & \frac{h_n}{6} & \\ & & & & \frac{h_n}{6} & \frac{h_n}{3} \end{bmatrix} \tag{1.74}$$

From (1.74) it is evident that the global mass matrix M can be written as a sum of n simple matrices, viz.,

$$M = \begin{bmatrix} \frac{h_1}{3} & \frac{h_1}{6} \\ \frac{h_1}{6} & \frac{h_1}{3} \end{bmatrix} + \begin{bmatrix} \frac{h_2}{3} & \frac{h_2}{6} \\ \frac{h_2}{6} & \frac{h_2}{3} \end{bmatrix} + \ldots + \begin{bmatrix} & \\ & \frac{h_n}{3} \quad \frac{h_n}{6} \\ & \frac{h_n}{6} \quad \frac{h_n}{3} \end{bmatrix} \tag{1.75}$$

$$= M^{I_1} + M^{I_2} + \ldots + M^{I_n} \tag{1.76}$$

Each matrix M^{I_i}, $i = 1, 2 \ldots, n$, is obtained by restricting integration to one subinterval, or element, I_i and is therefore called a global element mass matrix. In practice, however, these matrices are never formed, since it suffice to compute their 2×2 blocks of non-zero entries. From the sum (1.75) we see that on each

element I this small block takes the form

$$M^I = \frac{1}{6}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} h \tag{1.77}$$

where h is the length of I. We refer to M^I as the local element mass matrix.

The summation of the element mass matrices into the global mass matrix is called assembling. The assembly process lies at the very heart of finite element programming because it allows the forming of the mass matrix through the use of a single loop over the elements. It also generalizes to higher dimensions.

The following algorithm summarizes how to assemble the mass matrix M:

Algorithm 2 Assembly of the mass matrix

1: Allocate memory for the $(n+1) \times (n+1)$ matrix M and initialize all matrix entries to zero.
2: **for** $i = 1, 2, \ldots, n$ **do**
3: Compute the 2×2 local element mass matrix M^I given by

$$M^I = \frac{1}{6}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} h \tag{1.78}$$

where h is the length of element I_i.
4: Add M^I_{11} to M_{ii}
5: Add M^I_{12} to M_{ii+1}
6: Add M^I_{21} to M_{i+1i}
7: Add M^I_{22} to M_{i+1i+1}
8: **end for**

The following MATLAB routine assembles the mass matrix.

```
function M = MassAssembler1D(x)
n = length(x)-1; % number of subintervals
M = zeros(n+1,n+1); % allocate mass matrix
for i = 1:n % loop over subintervals
    h = x(i+1) - x(i); % interval length
    M(i,i) = M(i,i) + h/3; % add h/3 to M(i,i)
    M(i,i+1) = M(i,i+1) + h/6;
    M(i+1,i) = M(i+1,i) + h/6;
    M(i+1,i+1) = M(i+1,i+1) + h/3;
end
```

Input to this routine is a vector `x` holding the node coordinates. Output is the global mass matrix.

1.5.2 Assembly of the Load Vector

We next calculate the load vector b. Because the entries $b_i = \int_i f\varphi_i \, dx$ depend on the function f, we can not generally expect to calculate them exactly. However, we can approximate entry b_i using a quadrature rule. Using the Trapezoidal rule, for instance, we have

$$
\begin{aligned}
b_i &= \int_I f\varphi_i \, dx && (1.79)\\
&= \int_{x_{i-1}}^{x_{i+1}} f\varphi_i \, dx && (1.80)\\
&= \int_{x_{i-1}}^{x_i} f\varphi_i \, dx + \int_{x_i}^{x_{i+1}} f\varphi_i \, dx && (1.81)\\
&\approx (f(x_{i-1})\varphi_i(x_{i-1}) + f(x_i)\varphi_i(x_i))h_i/2 && (1.82)\\
&\quad + (f(x_i)\varphi_i(x_i) + f(x_{i+1})\varphi_i(x_{i+1}))h_{i+1}/2 && (1.83)\\
&= (0 + f(x_i))h_i/2 + (f(x_i) + 0)h_{i+1}/2 && (1.84)\\
&= f(x_i)(h_i + h_{i+1})/2 && (1.85)
\end{aligned}
$$

The approximate load vector then takes the form

$$
b = \begin{bmatrix} f(x_0)h_1/2 \\ f(x_1)(h_1 + h_2)/2 \\ f(x_2)(h_2 + h_3)/2 \\ \vdots \\ f(x_{n-1})(h_{n-1} + h_n)/2 \\ f(x_n)h_n/2 \end{bmatrix} \tag{1.86}
$$

Splitting b into a sum over the elements yields the n global element load vectors b^{I_i}

$$
\begin{aligned}
b &= \begin{bmatrix} f(x_0) \\ f(x_1) \\ \\ \\ \end{bmatrix} h_1/2 + \begin{bmatrix} f(x_1) \\ f(x_2) \\ \\ \\ \end{bmatrix} h_2/2 + \ldots + \begin{bmatrix} \\ \\ \\ f(x_{n-1}) \\ f(x_n) \end{bmatrix} h_n/2 && (1.87)\\
&= b^{I_1} + b^{I_2} + \ldots + b^{I_n}. && (1.88)
\end{aligned}
$$

Each vector b^{I_i}, $i = 1, 2, \ldots, n$, is obtained by restricting integration to element I_i. The assembly of the load vector b is very similar to that of the mass matrix as the following algorithm shows:

Algorithm 3 Assembly of the load vector

1: Allocate memory for the $(n + 1) \times 1$ vector b and initialize all vector entries to zero.
2: **for** $i = 1, 2, \ldots, n$ **do**
3: Compute the 2×1 local element load vector b^I given by

$$b^I = \frac{1}{2} \begin{bmatrix} f(x_{i-1}) \\ f(x_i) \end{bmatrix} h \tag{1.89}$$

where h is the length of element I_i.
4: Add b_1^I to b_{i-1}
5: Add b_2^I to b_i
6: **end for**

A MATLAB routine for assembling the load vector is listed below.

```
function b = LoadAssembler1D(x,f)
n = length(x)-1;
b = zeros(n+1,1);
for i = 1:n
  h = x(i+1) - x(i);
  b(i) = b(i) + f(x(i))*h/2;
  b(i+1) = b(i+1) + f(x(i+1))*h/2;
end
```

Here, `f` is assumed to be a separate routine specifying the function f. This needs perhaps a little bit of explanation. MATLAB has a something called function handles, which provide a way of passing a routine as argument to another routine. For example, suppose we have written a routine called `Foo1` to specify the function $f(x) = x \sin(x)$

```
function y = Foo1(x)
y=x.*sin(x)
```

To assemble the corresponding load vector, we type

```
b = LoadAssembler1D(x,@Foo1)
```

This passes the routine `Foo1` as argument to `LoadAssembler1D` and allows it to be evaluated inside the assembler. The at sign `@` creates the function handle. Indeed, function handles provide means for writing flexible and reusable code.

In this context we mention that if `Foo1` is a small routine, then it can be inlined and called, viz.,

```
Foo1 = inline('x.*sin(x)','x')
b = LoadAssembler1D(x,Foo1)
```

Note that there is no at sign in the call to the load vector assembler.

Putting it all together we get the following main routine for computing L^2-projections.

```
function L2Projector1D()
n = 5; % number of subintervals
h = 1/n; % mesh size
x = 0:h:1; % mesh
M = MassAssembler1D(x); % assemble mass
b = LoadAssembler1D(x,@Foo1); % assemble load
Pf = M\b; % solve linear system
plot(x,Pf) % plot L^2 projection
```

1.6 Problems

Exercise 1.1. Let $I = [x_0, x_1]$. Verify by direct calculation that the basis functions

$$\lambda_0(x) = \frac{x_1 - x}{x_1 - x_0}, \qquad \lambda_1(x) = \frac{x - x_0}{x_1 - x_0}$$

for $P_1(I)$ satisfies $\lambda_0(x) + \lambda_1(x) = 1$ and $x_0\lambda_0(x) + x_1\lambda_1(x) = x$. Give a geometrical interpretation by drawing $\lambda_0(x)$, $\lambda_1(x)$, $\lambda_0(x) + \lambda_1(x)$, $x_0\lambda_0(x)$, $x_1\lambda_1(x)$ and $x_0\lambda_0(x) + x_1\lambda_1(x)$.

Exercise 1.2. Let $0 = x_0 < x_1 < x_2 < x_3 = 1$, where $x_1 = 1/6$ and $x_2 = 1/2$, be a partition of the interval $I = [0, 1]$ into three subintervals, and let V_h be the space of continuous piecewise linear functions on this partition.

(a) Determine analytical expressions for the hat function $\varphi_1(x)$ and draw it.
(b) Draw the function $v(x) = -\varphi_0(x) + \varphi_2(x) + 2\varphi_3(x)$ and its derivative $v'(x)$.
(c) Draw the piecewise constant mesh function $h(x) = h_i$ on each subinterval I_i.
(d) What is the dimension of V_h?

Exercise 1.3. Determine the linear interpolant $\pi f \in P_1(I)$ on the single interval $I = [0, 1]$ to the following functions f.

(a) $f(x) = x^2$.
(b) $f(x) = 3\sin(2\pi x)$.

Make plots of f and πf in the same figure.

Exercise 1.4. Let V_h be the space of all continuous piecewise linears on a uniform mesh with four nodes of $I = [0, 1]$. Draw the interpolant $\pi f \in V_h$ to the following functions f.

(a) $f(x) = x^2 + 1$.
(b) $f(x) = \cos(\pi x)$.

Can you think of a better partition of I assuming we are restricted to three subintervals?

Exercise 1.5. Calculate $\|f\|_\infty$ with $f = x(x-1/2)(x-1/3)$ on the interval $I = [0,1]$.

Exercise 1.6. Let $I = [0,1]$ and $f(x) = x^2$ for $x \in I$.

(a) Calculate $\int_I f\,dx$ analytically.
(b) Compute $\int_I f\,dx$ using the Mid-point rule.
(c) Compute $\int_I f\,dx$ using the Trapezoidal rule.
(d) Compute the quadrature errors in (b) and (c).

Exercise 1.7. Let $I = [0,1]$ and $f(x) = x^4$ for $x \in I$.

(a) Calculate $\int_I f\,dx$ analytically.
(b) Compute $\int_I f\,dx$ using Simpson's formula on the single interval I.
(c) Divide I into two equal subintervals and compute $\int_I f\,dx$ using Simpson's formula on each subinterval.
(d) Compute the quadrature errors in (b) and (c). By what factor has the error decreased?

Exercise 1.8. Let $I = [0,1]$ and let $f(x) = x^2$ for $x \in I$.

(a) Let V_h be the space $P_1(I)$ of linear functions on I. Compute the L^2-projection $P_h f \in V_h$ of f.
(b) Divide I into two subintervals of equal length and let V_h be the corresponding space V_h of continuous piecewise linear functions. Compute the L^2-projection $P_h f \in V_h$ of f.
(c) Plot your results and compare with the nodal interpolant πf.

Exercise 1.9. Show that $\int_\Omega (f - P_h f)v\,dx = 0$ for all $v \in V_h$, if and only if $\int_\Omega (f - P_h f)\varphi_i\,dx = 0$, for $i = 0, 1, \ldots, n$, where $\{\varphi_i\}_{i=0}^n \subset V_h$ is the usual basis of hat functions.

Exercise 1.10. Let $(f,g) = \int_I fg\,dx$ and $\|f\|_{L^2(I)}^2 = (f,f)$ denote the L^2-scalar product and norm, respectively. Also, let $I = [0,\pi]$, $f = x$, $g = \cos(x)$, and $h = 2\cos(3x)$ for $x \in I$.

(a) Calculate (f,g).
(b) Calculate (g,h). Are g and h orthogonal?
(c) Calculate $\|f\|_{L^2(I)}$ and $\|g\|_{L^2(I)}$.

Exercise 1.11. Let V be a linear subspace of $\mathbb{R}^n$ with basis $\{v_1, \ldots, v_m\}$ with $m < n$. Let $Px \in V$ be the orthogonal projection of $x \in \mathbb{R}^n$ onto the subspace V. Derive a linear system of equations that determines Px. Note that your results are analogous to the L^2-projection when the usual scalar product in $\mathbb{R}^n$ is replaced by the scalar product in $L^2(I)$. Compare this method of computing the projection Px to the method used for computing the projection of a three dimensional vector onto a two dimensional subspace. What happens if the basis $\{v_1, \ldots, v_m\}$ is L^2-orthogonal?

Exercise 1.12. Show that $\{1, x, (3x^2-1)/2\}$ form a basis for the space of quadratic polynomials $P_2(I)$, on $I = [-1, 1]$. Then compute and draw the L^2-projections $P_h f \in P_2(I)$ on I for the following two functions f.

(a) $f(x) = 1 + 2x$.
(b) $f(x) = x^3$.

Exercise 1.13. Show that the hat function basis $\{\varphi_j\}_{j=0}^n$ of V_h is almost orthogonal. How can we see that it is almost orthogonal by looking at the non-zero elements of the mass matrix? What can we say about the mass matrix if we had a fully orthogonal basis?

Exercise 1.14. Modify `L2Projector1D` and compute the L^2-projection $P_h f$ of the following functions f.

(a) $f(x) = 1$.
(b) $f(x) = x^3(x-1)(1-2x)$.
(c) $f(x) = \arctan((x-0.5)/\epsilon)$, with $\epsilon = 0.1$ and 0.01.

Use a uniform mesh $\mathcal{I}$ of the interval $I = [0, 1]$ with $n = 5, 25$, and 100 subintervals.

http://www.springer.com/978-3-642-33286-9

The Finite Element Method: Theory, Implementation, and Applications

Larson, M.G.; Bengzon, F.

2013, XVII, 395 p., Hardcover
ISBN: 978-3-642-33286-9