Mats G. Larson, Fredrik Bengzon

The Finite Element Method: Theory, Implementation, and Practice

November 9, 2010

Springer

Preface

This is a set of lecture notes on finite elements for the solution of partial differential equations. The approach taken is mathematical in nature with a strong focus on the underlying mathematical principles, such as approximation properties of piecewise polynomial spaces, and variational formulations of partial differential equations, but with a minimum level of advanced mathematical machinery from functional analysis and partial differential equations.

In principle, these lecture notes should be accessible to students with only a basic knowledge of calculus of several variables and linear algebra as the necessary concepts from more advanced analysis are introduced when needed.

Throughout this text we emphasize implementation of the involved algorithms, and have thus mixed mathematical theory with concrete computer code using the numerical software MATLAB and its PDE-Toolbox.

Umeå, *Mats G. Larson*
December 2009 *Fredrik Bengzon*

Acknowledgements

These notes are based on courses given at Chalmers University of Technology and Umeå University during the last six years and the authors gratefully acknowledge the contributions of the teachers and students involved.

Contents

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 proving 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 $\mathscr{P}_1(I)$ denote the vector space of linear functions on I, defined by

$$\mathscr{P}_1(I) = \{v : v(x) = c_0 + c_1 x,\ x \in I,\ c_0, c_1 \in \mathbb{R}\} \tag{1.1}$$

In other words $\mathscr{P}_1(I)$ contains all functions of the form $v(x) = c_0 + c_1 x$ on I.

Perhaps the most natural basis for $\mathscr{P}_1(I)$ is the monomial basis $\{1, x\}$, since any function v in $\mathscr{P}_1(I)$ 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. Obviously, v is uniquely determined by specifying c_0 and c_1 called the coefficients of this linear combination. We say that v have 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 two given points, and that 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 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.

To prove this claim rigorously 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 get the following linear system

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

Computing the determinant of the system matrix we find that it equals $x_1 - x_0$, which also happends to be the length of the interval I. Hence, the determinant is positive, and therefore there exist a unique solution to the linear system for any right hand side vector. Further, as a consequence there is exactly one function v in $\mathscr{P}_1(I)$ with has the values α_0 and α_1 at x_0 and x_1, respectively. We remark that the system matrix above is called a Vandermonde matrix.

In the following we shall refer to the points x_0 and x_1 as the nodes.

Knowing that we can 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 $\mathscr{P}_1(I)$. This new basis is called a nodal basis, and is defined by

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

Thus, 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 because it allows us to express any linear function v in $\mathscr{P}_1(I)$ 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 constrast 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

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

which follow 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. Loosely speaking the basic idea in contructing a piecewise linear function v is to first subdivide the domain of v into smaller subintervals. On each subinterval v is then simply given by a linear function. Continuity of v between adjacent subintervals may be enforced by placing the degrees of freedom at the common points shared by these subintervals. We shall now formalize this more mathematically stringent.

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

$$a = x_0 < x_1 < x_2 < \ldots < x_{n-1} < x_n = b \tag{1.6}$$

of this interval 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 as to a mesh.

On the mesh we define the space V_h of continuous piecewise linear functions by

$$V_h = \{v : v \in \mathscr{C}^0(I),\ v|_{I_i} \in \mathscr{P}_1(I_i)\} \tag{1.7}$$

where $\mathscr{C}^0(I)$ denotes the space of continuous functions on I, and $\mathscr{P}_1(I_i)$ 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 Figure 1.1

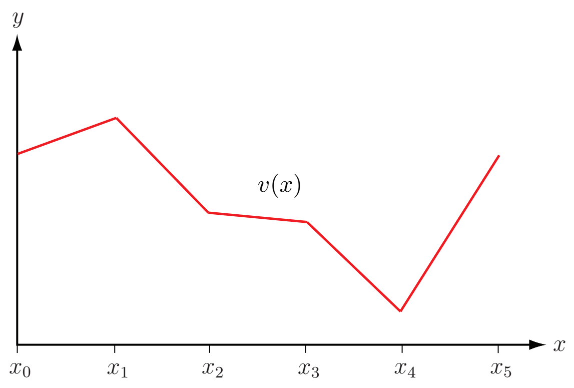


Fig. 1.1 A continuous piecewise linear function $v \in V_h$.

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 Figure 1.2.

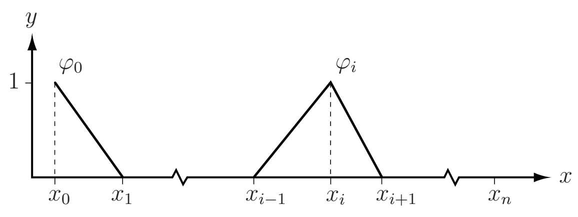


Fig. 1.2 A typical hat function φ_i on a mesh. Also shown is the "half hat" φ_0.

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. Thus, φ_i is only non-zero on the two intervals I_i and I_{i+1} containing node x_i. 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.

Due to the construction of the hat function basis, 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 $\alpha_i = v(x_i)$, $i = 0, 1, \dots, n$, 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 $\mathscr{P}_1(I)$ and V_h to construct approximations, one from each space, to a given function f. The approximation method 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 $\pi f \in \mathscr{P}_1(I)$ 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 $\pi f(x_1) = f(x_1)$).

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

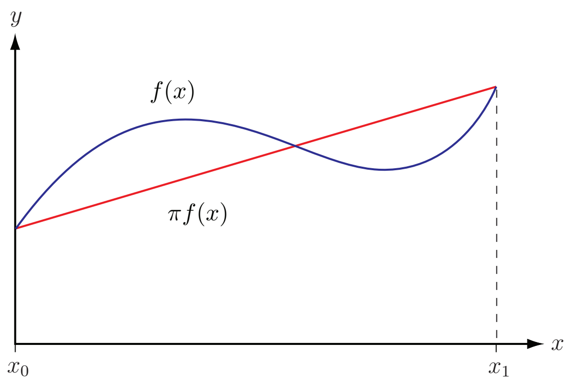


Fig. 1.3 A function f and its linear interpolant πf on the interval $I = [x_0, x_1]$.

Since generally πf only approximates f it is of interest to measure the difference $f - \pi f$ called the interpolation error. To this end we need a norm. Now, there are many norms and it is not obvious which norm to choose. For instance, should we measure the 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

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

For various reasons it turns out that the latter norm is a suitable norm, since it captures the average size of v, whereas the former only captures the pointwise maximum of v.

For later use we recall the Triangle inequality

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

and the Cauchy-Schwartz inequality

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

We then have the following result.

Proposition 1.1. *The following interpolation error estimates hold.*

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

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

with C 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

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

for any point y in I. We note that $e(x_0) = f(x_0) - \pi f(x_0) = 0$ by definition of πf. Now, using the Cauchy-Schwartz inequality we obtain

$$e(y) = \int_{x_0}^{y} e'\,dx \tag{1.20}$$

$$\leq \int_{x_0}^{y} |e'|\,dx \tag{1.21}$$

$$\leq \int_I 1 \cdot |e'|\,dx \tag{1.22}$$

$$\leq \left(\int_I 1^2\,dx\right)^{1/2} \left(\int_I e'^2\,dx\right)^{1/2} \tag{1.23}$$

$$= h^{1/2} \left(\int_I e'^2\,dx\right)^{1/2} \tag{1.24}$$

Hence, we have

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

Further, integrating this inequality over I we get

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

since the integrand to the right of the inequality is independent of y. This gives us

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

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

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

Thus, we conclude that

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

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 (1.28)

The difference in argument between deriving (1.27) and (1.28) stems from the fact that $e'(x_0) \neq 0$. Thus, we cannot simply replace e with e' in (1.19) and proceed as shown above to deduce (1.28). However, noting that, since $e(x_0) = e(x_1) = 0$, there exist by Rolle's theorem a point $\bar{x}$ in I such that $e'(\bar{x}) = 0$, we can instead of (1.19) start from

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

to show (1.28).

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 size of the interval h.

1.2.2 Continuous Piecewise Linear Interpolation

It is straight forward to extend the concept of interpolation to continuous piecewise linear functions. Given a continuous function f we define the continuous piecewise linear interpolant $\pi f \in V_h$ to f by

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

Figure 1.4 shows the continuous piecewise linear interpolant $\pi f(x)$ to $f(x) = x\sin(\pi x)$ on a uniform mesh with 6 nodes.

We have the following estimates for continuous piecewise linear interpolation.

Proposition 1.2. *The following interpolation estimates hold.*

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

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

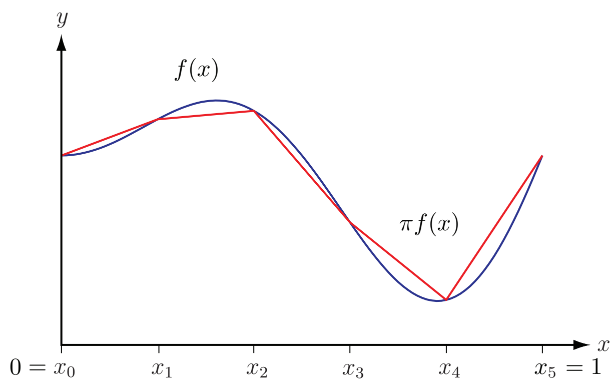


Fig. 1.4 A function f and its continuous piecewise linear interpolant πf on a mesh of $I = [0,1]$ with 6 nodes x_i, $i = 0, 1, \dots, 5$.

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 goes to zero. This is natural since we expect the interpolant πf to be a better approximation to f if the mesh is fine. The proposition also says that if the second derivative f'' of f is big then the interpolation error might be large. This is also natural since if the graph of f bends a lot (i.e., if f'' is big) then f is hard to approximate with a piecewise linear.

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 orthogonal-, or L^2-projection as a technique for approximating functions. L^2-projection gives a 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

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

In analogy with projection onto subspaces of $\mathbb{R}^n$, (1.36) 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. See Figure 1.5.

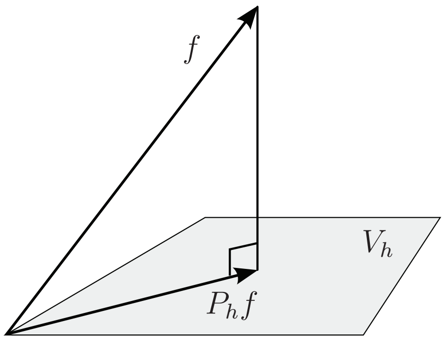


Fig. 1.5 Illustration of the orthogonal projection $P_h f$ of f onto the space V_h.

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 Figure 1.6 we show the L^2-projection of $f(x) = x\sin(\pi x)$ computed on the same mesh as was used for the interpolant πf shown in Figure 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.

1.3.2 Derivation of a Linear System of Equations

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

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

where φ_i, $i = 0, 1, \dots, n$, are the hat basis functions. This is a consequence of the fact that if (1.36) is satisfied for any choice of v as a hat function, then it is also satisfied for a linear combination of hat functions, and, conversely, since any function v in V_h is a linear combination of hat functions (1.36) implies (1.37).

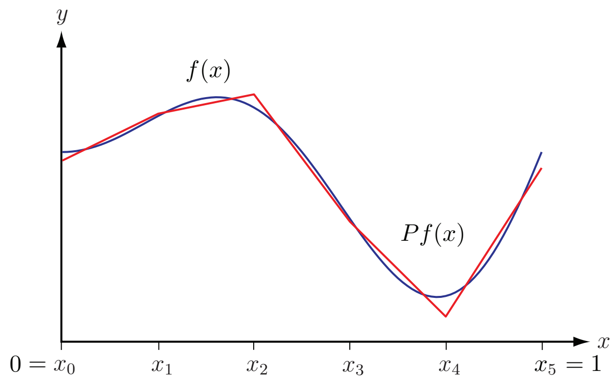


Fig. 1.6 The function $f = x\sin(\pi x)$ and its L^2-projection $P_h f$ on a mesh of $I = [0,1]$ with 6 nodes, x_i, $i = 1,2,\dots,6$.

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

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

with $n+1$ unknown coefficients ξ_j, $j = 0,1,\dots,n$, to be determined.

Inserting the ansatz (1.38) into the definition (1.36) we get

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

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

Further, introducing the notation

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

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

we have

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

which is an $(n+1)\times(n+1)$ linear system for the $n+1$ unknown coefficients ξ_j, $j = 0,1,\dots,n$. 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, $j = 0,1,\dots,n$ in the ansatz (1.38) satisfy a 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
       $M_{ij} = \int_I \varphi_j \varphi_i \, dx, \qquad b_i = \int_I f \varphi_i \, dx$      (1.45)
3: Solve the linear system
       $M\xi = b$      (1.46)
4: Set
       $P_h f = \sum_{j=0}^{n} \xi_j \varphi_j$      (1.47)
```

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 let f be a continuous function on the interval $I = [a,b]$, and consider the problem of evaluating approximately the integral

$$J = \int_I f(x)\, dx \tag{1.48}$$

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. We shall 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 of the integrand, respectively.

1.4.1 The Mid-point Rule

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

$$J \approx f(m)(b-a) \tag{1.49}$$

which is the Mid-point rule. Geometrically this means that we approximate the area under the integrand f with the area of the square $f(m)(b-a)$, see Figure 1.7.

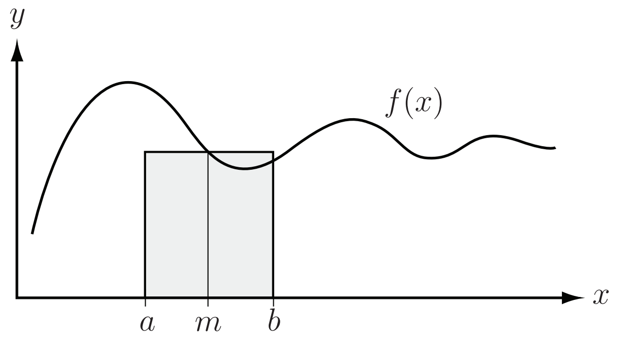


Fig. 1.7 The area of the shaded square approximates $\int_a^b f(x)\,dx$.

The Mid-point rule integrates linear polynomials exactly.

1.4.2 The Trapezoidal Rule

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

$$J \approx \frac{f(a)+f(b)}{2}(b-a) \tag{1.50}$$

which is the Trapezoidal rule. Geometrically this means that we approximate the area under f with the area under the trapezoidal domain formed by the points $(a,0)$, $(a, f(a))$, $(b,0)$, and $(b, f(b))$, see Figure 1.8. The Trapezoidal rule is also exact for linear polynomials.

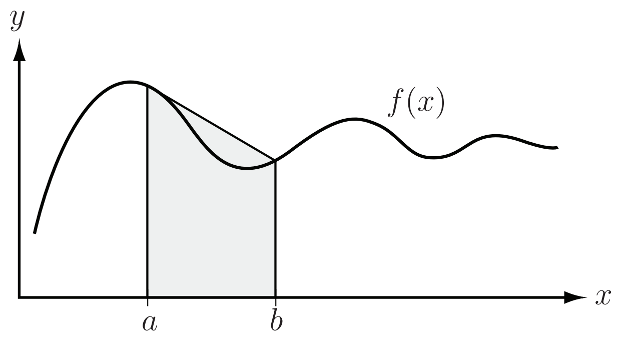


Fig. 1.8 The area of the shaded trapezoidal approximates $\int_a^b f(x)\,dx$.

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 $I = (0, 2h)$ 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 $(0, f(0))$, $(\frac{h}{2}, f(\frac{h}{2}))$, and $(h, f(h))$ (i.e., its graph passes trough these points) their coordinates must satisfy the equation for g. This gives the following linear system for c_0, c_1, and c_2.

$$\begin{bmatrix} 0 & 0 & 1 \\ \frac{1}{4}h^2 & \frac{1}{2}h & 1 \\ h^2 & h & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} f(0) \\ f(\frac{h}{2}) \\ f(h) \end{bmatrix} \tag{1.51}$$

Solving this one readily finds

$$c_0 = 2(f(0) - 2f(\tfrac{h}{2}) + f(h))/h^2, \quad c_1 = -(3f(0) - 4f(\tfrac{h}{2}) + f(h))/h, \quad c_2 = f(0) \tag{1.52}$$

Finally, integrating g from 0 to h one eventually ends up with

$$\int_0^h f\,dx \approx \int_0^h g(x)\,dx = \frac{f(0) + 4f(\frac{1}{2}h) + f(h)}{6} h \tag{1.53}$$

which is Simpson's formula.

On the interval $I = (a, b)$ Simpson's formula takes the form

$$J \approx \frac{f(a) + 4f(\frac{1}{2}(a+b)) + f(b)}{6}(b - a) \tag{1.54}$$

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 techniques let us now go through the somewhat intricate details of how to assemble the mass matrix M and load vector b. We begin by calculating the entries M_{ij} of the mass matrix. Recall that these involve products of hat functions. Since each hat is a linear polynomial the product of two hats is a quadratic polynomials. Thus, Simpson's formula can be used to integrate $M_{ij} = \int_I \varphi_i\varphi_j\,dx$ exactly. Moreover, 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 Figure 1.9 showing two neighbouring hat functions and their support. As a consequence, the mass matrix M is tridiagonal.

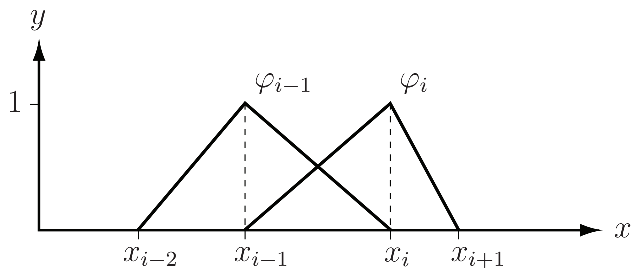


Fig. 1.9 Illustration of the hat functions φ_{i-1} and φ_i and their support.

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

$$\begin{aligned} M_{ii} &= \int_I \varphi_i^2\,dx && (1.55)\\ &= \int_{x_{i-1}}^{x_i} \varphi_i^2\,dx + \int_{x_i}^{x_{i+1}} \varphi_i^2\,dx && (1.56)\\ &= \frac{0+4\cdot(\frac{1}{2})^2+1}{6}h_i + \frac{1+4\cdot(\frac{1}{2})^2+0}{6}h_{i+1} && (1.57)\\ &= \frac{h_i}{3}+\frac{h_{i+1}}{3}, \quad i=1,2,\dots,n-1 && (1.58) \end{aligned}$$

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 $M_{i+1\,i}$ still using Simpson's formula we have

$$
\begin{aligned}
M_{i+1\,i} &= \int_I \varphi_i \varphi_{i+1}\, dx && (1.59)\\
&= \int_{x_i}^{x_{i+1}} \varphi_i \varphi_{i+1}\, dx && (1.60)\\
&= \frac{1\cdot 0 + 4(\frac{1}{2})^2 + 0\cdot 1}{6} h_{i+1} && (1.61)\\
&= \frac{h_{i+1}}{6}, \quad i = 0,1,\dots,n && (1.62)
\end{aligned}
$$

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

Hence, the mass matrix takes the form

$$
M = \begin{bmatrix}
\frac{h_1}{3} & \frac{h_1}{6} & & & & \\
\frac{h_1}{6} & \frac{h_1}{3}+\frac{h_2}{3} & \frac{h_2}{6} & & & \\
 & \frac{h_2}{6} & \frac{h_2}{3}+\frac{h_3}{3} & \frac{h_3}{6} & & \\
 & & \ddots & \ddots & \ddots & \\
 & & & \frac{h_{n-1}}{6} & \frac{h_{n-1}}{3}+\frac{h_n}{3} & \frac{h_n}{6} \\
 & & & & \frac{h_n}{6} & \frac{h_n}{3}
\end{bmatrix} \tag{1.63}
$$

The global mass matrix M can be written as a sum of n simpler matrices

$$
M = \begin{bmatrix}
\frac{h_1}{3} & \frac{h_1}{6} & & & \\
\frac{h_1}{6} & \frac{h_1}{3} & & & \\
 & & & & \\
 & & & & \\
 & & & &
\end{bmatrix}
+ \begin{bmatrix}
 & & & & \\
 & \frac{h_2}{3} & \frac{h_2}{6} & & \\
 & \frac{h_2}{6} & \frac{h_2}{3} & & \\
 & & & & \\
 & & & &
\end{bmatrix}
+ \dots +
\begin{bmatrix}
 & & & & \\
 & & & & \\
 & & & & \\
 & & & \frac{h_n}{3} & \frac{h_n}{6} \\
 & & & \frac{h_n}{6} & \frac{h_n}{3}
\end{bmatrix} \tag{1.64}
$$

$$
= M^{I_1} + M^{I_2} + \dots + M^{I_n} \tag{1.65}
$$

Each matrix M^{I_i}, $i = 1,2\dots,n$, is obtained by restricting the integration (1.41) 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 they are sparse and it suffice to compute the 2×2 blocks of non-zero entries. From the sum (1.65) we see that on each element I this small block takes the form

$$
M^I = \frac{1}{6}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} h \tag{1.66}
$$

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 $i = 1, 2, \dots, n$ do
3:     Compute the $2 \times 2$ local element mass matrix $M^I$ given by
           $$M^I = \frac{1}{6}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} h \qquad (1.67)$$
       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 = MassMat1D(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 consider the problem of calculating the load vector b. Because the entries $b_i = \int_I f\varphi_i \, dx$ depend on the function f we generally can not expect to calculate them exactly. However, we can approximate b_i using a quadrature rule. Using the Trapezoidal rule, for instance, we have

$$\begin{aligned}
b_i &= \int_I f\varphi_i \, dx && (1.68)\\
&= \int_{x_{i-1}}^{x_{i+1}} f\varphi_i \, dx && (1.69)\\
&= \int_{x_{i-1}}^{x_i} f\varphi_i \, dx + \int_{x_i}^{x_{i+1}} f\varphi_i \, dx && (1.70)\\
&\approx (f(x_{i-1})\varphi_i(x_{i-1}) + f(x_i)\varphi_i(x_i))h_i/2 && (1.71)\\
&\quad + (f(x_i)\varphi_i(x_i) + f(x_{i+1})\varphi_i(x_{i+1}))h_{i+1}/2 && (1.72)\\
&= (0 + f(x_i))h_i/2 + (f(x_i) + 0)h_{i+1}/2 && (1.73)\\
&= f(x_i)(h_i + h_{i+1})/2 && (1.74)
\end{aligned}$$

The approximate load vector then takes the form

$$b = \begin{bmatrix} f(x_0)h_1/2 \\ f(x_1)(h_1+h_2)/2 \\ f(x_2)(h_2+h_3)/2 \\ \vdots \\ f(x_{n-1})(h_{n-1}+h_n)/2 \\ f(x_n)h_n/2 \end{bmatrix} \tag{1.75}$$

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

$$\begin{aligned}
b &= \begin{bmatrix} f(x_0) \\ f(x_1) \\ \\ \\ \\ \end{bmatrix} h_1/2 + \begin{bmatrix} \\ f(x_1) \\ f(x_2) \\ \\ \\ \end{bmatrix} h_2/2 + \ldots + \begin{bmatrix} \\ \\ \\ f(x_{n-1}) \\ f(x_n) \end{bmatrix} h_n/2 && (1.76)\\
&= b^{I_1} + b^{I_2} + \ldots + b^{I_n}. && (1.77)
\end{aligned}$$

Each vector b^{I_i}, $i = 1, 2, \ldots, n$, is formally derived by restricting the integration (1.42) to element I_i. The assembly of the load vector is very similar to that of the mass matrix as the next algorithm shows.

Algorithm 3 Assembly of the Load Vector

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

$$b^I = \frac{1}{2}\begin{bmatrix} f(x_{i-1}) \\ f(x_i) \end{bmatrix} h \tag{1.78}$$

       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 = LoadVec1D(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 provides a way of passing a routine as argument to another routine. For example, suppose we have written a routine called `Foo` to specify the function $f(x) = x\sin(x)$

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

To assemble the corresponding load vector, we type

```
b = LoadVec1D(x,@Foo)
```

This passes the routine `Foo` as argument to `LoadVec1D` and allows it to be evaluated inside the assembler. The at sign `@` creates the function handle. Thus, function handles provide means for writing flexible and reusable code.

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 = MassMat1D(x) % assemble mass
b = LoadVec1D(x,@Foo) % 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 $\mathscr{P}_1(I)$ 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 $[0, 1]$ 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 subinterval I_i.
(d) What is the dimension of V_h?

Exercise 1.3. Determine the linear interpolant $\pi f \in \mathscr{P}_1(I)$ defined on the single interval $I = [0, 1]$ to the following functions f. Then make a plot of f and πf in the same figure.

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

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 $\pi f \in V_h$ for 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 if we are restricted to three subintervals?

Exercise 1.5. Let $I = [0, 1]$. Compute $\|f\|_\infty$ for $f = x(x - 1/2)(x - 1/3)$.

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

(a) Calculate $\int_I f\, dx$.
(b) Compute an approximation to $\int_I f\, dx$ using the Trapezoidal rule.
(c) Compute an approximation to $\int_I f\, dx$ using the Mid-point rule.
(d) Compute the errors in (b) and (c) and compare with theory.

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

(a) Calculate $\int_I f\,dx$.
(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 fomrula on each subinterval.
(d) Compute the errors in (b) and (d). 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 $\mathscr{P}_(I)$ of linear functions on I. Calculate 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. Calculate the L^2-projection $P_h f \in V_h$ of f.
(c) Plot your results and compare with the nodal interpolant $\pi_h f$.

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 $i = 0, 1, \dots, n$, where $\{\varphi_i\}_{i=0}^n \subset V_h$ is the usual basis of hat functions.

Exercise 1.10. Recall that $(f,g) = \int_I fg\,dx$ and $\|f\|_{L^2(I)}^2 = (f,f)$ are the L^2-scalar product and norm, respectively. 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 $\{v_1, \dots, v_m\}$ 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 $\{v_1, \dots, v_m\}$ is orthogonal?

Exercise 1.12. Show that $\{1, x, (3x^2 - 1)/2\}$ form a basis for the space of quadratic polynomials $\mathscr{P}_2(I)$, on $I = [-1,1]$. Then compute and draw the L^2-projections $P_h f \in \mathscr{P}_2(I)$ 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. Use the MATLAB code above to 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) $f(x) = \arctan((x-0.5)/\varepsilon)$, with $\varepsilon = 0.1$ and 0.01.

Test on a uniform mesh with $n = 5$, 25, and 100 subintervals.

Chapter 2
The Finite Element Method in 1D

Abstract In this chapter we shall introduce the finite element method as a general tool for the numerical solution of two-point boundary value problems. In doing so, the basic idea is to first rewrite the boundary value problem as a variational equation, and then seek a solution approximation from the space of continuous piecewise linears. This discretization procedure results in a linear system that can be solved on a computer. We then prove basic error estimates and show how to use them to formulate adaptive algorithms that can be used to automatically improve the accuracy of the computed solution. The derivation and areas of application of the studied boundary value problems are also discussed.

2.1 The Finite Element Method for a Model Problem

2.1.1 A Two-point Boundary Value Problem

Let us consider the following two-point boundary value problem: find u such that

$$-u'' = f, \quad x \in I = (0,1) \tag{2.1a}$$
$$u(0) = u(1) = 0 \tag{2.1b}$$

where f is a given function. Sometimes this problem is easy to solve analytically. For instance, if $f = 1$ then we readily find $u = x(1-x)/2$ by integrating f twice and using the boundary conditions $u(0) = u(1) = 0$. However, for a general f it may be difficult or even impossible to find u with analytical techniques. Thus, we see that even a very simple differential equation like this may be difficult to solve analytically. We take this as a good motivation for studying numerical techniques and, in particular, for introducing the finite element method.

2.1.2 Variational Formulation

The derivation of a finite element method always starts by rewriting the differential equation under consideration as variational equation. In our case this so-called variational formulation is obtained by multiplying $-u'' = f$ by a test function v, which is assumed to vanish at the end-points of the interval I, and integrate by parts. Doing so we have

$$\int_0^1 fv\,dx = -\int_0^1 u''v\,dx \tag{2.2}$$

$$= \int_0^1 u'v'\,dx - u'(1)v(1) + u'(0)v(0) \tag{2.3}$$

$$= \int_0^1 u'v'\,dx \tag{2.4}$$

where we have used the assumption $v(0) = v(1) = 0$. For this calculation to make sense we must assert that the test function v is not too badly behaved so that the involved integrals exist. To do so, we require that both v and v' be bounded on I. To this end we introduce the space

$$V_0 = \{v : \|v'\| < \infty,\ \|v\| < \infty,\ v(0) = v(1) = 0\} \tag{2.5}$$

which is the largest space imaginable for v. Obviously, this space contans many functions, which all can be used as test function. In fact there are infinitely many functions in V_0 and we therefore say that V_0 has infinite dimension. Further, since u is twice differentiable and satisfies the boundary conditions $u(0) = u(1) = 0$ it is easy to see that it too belongs to V_0. This leads to the following variational formulation of (2.2): find $u \in V_0$ such that

$$\int_0^1 u'v'\,dx = \int_0^1 fv\,dx, \quad \forall v \in V_0 \tag{2.6}$$

By analogy with the name test function for v, the solution u is sometimes called trial function.

2.1.3 Finite Element Approximation

We next try to approximate u by a continuous piecewise linear function. To this end we introduce a mesh on the interval I consisting of n subintervals, and the corresponding space V_h of all continuous piecewise linears. Since we are dealing with functions vanishing at the end-points of I, we also introduce the following subspace $V_{h,0}$ of V_h that satisfies the boundary conditions

$$V_{h,0} = \{v \in V_h : v(0) = v(1) = 0\} \tag{2.7}$$

In other words $V_{h,0}$ contains all piecewise linears which are zero at $x = 0$ and $x = 1$. In terms of hat basis functions this means that a basis for $V_{h,0}$ is obtained by deleting the half hats φ_0 and φ_n from the usual set $\{\varphi_j\}_{j=0}^n$ of hat functions spanning V_h.

Replacing the large space V_0 with the much smaller subspace $V_{h,0} \subset V_0$ of piecewise linears in the variational formulation (2.6) we obtain the following finite element method: find $u_h \in V_{h,0}$ such that

$$\int_0^1 u_h' v' \, dx = \int_0^1 f v \, dx, \quad \forall v \in V_{h,0} \tag{2.8}$$

We mention that this type of finite element method with similar trial and test space is sometimes called a Galerkin method, named after a famous russian matematican.

2.1.4 Derivation of a Linear System of Equations

To compute the finite element approximation u_h we first note that (2.8) is equivalent to

$$\int_0^1 u_h' \varphi_i' \, dx = \int_0^1 f \varphi_i \, dx, \quad i = 1, 2, \dots, n-1 \tag{2.9}$$

where, as said before, φ_i, $i = 1, 2, \dots, n-1$ are the hat functions spanning $V_{h,0}$. This is a consequence of the fact that if (2.9) is satisfied for all hat functions $\{\varphi_j\}_{j=1}^{n-1}$, then it is also satisfied for a linear combination of hats.

Now, since u_h belongs to $V_{h,0}$ we can write it as the linear combination

$$u_h = \sum_{j=1}^{n-1} \xi_j \varphi_j \tag{2.10}$$

with $n-1$ unknown coefficients ξ_j, $j = 1, 2 \dots, n-1$, to be determined.

Inserting the ansatz (2.10) into the finite element method (2.9) we get

$$\begin{aligned} \int_0^1 f \varphi_i \, dx &= \int_0^1 \left(\sum_{j=1}^{n-1} \xi_j \varphi_j' \right) \varphi_i' \, dx \\ &= \sum_{j=1}^{n-1} \xi_j \int_0^1 \varphi_j' \varphi_i' \, dx, \quad i = 1, 2, \dots, n-1 \end{aligned} \tag{2.11}$$

Further, introducing the notation

$$A_{ij} = \int_0^1 \varphi_j' \varphi_i' \, dx, \quad i, j = 1, 2, \dots, n-1 \tag{2.12}$$

$$b_i = \int_0^1 f \varphi_i \, dx, \quad i = 1, 2, \dots, n-1 \tag{2.13}$$

we have

$$b_i = \sum_{j=1}^{n-1} A_{ij}\xi_j, \quad i = 1,2,\dots,n-1 \tag{2.14}$$

which is an $(n-1)\times(n-1)$ linear system for the $n-1$ unknown coefficients ξ_j, $j = 1,2,\dots,n-1$. In matrix form we write this

$$b = A\xi \tag{2.15}$$

where the entries of the $(n-1)\times(n-1)$ matrix A and the $(n-1)\times 1$ vector b are defined by (2.12) and (2.13), respectively.

We thus conclude that the coefficients ξ_j, $j = 1,2,\dots,n-1$ in the ansatz (2.10) satisfy a linear system, which must be solved to obtain the finite element solution u_h.

We refer to A as the stiffness matrix and to b as the load vector.

2.1.5 Basic Algorithm to Compute the Finite Element Solution

The following algorithm summarizes the basic steps for computing the finite element solution u_h.

Algorithm 4 Basic Finite Element Algorithm

```
1: Create a mesh with $n$ elements on the interval $I$ and define the corresponding space of continuous piecewise linear functions $V_{h,0}$.
2: Compute the $(n-1)\times(n-1)$ matrix $A$ and the $(n-1)\times 1$ vector $b$, with entries
       $$A_{ij} = \int_I \varphi_j'\varphi_i'\,dx, \qquad b_i = \int_I f\varphi_i\,dx \tag{2.16}$$
3: Solve the linear system
       $$A\xi = b \tag{2.17}$$
4: Set
       $$u_h = \sum_{j=1}^{n-1} \xi_j\varphi_j \tag{2.18}$$
```

2.2 Basic A Priori Error Estimate

Since u_h only approximates u, estimates of the error $e = u - u_h$ are necessary to judge the quality and, consequently, the usability of u_h. The following theorem gives a key result for deriving such error estimates.

Theorem 2.1 (Galerkin orthogonality). *The finite element approximation u_h, defined by* (2.8)*, satisfies the orthogonality*

$$\int_0^1 (u-u_h)'v'\,dx = 0, \quad \forall v \in V_{h,0} \tag{2.19}$$

Proof. From the variational formulation we have

$$\int_0^1 u'v'\,dx = \int_0^1 fv\,dx \quad \forall v \in V_0 \tag{2.20}$$

and from the definition of the finite element method

$$\int_0^1 u_h'v'\,dx = \int_0^1 fv\,dx \quad \forall v \in V_{h,0} \tag{2.21}$$

Subtracting these and using the fact that $V_{h,0} \subset V_0$ immediately proves the claim.

The next theorem is a best approximation result.

Theorem 2.2. *The finite element solution u_h, defined by* (2.8) *satisfies*

$$\|(u-u_h)'\| \leq \|(u-v)'\|, \quad \forall v \in V_{h,0} \tag{2.22}$$

Proof. Writing $u-u_h = u-v+v-u_h$ for any $v \in V_{h,0}$ we have

$$\begin{align}
\|(u-u_h)'\|^2 &= \int_I (u-u_h)'(u-v+v-u_h)'\,dx \tag{2.23}\\
&= \int_I (u-u_h)'(u-v)'\,dx + \int_I (u-u_h)'(v-u_h)'\,dx \tag{2.24}\\
&= \int_I (u-u_h)'(u-v)'\,dx \tag{2.25}\\
&\leq \|(u-u_h)'\|\,\|(u-v)'\| \tag{2.26}
\end{align}$$

where we used the Galerkin orthogonality to conclude that

$$\int_I (u-u_h)'(v-u_h)'\,dx = 0 \tag{2.27}$$

since $v-u_h \in V_h$. Dividing by $(v-u_h)'$ concludes the proof.

There are two types of error estimates, namely, *a priori* error estimates and *a posteriori* error estimates. The difference between the two types is that a priori error estimates express the error in terms of the exact solution u, while a posteriori error estimates express the error in terms of the finite element approximation u_h. We shall now state and prove a basic a priori error estimate.

Theorem 2.3. *The finite element solution u_h, defined by* (2.8) *satisfies*

$$\|(u-u_h)'\|^2 \leq C\sum_{i=1}^{n} h_i^2 \|u''\|^2_{L^2(I_i)} \tag{2.28}$$

where C is a constant.

Proof. Starting from the best approximation result (2.22) and choosing $v = \pi u$, and using the interpolation estimate (1.16) the a priori error estimate immediately follows.

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

$$\|(u - u_h)'\| \le Ch\|u''\| \tag{2.29}$$

and thus the derivative of the error tends to zero as the maximum mesh size h tend to zero.

2.3 Mathematical Modeling

A fundamental tool for deriving the equations of applied mathematics and physics is the idea that some quantities can be tracked within a physical system. This idea is used to create some balance laws for the system and then to express these with equations. Common examples include conservation of mass, energy, and balance of momentum or force. To familiar ourselves with this way of thinking we shall now derive two differential equations governing heat transfer and the elastic deformation of a bar. As we shall see the modeling of both these physical phenomenons leads to the two-point boundary value problem (2.1). From this we make the observation that even though the underlying physics are very different, it is often so that the mathematical derivation and resulting partial differential equations are similar. Thus, many physical phenomena are described by the same partial differential equations, and therefore the methods and mathematical theory can often be developed for certain model problems and still be applied to a wide range of different applications.

2.3.1 Derivation of the Stationary Heat Equation

Consider a thin metal rod of length L and cross section area A [m^2] occupying the interval $[0, L]$. The rod is heated by a heat source (e.g., a small electrical current) of intensity f [J/(sm)], which has been acting for a long time so that the heat transfer process is at a steady state, and all physical quantities are independent of time. We want to find the distribution of temperature T [K] within the rod.

Let q [J/(sm^2)] be the heat flux along the direction of increasing x. The first law of thermodynamics, which expresses conservation of energy, states that the amount of heat produced by the heat source equals the flow of heat out of the rod. That is,

$$A(L)q(L) - A(0)q(0) = \int_0^L f\,dx \tag{2.30}$$

From the fundamental theorem of calculus we have

$$A(L)q(L) - A(0)q(0) = \int_0^L (Aq)'\, dx \tag{2.31}$$

which gives

$$\int_0^L (Aq)'\, dx = \int_0^L f\, dx \tag{2.32}$$

Energy conservation is a fundamental principle of nature, but it is not enough to yield a closed form differential equation for T. To this end we need some sort of empirical law deduced from experiments relating temperature T and heat flux q. Now, since heat flows from hot to cold regions, it is reasonable to assume that heat flux is proportional to the negative temperature gradient. This is neatly expressed by Fourier's law,

$$q = -kT' \tag{2.33}$$

where k [J/(Kms)] the thermal conductivity of the rod.

Combining (2.32) and (2.33) we have

$$\int_0^L ((AkT')' + f)\, dx = 0 \tag{2.34}$$

Letting $L \to 0$ we conclude that

$$-(AkT')' = f \tag{2.35}$$

which is the stationary Heat equation.

We note that this is a problem with variable coefficients since A, k, and f might vary.

2.3.2 Boundary Conditions for the Heat Equation

Generally there are many functions T which satisfies the Heat equation (2.35) for a given right hand side f. For example, if $A = k = 1$ and $f = 0$, then any linear function T will do as a solution. Thus, to obtain a unique solution it is necessary to impose some auxiliary constraints on the equation. These are called boundary conditions and specifies T at the end-points $x = 0$ and $x = L$ of the rod. There are essentially three types of boundary conditions, namely, Dirichlet, Neumann, and Robin boundary conditions, named after famous mathematicians.

2.3.2.1 Dirichlet Boundary Conditions

Dirichlet, or strong, boundary conditions prescribe the value of the solution at the boundary. For example $T(0) = 0$. From a physical point of view this corresponds to cooling the left end-point of the rod so that it is always kept at constant zero temperature.

2.3.2.2 Neumann Boundary Conditions

Neumann, or natural, boundary conditions prescribe the value of the solution derivative at the boundary. Since $T' = -q/k$ we see that this corresponds to prescribing the heat flux q at the boundary. In particular, $T'(0) = 0$ means that the left end-point of the rod is thermally isolated.

2.3.2.3 Robin Conditions

Robin boundary conditions is a mixture of Dirichlet and Neumann boundary conditions, typically $AkT'(0) = T(0) - T_\infty$. In real-world applications this is perhaps the most realistic boundary condition, since it means that the heat flux is proportional to the difference between the temperature of the rod and the ambient media T_∞.

Robin boundary conditions can be used to approximate boundary conditions of either Dirichlet or Neumann type. To see this consider the general Robin boundary condition

$$AkT'(0) = \kappa(T(0) - T_\infty) + q_\infty \tag{2.36}$$

where $\kappa \geq 0$, T_∞, and q_∞ are parameters to be chosen. Choosing $\kappa = 0$ we immediately obtain the Neumann boundary condition $AkT'(0) = q_\infty$. Choosing on the other hand κ large means that whenever $T(0) \neq T_\infty$ there will be a heat flux between the rod and the ambient media, which will counteract this difference. As a consequence, the Robin condition will approximate the Dirichlet condition $T(0) = T_\infty$ as κ tends to infinity.

2.3.3 Derivation of a Differential Equation for the Deformation of a Bar

A bar is a structure that is only subjected to axial loads. Consider a bar occupying the interval $[x_0, x_1]$ subjected to a line load f [N/m] and assume we wish to compute the resulting displacement u [m]. The equilibrium equation for the interval $[x_0, x_1]$ is

$$A(x_1)\sigma(x_1) - A(x_0)\sigma(x_0) + \int_{x_0}^{x_1} f\,dx = 0 \tag{2.37}$$

where A [m^2] is the area of the cross section of the bar and σ [N/m^2] is the stress and thus $A\sigma = F$ [N] is the force at any given point. Dividing (2.37) by $x_1 - x_0$ and letting $x_1 \to x_0$ we get the differential equation

$$-(A\sigma)' = f \tag{2.38}$$

Next, assuming we have a linear elastic material the relation between the stress and the deformation is given by Hooke's law

$$\sigma = E\varepsilon \tag{2.39}$$

where E is the elastic modulus and $\varepsilon = u'$ is the strain, with u the vertical displacement of the bar.

Combining (2.37) and (2.39) we arrive at the following second order differential equation

$$-(AEu')' = f \tag{2.40}$$

We note that this is also a problem with variable coefficients since A, E, and f might vary.

2.3.4 Boundary Conditions for the Bar

Similarly to Heat equation we now need to equip the bar equation with boundary conditions, which describe the bar at the boundary.

2.3.4.1 Dirichlet Boundary Conditions

These conditions take the form $u(0) = g_0$ and are used to model a given displacement g_0 at the endpoint. For example, if $g_0 = 0$ then the bar is clamped at $x = 0$.

2.3.4.2 Neumann Boundary Conditions

These conditions take the form $AEu'(0) = g_0$ and models the situation when a given force acts at the endpoint $x = 0$.

2.3.4.3 Robin Boundary Conditions

Finally, we recall that Robin boundary conditions is a mixture of Dirichlet and Neumann boundary conditions of the form, $AEu'(0) = k_0(u(0) - g_0)$, which models a situation where the force at $x = 0$ is proportional to the displacement at the end-point adjusted by g_0. We may think of a bar which at its end-point is connected to a spring with spring constant k such that $k(u(0)-g_0)$ is the force from the spring acting on the bar.

2.4 A Model Problem with Variable Coefficients and Robin Boundary Conditions

Considering the two real-world applications just presented we realize that it is desirable to be able to treat equations with variable coefficients and different types of boundary conditions. To this end let us consider a more general two-point value problem. More specific, we wish to find the solution u to

$$-(au')' = f, \quad x \in I = (0,1) \tag{2.41a}$$
$$au'(0) = \kappa_0(u(0)-g_0) \tag{2.41b}$$
$$-au'(1) = \kappa_1(u(1)-g_1) \tag{2.41c}$$

where $a > 0$ and f are given functions, and $\kappa_0 \geq 0$, $\kappa_1 \geq 0$, g_0, and g_1 are given parameters. The positiveness assumption on a, κ_0, and κ_1 is necessary to assert existence and uniqueness of the solution u. We do not dwell on this right now, but shall return to discuss the well-posedness of (2.41) later on.

2.4.1 Variational Formulation

Multiplying (2.41a) by a test function v and integrating by parts we have

$$\int_0^1 fv\,dx = \int_0^1 -(au')'v\,dx \tag{2.42}$$
$$= \int_0^1 au'v'\,dx + a(1)u'(1)v(1) + a(0)u'(0)v(0) \tag{2.43}$$
$$= \int_0^1 au'v'\,dx + \kappa_1(u(1)-g_1)v(1) + \kappa_0(u(0)-g_0)v(0) \tag{2.44}$$

where we used the boundary conditions to rewrite the boundary terms. Note that we do require v to satisfy any boundary conditions as this is only necessary for problems with Dirichlet boundary conditions. Consequently, the appropriate test and trial space is given by

$$V = \{v : \|v'\| < \infty,\ \|v\| < \infty\} \tag{2.45}$$

Collecting terms involving u on the left hand side, and terms involving given functions on the right hand side we obtain the following variational formulation of

(2.41): find $u \in V$ such that

$$\begin{aligned}&\int_0^1 au'v'\,dx + \kappa_1 u(1)v(1) + \kappa_0 u(0)v(0)\\ &= \int_0^1 fv\,dx + \kappa_1 g_1 v(1) + \kappa_0 g_0 v(0), \quad \forall v \in V\end{aligned} \tag{2.46}$$

2.4.2 *Finite Element Approximation*

Replacing the space V by the space of all continuous piecewise polynomials V_h in the variational formulation (2.46) we obtain the following finite element method: find $u_h \in V_h$ such that

$$\begin{aligned}&\int_0^1 au_h'v'\,dx + \kappa_1 u_h(1)v(1) + \kappa_0 u_h(0)v(0)\\ &= \int_0^1 fv\,dx + \kappa_1 g_1 v(1) + \kappa_0 g_0 v(0), \quad \forall v \in V_h\end{aligned} \tag{2.47}$$

We next show how to implement this finite element method in a computer.

2.5 Computer Implementation

In this section we describe the main components of a finite element solver. We do this by writing a computer code implementing the finite element method (2.47).

2.5.1 *Assembly of the Stiffness Matrix and Load Vector*

Inserting the ansatz

$$u_h = \sum_{j=0}^{n} \xi_j \varphi_j \tag{2.48}$$

into the finite element method (2.47) we eventually end up with the linear system

$$(A+R)\xi = b + r \tag{2.49}$$

where the entries of the $(n+1)\times(n+1)$ matrices A and R, and the $n+1$ vectors b and r are given by

$$A_{ij} = \int_0^1 a\varphi_j'\varphi_i'\,dx \tag{2.50}$$

$$R_{ij} = \kappa_1\varphi_j(1)\varphi_i(1) + \kappa_0\varphi_j(0)\varphi_i(0) \tag{2.51}$$

$$b_i = \int_0^1 f\varphi_i\,dx \tag{2.52}$$

$$r_i = \kappa_1 g_1\varphi_i(1) + \kappa_0 g_0\varphi_i(0) \tag{2.53}$$

To assemble A and b we recall that the explicit expression for a hat function φ_i is given by

$$\varphi_i = \begin{cases} (x - x_{i-1})/h_i, & \text{if } x \in I_i \\ (x_{i+1} - x)/h_{i+1}, & \text{if } x \in I_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{2.54}$$

Hence, the derivative φ_i' is either $1/h_i$, $-1/h_{i+1}$, or 0 depending on the subinterval.

Using (2.54) it is straight forward to calculate the entries of A. For $|i - j| > 1$, we have $A_{ij} = 0$, since φ_i and φ_j lack common support. However, when $i = j$, the support of φ_i and φ_j overlap and A_{ij} is potentially non-zero. Let us use mid-point quadrature to approximate A_{ij}. To this end let a_i be the value of a at the mid-point of I_i. When $i = j$ we have the diagonal entries

$$\begin{aligned} A_{ii} &= \int_0^1 a\varphi_i'^2\,dx && (2.55)\\ &= \int_{x_{i-1}}^{x_i} a\varphi_i'^2\,dx + \int_{x_i}^{x_{i+1}} a\varphi_i'^2\,dx && (2.56)\\ &\approx a_i\frac{1}{h_i^2}h_i + a_{i+1}\frac{(-1)^2}{h_{i+1}^2}h_{i+1} && (2.57)\\ &= \frac{a_i}{h_i} + \frac{a_{i+1}}{h_{i+1}}, \quad i = 1,2,\ldots,n-1 && (2.58) \end{aligned}$$

The integrals of the first and last diagonal entries are a_1/h_1 and a_n/h_n since φ_0 and φ_n are only half.

Further, when $j = i + 1$ we have the subdiagonal entries

$$\begin{aligned} A_{i\,i+1} &= \int_0^1 a\varphi_{i+1}'\varphi_i'\,dx && (2.59)\\ &= \int_{x_i}^{x_{i+1}} a\varphi_{i+1}'\varphi_i'\,dx && (2.60)\\ &\approx a_{i+1}\frac{(-1)}{h_{i+1}}\cdot\frac{1}{h_{i+1}}h_{i+1} && (2.61)\\ &= -\frac{a_{i+1}}{h_{i+1}}, \quad i = 0,1,\ldots,n && (2.62) \end{aligned}$$

The superdiagonal entries are obviously the same as the subdiagonal entries.

The entries $R_{ij} = \kappa_0 \varphi_j(0)\varphi_i(0) + \kappa_1 \varphi_j(1)\varphi_i(1)$ are all zero, except when $i = j = 0$ or $i = j = n$, in which case we have $R_{00} = \kappa_0$ and $R_{nn} = \kappa_1$.

Hence, the stiffness matrix $A + R$ takes the form

$$A+R = \begin{bmatrix} \frac{a_1}{h_1} & -\frac{a_1}{h_1} & & & & \\ -\frac{a_1}{h_1} & \frac{a_1}{h_1}+\frac{a_2}{h_2} & -\frac{a_2}{h_2} & & & \\ & -\frac{a_2}{h_2} & \frac{a_2}{h_2}+\frac{a_3}{h_3} & -\frac{a_3}{h_3} & & \\ & & \ddots & \ddots & \ddots & \\ & & & -\frac{a_{n-1}}{h_{n-1}} & \frac{a_{n-1}}{h_{n-1}}+\frac{a_n}{h_n} & -\frac{a_n}{h_n} \\ & & & & -\frac{a_n}{h_n} & \frac{a_n}{h_n} \end{bmatrix} + \begin{bmatrix} \kappa_0 & & \\ & & \\ & & \\ & & \kappa_1 \end{bmatrix} \tag{2.63}$$

The computation of the load vector $b + r$ is done exactly as shown for the L^2-projection, apart from the addition of the terms $r_1 = \kappa_0 g_0 \varphi_i(0)$ and $r_n = \kappa_1 g_1 \varphi_i(1)$ to the first and last vector entry. Hence, we have

$$b+r = \begin{bmatrix} f(x_0)h_1/2 \\ f(x_1)(h_1+h_2)/2 \\ f(x_2)(h_2+h_3)/2 \\ \vdots \\ f(x_{n-1})(h_{n-1}+h_n)/2 \\ f(x_n)h_n/2 \end{bmatrix} + \begin{bmatrix} \kappa_0 g_0 \\ \\ \\ \vdots \\ \\ \kappa_1 g_1 \end{bmatrix} \tag{2.64}$$

The global stiffness matrix $A + R$ can be split into a sum of global element stiffness matrices

$$A+R = \frac{a_1}{h_1}\begin{bmatrix} 1 & -1 & & \\ -1 & 1 & & \\ & & & \\ & & & \end{bmatrix} + \frac{a_2}{h_2}\begin{bmatrix} & & & \\ & 1 & -1 & \\ & -1 & 1 & \\ & & & \end{bmatrix} + \ldots + \frac{a_n}{h_n}\begin{bmatrix} & & & \\ & & & \\ & & 1 & -1 \\ & & -1 & 1 \end{bmatrix} \tag{2.65}$$

$$+ \begin{bmatrix} \kappa_0 & & \\ & & \\ & & \kappa_1 \end{bmatrix}$$

$$= A^{I_1} + A^{I_1} + \ldots + A^{I_n} + R \tag{2.66}$$

Each global element stiffness matrix A^{I_i}, $i = 1,2,\ldots,n$ is found by performing the integration (2.50) over a single element I_i. The following algorithm summarizes the assembly process of A.

Algorithm 5 Assembly of the Stiffness Matrix

```
 1: Allocate memory for the $(n+1)\times(n+1)$ matrix $A$ and initialize all matrix entries to zero.
 2: for $i = 1,2,\dots,n$ do
 3:     Compute the $2\times 2$ local element stiffness matrix $A^I$ given by
            $$A^I = \frac{a_i}{h}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \qquad (2.67)$$
        where $h$ is the length of element $I_i = [x_{i-1}, x-i]$, and $a_i = a((x_{i-1}+x_i)/2)$.
 4:     Add $A^I_{11}$ to $A_{ii}$.
 5:     Add $A^I_{12}$ to $A_{ii+1}$.
 6:     Add $A^I_{21}$ to $A_{i+1i}$.
 7:     Add $A^I_{22}$ to $A_{i+1i+1}$.
 8: end for
 9: Add $\kappa_0$ to $a_{00}$.
10: Add $\kappa_1$ to $a_{n+1n+1}$.
```

A MATLAB routine for assembling the stiffness matrix is listed below.

```
function A = StiffMat1D(x,a,kappa)
n = length(x)-1;
A = zeros(n+1,n+1);
for i = 1:n
  h = x(i+1) - x(i);
  xmid = (x(i+1) + x(i))/2; % interval mid-point
  amid = a(xmid); % value of a(x) at mid-point
  A(i,i) = A(i,i) + amid/h; % add amid/h to A(i,i)
  A(i,i+1) = A(i,i+1) - amid/h;
  A(i+1,i) = A(i+1,i) - amid/h;
  A(i+1,i+1) = A(i+1,i+1) + amid/h;
end
A(1,1) = A(1,1) + kappa(1);
A(n+1,n+1) = A(n+1,n+1) + kappa(2);
```

Input to this routine is a vector `x` holding node coordinates, a function handle `a` to a routine specifying the function a, and a vector `kappa` for the boundary condition parameters κ_0 and κ_1. Output is the assembled stiffness matrix $A+R$.

The load vector $b+g$ is computed in a similar manner by modifying the routine `LoadVec1D` as shown below.

```
function b = LoadVec1D(x,f,kappa,g)
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
```

```
b(1) = b(1) + kappa(1)*g(1);
b(n+1) = b(n+1) + kappa(2)*g(2);
```

The inputs `x`, `f`, and `kappa` are as before. The vector `g`, holds the boundary parameters g_0 and g_1. Output is the assembled load vector $b+g$.

2.5.2 A Finite Element Solver for a General Two-point Boundary Value Problem

With the above pieces of code it is easy to write a finite element solver for (2.41). For fun sake let us use it to compute the temperature T in a rod of length $L = 6$ m, cross section $A = 0.1$ m^2, thermal conductivity $k = 5 - 0.6x$ J/(Ksm), internal heat source $f = 0.03(x-6)^4$ J/sm, held at constant temperature $T = -1$ K at $x = 2$, and thermally insulated at $x = 8$. Thus, we want to solve

$$-(0.5+0.7x)T'' = 0.3x^2, \quad 2 < x < 8, \quad T(2) = -1, \quad T'(8) = 0 \qquad (2.68)$$

To approximate the Dirichlet condition $T(2) = 7$ we use the Robin condition (2.41b) with parameters $\kappa_0 = 10^6$ and $g_0 = -1$. Similarly, to impose the Neumann condition $T'(8) = 0$ we let $\kappa_1 = 0$ in (2.41c). The value of g_1 does not matter.

The main solver routine takes the following form.

```
function PoissonSolver1D()
h = 0.1; % mesh size
x = 2:h:8; % mesh
kappa = [1.e+6 0];
g = [-1 0];
A = StiffMat1D(x, @Conductivity, kappa);
b =  LoadVec1D(x, @Source, kappa, g);
U = A\b;
plot(x,U)
```

Here, the heat conductivity and source are specified by the following routines.

```
function y = Conductivity(x)
y = 0.1*(5 - 0.6*x); % heat conductivity times area

function y = Source(x)
y = 0.03*(x-6)^4; % heat source
```

Running this code we get the temperature distribution shown in Figure 2.1.

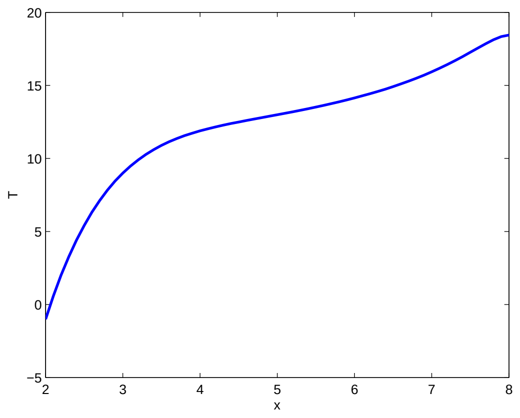


Fig. 2.1 Computed temperature on a uniform mesh with 25 elements.

2.6 Adaptive Finite Element Methods

Adaptive finite element methods uses information extracted from earlier computations to locally refine or modify the mesh in order to obtain a better solution approximation u_h. This information is obtained using a posteriori error estimates. The aim is to get u_h to be optimal in the sense that a desired level of accuracy is reached at a minimal computational cost.

2.6.1 A Posteriori Error Estimates

Let us return to consider the simple model problem (2.1). We have the following a posteriori error estimate for its finite element solution u_h.

Proposition 2.1. *The following estimate holds*

$$\|(u-u_h)'\|^2 \leq C\sum_{i=1}^{n} \rho_i^2(u_h) \tag{2.69}$$

where the element residual $R_i(u_h)$ is defined by

$$\rho_i(u_h) = h_i\|f+u_h''\|_{L^2(I_i)} \tag{2.70}$$

Note that for piecewise linear approximation $u_h'' = 0$ so that the the residual simplifies to

$$\rho_i(u_h) = h_i \|f\|_{L^2(I_i)} \tag{2.71}$$

Proof. Let $e = u - u_h$ be the error. We then have

$$\|e'\|^2 = \int_0^1 e'^2 \, dx \tag{2.72}$$
$$= \int_0^1 e'(e - \pi e)' \, dx \tag{2.73}$$
$$= \sum_{i=1}^n \int_{x_{i-1}}^{x_i} e'(e - \pi e)' \, dx \tag{2.74}$$
$$= \sum_{i=1}^n \int_{x_{i-1}}^{x_i} (-e'')(e - \pi e) \, dx + \left[e'(e - \pi e)\right]_{x_{i-1}}^{x_i} \tag{2.75}$$
$$= \sum_{i=1}^n \int_{x_{i-1}}^{x_i} (-e'')(e - \pi e) \, dx \tag{2.76}$$

where we first have used the Galerkin orthogonality property (2.19) to subtract an interpolant $\pi e \in V_h$ to e, then integration by parts on each element, and finally that e and πe coincide at the nodes to get rid of the boundary terms. Here, we note that on element I_i

$$-e'' = -(u - u_h)'' = -u'' + u_h'' = f + u_h'' \tag{2.77}$$

Using now the Cauchy-Schwartz inequality and a standard interpolation error estimate we have

$$\|e'\|^2 = \sum_{i=1}^n \int_{x_{i-1}}^{x_i} (f + u_h'')(e - \pi e) \tag{2.78}$$
$$\leq \sum_{i=1}^n \|f + u_h''\|_{L^2(I_i)} \|e - \pi e\|_{L^2(I_i)} \tag{2.79}$$
$$\leq \sum_{i=1}^n \|f + u_h''\|_{L^2(I_i)} C h_i \|e'\|_{L^2(I_i)} \tag{2.80}$$
$$= C \sum_{i=1}^n h_i \|f + u_h''\|_{L^2(I_i)} \|e'\|_{L^2(I_i)} \tag{2.81}$$
$$\leq C \left(\sum_{i=1}^n h_i^2 \|f + u_h''\|_{L^2(I_i)}^2 \right)^{1/2} \left(\sum_{i=1}^n \|e'\|_{L^2(I_i)}^2 \right)^{1/2} \tag{2.82}$$
$$= C \left(\sum_{i=1}^n h_i^2 \|f + u_h''\|_{L^2(I_i)}^2 \right)^{1/2} \|e'\|_{L^2(I)} \tag{2.83}$$

Dividing both sides by $\|e'\|_{L^2(I)}$ concludes the proof.

2.6.2 Adaptive Mesh Refinement

From Proposition 2.1 we see that the error gradient e' is bounded by the local mesh size h_i, and the element residual $f + u_h''$. This is natural, since we expect to get a small error on a fine mesh and also if the equation is well satisfied by u_h. Recall that if u_h was the exact solution u, then $f + u_h'' = 0$. Thus, the element residual R_i is proportional to the error on element I_i. To increase the accuracy of the finite element solution u_h it is therefore tempting to selectively split the elements with the largest element residuals into smaller ones, since this will decrease h_i and (hopefully) also $\rho_i(u_h)$. In doing so, one strives to obtain a uniform distribution of the error among the elements. This reasoning leads us to the following algorithm for designing adaptive, or smart, finite element methods with automatic error control based on a posteriori estimates in combination with local mesh refinement.

Algorithm 6 Algorithm for A Posteriori Based Adaptive Mesh Refinement

```
1: Given a (coarse) mesh with $n$ nodes.
2: **while** $n$ is not too large **do**
3:     Compute the finite element approximation $u_h$.
4:     Evaluate the element residuals $\rho_i$, $i = 1, 2, \dots, n$.
5:     Select and refine the the most error prone elements.
6: **end while**
```

The adaptive algorithm above consists of four main components:

1. Computation of the element residuals ρ_i.
2. Selection of elements to be refined.
3. A refinement procedure.
4. A stopping criterion.

Let us discuss the computer implementation of these four steps.

In practice, we calculate the element residuals ρ_i using quadrature. It is convenient to store them in a vector, `rho`.

```
rho = zeros(n,1); % allocate element residuals
for i = 1:n % loop over elements
  h = x(i+1) - x(i); % element length
  a = f(x(i)); % temporary variables
  b = f(x(i+1));
  t = (a^2+b^2)*h/2; % integrate f^2. Trapezoidal rule
  rho(i) = h^2*t; % element residual
end
```

As usual `x` is a vector of node coordinates and `n` is the number of elements.

There are different possibilities for selecting the elements to be refined given the element residuals ρ_i. A popular method is to refine element i if

$$\rho_i > \alpha \max_{i=1,2,\dots,n} \rho_i, \tag{2.84}$$

where $0 \leq \alpha \leq 1$ is a parameter to be chosen. Note that $\alpha = 0$ gives a uniform refinement, while $\alpha = 1$ gives no refinement at all.

The refinement procedure consists of the insertion of a new node at the mid-point of each element chosen for refinement. In other words, if we are refining element $I_i = [x_i, x_{i+1}]$, then we replace it by $[x_i, (x_i + x_{i+1})/2] \cup [(x_i + x_{i+1})/2, x_{i+1}]$. This is easily implemented by looping over the elements and inserting the mid-point coordinate of any element with a too large residual at the end of the vector `x` holding all node coordinates, and then sort the vector.

```
alpha = 0.9 % refinement parameter
for i = 1:length(rho)
  if rho(i) > alpha*max(rho) % if large residual
    x = [x (x(i+1)+x(i))/2]; % insert new node point
  end
end
x = sort(x); % sort node points accendingly
```

The stopping criterion determines when the adaptive algorithm should stop. It can, for instance, take the form of a maximum bound on the number of nodes or elements, the memory usage, the time of the computation, the total size of the residual, or a combination of these.

Adaptive mesh refinement is particularly useful for problems with solutions containing high localized gradients, such as shocks or kinks, for instance. One such problem is $-u'' = \delta$, $0 < x < 1$, $u(0) = u(1) = 0$, where delta is the narrow pulse $\delta = \exp(-c|x-0.5|^2)$, with $c = 100$. The solution to this problem looks like a single triangle wave with its peak at $x = 0.5$. In Figure 2.2 we show the computed solution u_h to this problem after 25 mesh refinement loops starting from a coarse mesh with 5 nodes distributed more or less randomly over the computational domain. Clearly, the adaptive algorithm has identified and resolved the difficult region with high gradients near the peak of the triangle wave. This allows for high accuracy while at the same time saving computational resources.

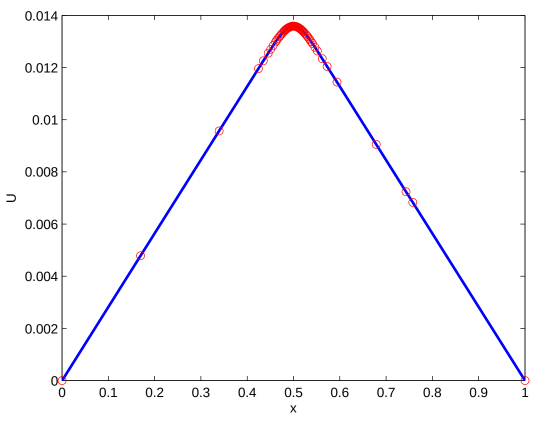


Fig. 2.2 Adaptively computed solution u_h. Each red ring symbolize a node.

2.7 Problems

Exercise 2.1. Solve the model problem (2.1) analytically with

(a) $f(x) = 1$.
(b) $f(x) = x - u$.

Exercise 2.2. 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 $[0,1]$ into three subintervals. Furthermore, let $V_{h,0}$ be the space of continuous piecewise linear functions on this partition that vanish at the end-points $x = 0$ and $x = 1$.

(a) Compute the stiffness matrix A defined by (2.12).
(b) Compute the load vector with $f(x) = 1$ defined by (2.13).
(c) Solve the linear system $A\xi = b$ and compute the finite element solution u_h. Plot u_h.

Exercise 2.3. Consider the problem

$$-u'' = 7, \quad x \in (0,1)$$
$$u(0) = 2, \; u(1) = 3$$

(a) What is a suitable finite element space V_h?
(b) Formulate a finite element method for this problem.
(c) Derive the discrete system of equations using a uniform mesh with 4 nodes.

Exercise 2.4. Consider the problem

$$-((1+x)u')' = 0, \quad x \in (0,1)$$
$$u(0) = 0, \; u'(1) = 1$$

Divide the interval $(0,1)$ into 3 subintervals of equal length $h = 1/3$ and let V_h be the corresponding space of continuous piecewise linear functions vanishing at $x = 0$.

(a) Determine the analytical solution u.
(b) Use V_h to formulate a finite element method.
(c) Verify that the stiffness matrix A and load vector b are given by

$$A = \frac{1}{2}\begin{bmatrix} 16 & -9 & 0 \\ -9 & 20 & -11 \\ 0 & -11 & 11 \end{bmatrix}, \qquad b = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$$

(d) Verify that A is positive definite.

Exercise 2.5. Compute the stiffness matrix to the Neumann problem

$$-u'' = f, \quad x \in (0,1)$$
$$u'(0) = u'(1) = 0$$

on a uniform partition of $(0,1)$ into 2 subintervals. Why is the corresponding stiffness matrix singular?

Exercise 2.6. Consider the problem

$$-u'' + u = f, \quad x \in (0,1)$$
$$u(0) = u(1) = 0$$

(a) Choose a suitable finite element space V_h.
(b) Formulate a finite element method.
(c) Derive the discrete system of equations.

Exercise 2.7. Let u be defined on $I = (0,1)$ and such that $u(0) = 0$. Prove the Poincaré inequality

$$\|u\|_{L^2(I)} \leq C\|u'\|_{L^2(I)}$$

Exercise 2.8. Derive an a posteriori error estimate for the problem

$$-u'' + u = f, \quad x \in I$$
$$u(0) = u(1) = 0$$

Exercise 2.9. Consider the problem

$$-\varepsilon u'' + xu' + u = f, \quad x \in I$$
$$u(0) = u'(1) = 0$$

where $\varepsilon > 0$ is a constant. Prove that the solution satisfies

$$\|\varepsilon u''\|_{L^2(I)} \leq \|f\|_{L^2(I)}$$

Exercise 2.10. Consider the model problem

$$\begin{aligned} -u'' &= f, \quad x \in I \\ u(0) &= u(1) = 0 \end{aligned}$$

Its variational formulation reads: find $u \in V_0$ such that

$$\int_I u'v'\,dx = \int_I fv\,dx \quad \forall v \in V_0$$

Show that the solution $u \in V_0$ to the variational formulation minimizes the functional

$$F(w) = \frac{1}{2}\int_I w'^2\,dx - \int_I fw\,dx$$

over the space V_0. *Hint:* Write $w = u + v$ and show that $F(w) = F(u) + \ldots \geq F(u)$.

Chapter 3
Piecewise Polynomial Approximation in 2D

Abstract In this chapter we extend the concept of piecewise polynomial approximation to two dimensions. As before the basic idea is to first construct spaces of piecewise polynomial functions that are easy to manipulate (e.g., differentiate and integrate), and then to show that one can approximate more complicated functions by these simple polynomials. A difficulty with the construction of piecewise polynomials in higher dimension is that the underlying domain must be partitioned into simplex, such as triangles, or quadrilaterals, for instance, which is a non-trivial task for a complex shaped domain. In this context a very important principle is that the smaller the simplex, the better the representation of the domain as well as the approximation properties of the resulting function spaces. The price we have to pay is higher computational costs and increasing memory requirements. However, we shall present a technology for building representations of piecewise polynomials that is efficient and suitable for computer implementation.

3.1 Meshes

3.1.1 Triangulations

Let $\Omega \subset \mathbb{R}^2$ be a simply connected domain with polygonal boundary $\partial\Omega$. A triangulation, or mesh, $\mathscr{K}$ of Ω is a set $\{K\}$ of triangles K such that $\Omega = \cup_{K\in\mathscr{K}} K$, and such that the intersection of two triangles is either an edge, a corner, or empty. No triangle corner is allowed to lie on an edge of another triangle. The corners of the triangles are called the nodes. Figure 3.1 shows a triangle mesh of the greek letter π.

To measure the size of a triangle K we introduce the local mesh size h_K, defined as the length of the longest edge on K, see Figure 3.5. Moreover, to measure the quality of K, let d_K be the diameter of the inscribed circle and introduce the chunkiness parameter α_K, defined by

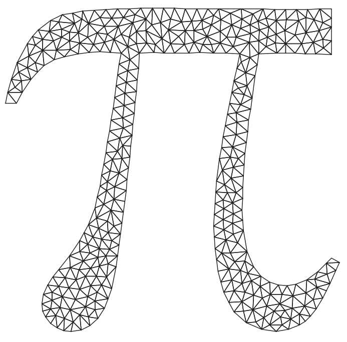


Fig. 3.1 A mesh of π.

$$\alpha_K = h_K / d_K \tag{3.1}$$

We say that a triangulation $\mathscr{K}$ is shape regular if there is a constant $\alpha_0 > 0$ such that

$$\alpha_K \geq \alpha_0, \quad \forall K \in \mathscr{K} \tag{3.2}$$

This condition means that the shape of the triangles can not be too bad in the sense that the angles of any triangle can neither be very wide nor very narrow. As we shall see this has implications for the approximation properties of the piecewise polynomial spaces do be defined on these meshes.

3.1.2 Data Storage Structures

The standard way of representing a triangulation with n_p nodes and n_t elements in a computer is to store it as two matrices P and T called the point matrix, and the connectivity matrix, respectively. The point matrix P is $2 \times n_p$ and column j contains the coordinates $x_1^{(j)}$ and $x_2^{(j)}$ of node N_j. The connectivity matrix T is $3 \times n_t$ and column j contains the numbers of the three nodes in triangle j. Here, we shall adopt the common convention of ordering these three nodes in a counter clockwise sense. It does not, however, matter on which of the nodes the ordering starts.

Figure 3.2 shows a small triangulation of an L-shape domain. The mesh has eight nodes and six triangles. The point matrix and connectivity matrix for this mesh are given by

$$P = \begin{bmatrix} 0.0 & 1.0 & 2.0 & 0.0 & 1.0 & 2.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 1.0 & 1.0 & 2.0 & 2.0 \end{bmatrix}, \qquad T = \begin{bmatrix} 1 & 2 & 5 & 3 & 4 & 5 \\ 2 & 5 & 2 & 6 & 5 & 8 \\ 4 & 4 & 8 & 5 & 7 & 7 \end{bmatrix} \tag{3.3}$$

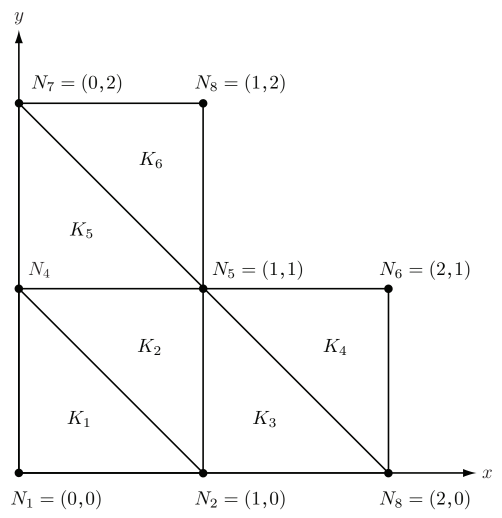


Fig. 3.2 A triangle mesh of the L-shaped domain.

Thus, for example, the coordinates $(x_1^{(3)}, x_2^{(3)}) = (2,0)$ of node N_3 are given by the matrix entries p_{13} and p_{23}, respectively. In the connectivity matrix T column 2 contains the numbers 2, 5, and 4 of the three nodes N_2, N_5, and N_4 making up triangle K_2. Note that the nodes are ordered counter clockwise.

3.1.3 Mesh Generation

Over the past decades advanced computer algorithms for the automatic construction of meshes have been developed. However, depending on the complexity of the domain it may still be more or less difficult to generate a mesh. In particular, difficulties may arise for three dimensional geometries, since they have often have a difficult topology. However, in two dimensions there are efficient algorithms for creating a mesh on quite general domains. One of these is the Delaunay algorithm, which given a set of points can determine a triangulation with the given points as triangle nodes. Delaunay triangulations are optimal in the sense that the angles of all triangles are maximal.

MATLAB has a non-standard set of routines called the PDE toolbox which includes a Delaunay mesh generator for creating high quality triangulations of two dimensional geometries. We illustrate its use by creating a mesh of the L-shaped domain.

In MATLAB the geometry of the L-shaped domain is defined by the following matrix `g`

```
g = [2 0 2 0 0 1 0;
```

```
       2 2 2 0 1 1 0;
       2 2 1 1 1 1 0;
       2 1 1 1 2 1 0;
       2 1 0 2 2 1 0;
       2 0 0 2 0 1 0]';
```

Each column of g describes one of the six line segments making up the boundary of the L-shaped domain. In each such column rows two and three contain the starting x_1-coordinate, and rows four and five the corresponding x_2-coordinate. Rows six and seven indicate if the geometry is on the left or right side of the line segment when traversing it in the direction induced by the start and end-points. The fact that we are defining a line segment is indicated by the number 2 in the first column.

To generate a mesh of the domain we type

```
[p,e,t] = initmesh(g,'hmax',0.1)
```

The call to initmesh routine invokes the mesh generator which triangulates the domain g. The final two arguments 'hmax',0.1 specifies the maximum edge length $h_K = 0.1$ of the triangles to be generated. Output is the point matrix p, the connectivity matrix t, and the so-called edge matrix e containing the node numbers of the triangle edges making up the boundary of the mesh. We will return to discuss the e matrix later on.

There are also a few built-in geometries, including:

- cicrcleg, the unit radius circle centered at origo.
- squareg, the square $[-1,1]^2$.

For furture use we extend this list of geometries with a rectangle, defined by

```
function r = Rectg(xmin,ymin,xmax,ymax)
r=[2 xmin xmax ymin ymin 1 0;
   2 xmax xmax ymin ymax 1 0;
   2 xmax xmin ymax ymax 1 0;
   2 xmin xmin ymax ymin 1 0]';
```

To view the generated mesh one can type

```
pdemesh(p,e,t)
```

More general geometries can be drawn in the PDE toolbox GUI. It is opened by typing

```
pdetool
```

3.2 Piecewise Polynomial Spaces

The reason for introducing a mesh of a domain is that it allows for a simple construction of piecewise polynomial function spaces on this domain, which is otherwise a very difficult task. We shall now discuss how this is done in the special case of linear polynomials on triangle meshes.

3.2.1 The Space of Linear Polynomials

Let K be a triangle and let $\mathscr{P}_1(K)$ be the space of linear functions on K, defined by

$$\mathscr{P}_1(K) = \{v : v = c_0 + c_1x_1 + c_2x_2,\ (x_1,x_2) \in K,\ c_0,c_1,c_2 \in \mathbb{R}\} \tag{3.4}$$

In other words $\mathscr{P}_1(K)$ contains all functions of the form $v = c_0 + c_1x_1 + c_2x_2$ on K.

We observe that any v in $\mathscr{P}_1(K)$ is uniquely determined by its nodal values $\alpha_i = v(N_i)$, $i = 1,2,3$. This follows by assuming α_i to be given and evaluating v at the three nodes $N_i = (x_1^{(i)}, x_2^{(i)})$. In doing so, we end up with the following linear system

$$\begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \tag{3.5}$$

Computing the determinant of the matrix we find that its absolute value equals $2|K|$, where $|K|$ is the area of K, so the linear system has a unique solution as long as K is not degenerate.

The natural basis $\{1,x_1,x_2\}$ for $\mathscr{P}_1(K)$ is not suitable since we wish to use the nodal values as degrees of freedom. Therefore we introduce a nodal basis $\{\lambda_1,\lambda_2,\lambda_3\}$, defined by

$$\lambda_j(N_i) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}, \quad i,j = 1,2,3 \tag{3.6}$$

Using the new basis we can express any function v in $\mathscr{P}_1(K)$ as

$$v = \alpha_1\lambda_1 + \alpha_2\lambda_2 + \alpha_3\lambda_3 \tag{3.7}$$

where $\alpha_i = v(N_i)$.

On the reference triangle $\bar{K}$ with nodes at origo, $(1,0)$, and $(0,1)$, the nodal basis functions for $\mathscr{P}_1(\bar{K})$ are given by

$$\lambda_1 = 1 - x_1 - x_2, \quad \lambda_2 = x_1, \quad \lambda_3 = x_2 \tag{3.8}$$

3.2.2 The Space of Continuous Piecewise Linear Polynomials

The contruction of piecewise linear functions on a mesh $\mathscr{K} = \{K\}$ of a domain Ω is straight forward. On each triangle K any such function v is simply required to belong to $\mathscr{P}_1(K)$. Requiring also continuity of v between neighbouring triangels, we obtain the space of all continuous piecewise linear polynomials V_h, defined by

$$V_h = \{v : v \in \mathscr{C}(\Omega),\ v|_K \in \mathscr{P}_1(K)\ \forall K \in \mathscr{K}\} \tag{3.9}$$

Here, $\mathscr{C}^0(\Omega)$ denotes the space of all continuous functions on Ω.

An example of a continuous piecewise linear function is given in Figure 3.3.

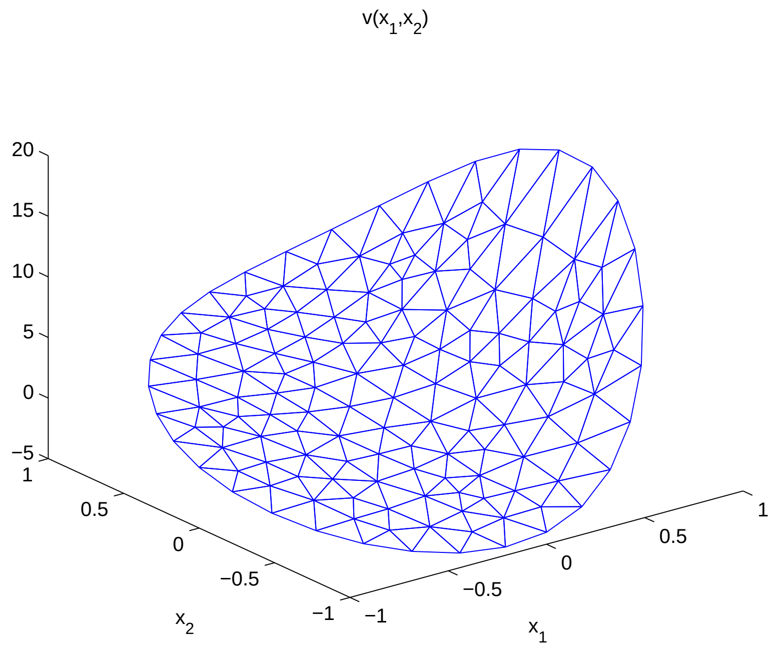


Fig. 3.3 A continuous piecewise linear function $v \in V_h$.

To construct a basis for V_h we first show that a function v in V_h is uniquely determined by its nodal values

$$\{v(N_j)\}_{j=1}^{n_p} \tag{3.10}$$

and, conversely, that for each set of nodal values there is a unique function v in V_h with these nodal values. To prove this claim we first note that the nodal values determines a function in $\mathscr{P}_1(K)$ uniquely for each $K \in \mathscr{K}$, and thus a function in V_h is uniquely determined by its values in the nodes. Next we consider two triangles K_1 and K_2 that share an edge $E = K_1 \cap K_2$. Let v_1 and v_2 be the two unique linear polynomials in $\mathscr{P}_1(K_1)$ and $\mathscr{P}_1(K_2)$, respectively, determined by the nodal values on K_1 and K_2. Since v_1 and v_2 are linear polynomials on K_1 and K_2 they are also linear polynomials when restricted to the edge E, and since they coincide in the endpoints of E we conclude that $v_1 = v_2$ on E. Therefore, for any set of nodal values there is a continuous piecewise linear polynomial with these nodal values.

Motivated by this result we let the nodal values be our degrees of freedom and define a corresponding basis $\{\varphi_j\}_{j=1}^{n_p} \subset V_h$ such that

$$\varphi_j(N_i) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}, \quad i,j = 1,2,\dots,n_p \tag{3.11}$$

Figure 3.4 illustrates a typical basis function φ_j.

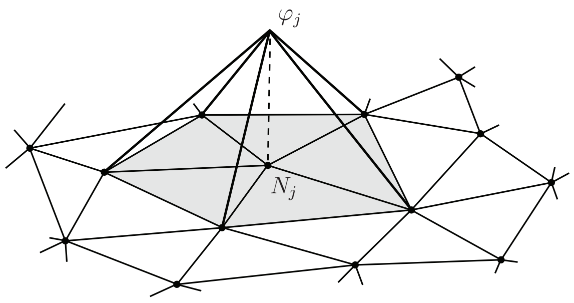


Fig. 3.4 A two-dimensional hat function φ_j on a general triangle mesh.

From the figure it is clear that each basis function φ_j is continuous, piecewise linear, and with support only on the small set of triangles sharing node N_j. Similar to the one-dimensional case, these basis functions are also called hat functions.

Now, using the hat function basis we note that any function v in V_h can be written

$$v = \sum_{i=1}^{n_p} \alpha_i \varphi_i \tag{3.12}$$

where $\alpha_i = v(N_i)$, $i = 1,2,\dots,n_p$, are the nodal values of v.

3.3 Interpolation

3.3.1 Linear Interpolation

We now return to the problem of approximating functions. Given a continuous function f on a triangle K with nodes N_i, $i = 1,2,3$, the linear interpolant $\pi f \in \mathscr{P}_1(K)$ to f is defined by

$$\pi f = \sum_{i=1}^{3} f(N_i)\varphi_i \tag{3.13}$$

The interpolant $\pi f \in \mathscr{P}_1(K)$ is a plane, which coincides with f at the three node points. Thus, by definion we have N_i $\pi f(N_i) = f(N_i)$. See Figure 3.5.

To estimate the interpolation error $f - \pi f$ we need to introduce some measure of the size of the first and second order derivatives of f. More precisely, let Df and

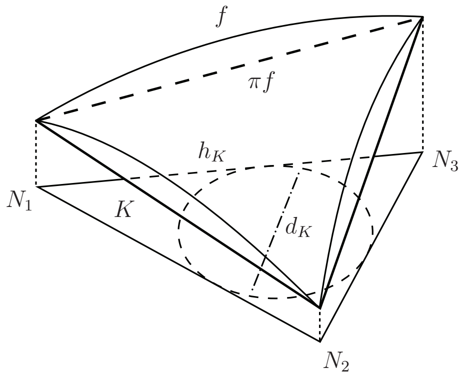


Fig. 3.5 Illustration of the linear interpolant πf to a function f on a triangle K with nodes N_1, N_2, and N_3. Also shown is the longest edge length h_K, and the diameter d_K of the inscribed circle.

$D^2 f$ be defined by

$$Df = \left(\left| \frac{\partial f}{\partial x} \right|^2 + \left| \frac{\partial f}{\partial y} \right|^2 \right)^{1/2}, \qquad D^2 f = \left(\left| \frac{\partial^2 f}{\partial x^2} \right|^2 + 2 \left| \frac{\partial^2 f}{\partial x \partial y} \right|^2 + \left| \frac{\partial^2 f}{\partial y^2} \right|^2 \right)^{1/2} \tag{3.14}$$

Since the operators D and D^2 include all first and second partial derivatives, we say that Df and $D^2 f$ are the total first and second derivative operators of f, respectively.

In this context we also recall that the $L^2(\Omega)$-norm of a function f of two variables x_1 and x_2 is given by.

$$\|f\| = \|f\|_{L^2(\Omega)} = \left(\int_\Omega f^2 \, dx \right)^{1/2} \tag{3.15}$$

Using these notations we have the following estimate of the interpolation error.

Proposition 3.1. *The following interpolation estimates hold.*

$$\|f - \pi f\|_{L^2(K)} \leq C h_K^2 \|D^2 f\|_{L^2(K)} \tag{3.16}$$

$$\|D(f - \pi f)\|_{L^2(K)} \leq C h_K \|D^2 f\|_{L^2(K)} \tag{3.17}$$

where C is a constant.

We omit the proof of this result. It is a consequence of Taylor's formula.

In Proposition 3.1, it is possible to show that the interpolation costants C are proportional to the inverse of $\sin(\theta_K)$, where θ_K is the smallest angle in triangle K. This, C blows up if θ_K is becomes small, which renders the interpolation error estimate uselss. This explains why it is critical that K has neither too narrow nor too wide angles. Recall that we measure this by the chunkiness parameter α_K, which should be bound away from zero.

3.3.2 Continuous Piecewise Linear Interpolation

The concept of continuous piecewise linear interpolation easily extends from one to two dimensions. Indeed, given a continuous function $f \in \mathscr{C}^0(\Omega)$ we define its continuous piecewise linear interpolant $\pi f \in V_h$ by

$$\pi f = \sum_{i=1}^{n_p} f(N_i)\varphi_i \tag{3.18}$$

Again, πf approximates f by taking on the same values in the nodes N_i.

In MATLAB it is easy to draw πf given f. For example, to plot the interpolant to $f = x_1 x_2$ on the square domain $\Omega = [-1,1]^2$ it takes only the following four lines of code.

```
[p,e,t] = initmesh('squareg','hmax',0.1); % mesh
x = p(1,:); y = p(2,:); % node coordinates
pif = x.*y; % nodal values of interpolant
pdesurf(p,t,pif) % plot interpolant
```

Looking at the above code let us make a remark about out programming style. The conversion of methematical symbols to computer code is not always clear and easy. In this book we have tried to keep a close correlation between the notation introduced in the formulas and equations, and the names of the variables used in the codes presented. However, attempting to write as efficient and short code as much as possible has unavoidable lead to a few inconsistencies in this respect. For example, to limit the number of indices used we have troughout used the variables `x` and `y` to denote the space coordinates x_1 and x_2. We hope that the code comments and the context shall make it clear what is meant.

The size of the interpolation error $f - \pi f$ can be estimated with the help of the following proposition.

Proposition 3.2. *The following interpolation estimates hold.*

$$\|f - \pi f\|^2_{L^2(\Omega)} \leq C \sum_{K \in \mathscr{K}} h_K^4 \|D^2 f\|^2_{L^2(K)} \tag{3.19}$$

$$\|D(f - \pi f)\|^2_{L^2(\Omega)} \leq C \sum_{K \in \mathscr{K}} h_K^2 \|D^2 f\|^2_{L^2(K)} \tag{3.20}$$

Proof. Using the triangle inequality followed by Proposition 3.1 we have

$$\|f - \pi f\|^2_{L^2(\Omega)} = \sum_{K \in \mathscr{K}} \|f - \pi f\|^2_{L^2(K)} \tag{3.21}$$

$$\leq \sum_{K \in \mathscr{K}} C h_K^4 \|D^2 f\|^2_{L^2(K)} \tag{3.22}$$

which proves the first estimate. The second follows similarly.

3.4 L^2-projection

3.4.1 Definition

The L^2-projection $P_h f \in V_h$ of a function of two variables $f \in L^2(\Omega)$ is defined by

$$\int_\Omega (f - P_h f) v \, dx = 0, \quad \forall v \in V_h \tag{3.23}$$

3.4.2 Derivation of a Linear System of Equations

To compute the L^2-projection $P_h f$ we first note that the definition (3.23) is equivalent to

$$\int_\Omega (f - P_h f) \varphi_i \, dx = 0 \quad i = 1, 2 \dots, n_p \tag{3.24}$$

where φ_i are the hat basis functions spanning V_h.

Since $P_h f$ belongs to V_h it can be written as the linear combination

$$P_h f = \sum_{j=1}^{n_p} \xi_j \varphi_j \tag{3.25}$$

where ξ_j, $j = 1, 2, \dots, n_p$, are n_p unknown coefficients to be determined.

Inserting the ansatz (3.25) into (3.25) we get

$$\int_\Omega f \varphi_i \, dx = \int_\Omega \left(\sum_{j=1}^{n_p} \xi_j \varphi_j \right) \varphi_i \, dx \tag{3.26}$$

$$= \sum_{j=1}^{n_p} \xi_j \int_\Omega \varphi_j \varphi_i \, dx \tag{3.27}$$

Using the notation

$$M_{ij} = \int_\Omega \varphi_j \varphi_i \, dx, \quad i, j = 1, 2, \dots, n_p \tag{3.28}$$

$$b_i = \int_\Omega f \varphi_i \, dx, \quad i = 1, 2 \dots, n_p \tag{3.29}$$

we have

$$b_i = \sum_{j=1}^{n_p} M_{ij} \xi_j, \quad i = 1, 2 \dots, n_p \tag{3.30}$$

which is a linear system for the unknowns ξ_j. In matrix form we write this

$$b = M\xi \tag{3.31}$$

where the entries of the $n_p \times n_p$ mass matrix M and the $n_p \times 1$ load vector b are defined by (3.28) and (3.29), respectively. Solving the linear system (3.30) we obtain the unknowns ξ_j, and thus $P_h f$.

3.4.3 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 7 Basic Algorithm to Compute the L^2-projection

```
1: Create a mesh $\mathscr{K}$ of $\Omega$ and define the corresponding space of continuous piecewise linear
   functions $V_h$ with hat function basis $\{\varphi_i\}_{i=1}^{n_p}$.
2: Assemble the $n_p \times n_p$ mass matrix $M$, and the $n_p \times 1$ load vector $b$, with entries
   $$M_{ij} = \int_\Omega \varphi_j \varphi_i \, dx, \quad b_i = \int_\Omega f \varphi_i \, dx \tag{3.32}$$
3: Solve the linear system
   $$M\xi = b \tag{3.33}$$
4: Set
   $$P_h f = \sum_{j=1}^{n_p} \xi_j \varphi_j \tag{3.34}$$
```

3.4.4 Existence and Uniqueness of the L^2-projection

Theorem 3.1. *The L^2-projection $P_h f$ of $f \in L^2(\Omega)$ defined by* (3.23) *exists and is unique.*

Proof. We first show that the L^2-projection is uniquely determined by (3.23). The argument is by contradiction. Assume that there are two L^2-projections $P_h f$ and $\widetilde{P_h f}$ satifying (3.23). Then we have

$$\int_\Omega P_h f v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V_h \tag{3.35}$$

$$\int_\Omega \widetilde{P_h f} v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V_h \tag{3.36}$$

Subtracting these equations we get

$$\int_\Omega (P_h f - \widetilde{P_h f}) v \, dx = 0, \quad \forall v \in V_h \tag{3.37}$$

Now, choosing $v = P_h f - \widetilde{P_h f} \in V_h$ we get

$$\int_\Omega |P_h f - \widetilde{P_h f}|^2 \, dx = 0 \tag{3.38}$$

From this identity we conclude that $P_h f - \widetilde{P_h f}$ must be zero.

To prove the existence of $P_h f$ we recall that is is determined by a $n_p \times n_p$ linear system of equations. The existence of a solution to a square system of linear equations follows from the uniqueness of the solution.

3.4.5 A Priori Error Estimates

Theorem 3.2. *The L^2-projection $P_h f$, defined by* (3.23) *satisfies the following best approximation estimate.*

$$\|f - P_h f\| \leq \|f - v\|, \quad \forall v \in V_h \tag{3.39}$$

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

$$\begin{aligned}
\|f - P_h f\|^2 &= \int_\Omega (f - P_h f)(f - v + v - P_h f) \, dx && (3.40)\\
&= \int_\Omega (f - P_h f)(f - v) \, dx + \int_\Omega (f - P_h f)(v - P_h f) \, dx && (3.41)\\
&= \int_\Omega (f - P_h f)(f - v) \, dx && (3.42)\\
&\leq \|f - P_h f\| \, \|f - v\| && (3.43)
\end{aligned}$$

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

$$\int_\Omega (f - P_h f)(v - P_h f) \, dx = 0 \tag{3.44}$$

since $v - P_h f \in V_h$. Dividing by $\|f - P_h f\|$ concludes the proof.

Theorem 3.3. *If f has square integrable second derivatives then its L^2-projection $P_h f$ satisfies*

$$\|f - P_h f\|^2 \leq C \sum_{K \in \mathscr{K}} C h_K^4 \|D^2 f\|^2_{L^2(K)} \tag{3.45}$$

Proof. Starting from the best approximation result and choosing $v = \pi f$ the interpolant of f, and using the interpolation error estimate of Proposition 3.1 we obtain

$$\|f - P_h f\|^2 \leq \|f - \pi f\|^2 \tag{3.46}$$
$$\leq \sum_{K \in \mathscr{K}} \|f - \pi f\|^2_{L^2(K)} \tag{3.47}$$
$$\leq \sum_{K \in \mathscr{K}} C h_K^4 \|D^2 f\|^2_{L^2(K)} \tag{3.48}$$

which proves the estimate.

Defining $h = \max_{K \in \mathscr{K}} h_K$ we conclude that

$$\|f - P_h f\| \leq C h^2 \|D^2 f\| \tag{3.49}$$

that is, the L^2-error $\|f - P_h f\|$ tends to zero when the mesh size h tends to zero.

3.4.6 Properties of the Mass matrix

Theorem 3.4. *The mass matrix is symmetric and positive definite.*

Proof. M is obviously symmetric since $M_{ij} = M_{ji}$ by definition.

To prove that M is positive definite we must show that

$$0 \leq \xi^T M \xi \tag{3.50}$$

for all $n_p \times 1$ vectors ξ and with equality only if $\xi = 0$.

Now, a straight forward calculation reveals that

$$\xi^T M \xi = \sum_{i,j=1}^{n_p} \xi_i M_{ij} \xi_j \tag{3.51}$$
$$= \sum_{i,j=1}^{n_p} \xi_i \left(\int_\Omega \varphi_j \varphi_i \, dx \right) \xi_j \tag{3.52}$$
$$= \int_\Omega \left(\sum_{i=1}^{n_p} \xi_i \varphi_i \right) \left(\sum_{j=1}^{n_p} \xi_j \varphi_j \right) dx \tag{3.53}$$
$$= \left\| \sum_{i=1}^{n_p} \xi_i \varphi_i \right\|^2_{L^2(\Omega)} \tag{3.54}$$

The last norm is equal to zero if and only if the sum $s = \sum_{i=1}^{n_p} \xi_i \varphi_i = 0$. However, the sum s can be viewed as a function in V_h and if $s = 0$ then all coefficients ξ_i mush vanish.

3.5 Quadrature and Numerical Integration

Quadrature in two dimensions works in principle the same as in one dimension. One approximates the integral with a sum of weights times integrand values. A general quadrature rule on a triangle K takes the form

$$\int_K f\,dx \approx \sum_j \omega_j f(q_j) \tag{3.55}$$

where $\{q_j\}$ is the set of quadrature points in K, and $\{\omega_j\}$ the corresponding quadrature weights. Below we list a few useful quadrature formulas for integrating a continuous function f over a general triangle K with nodes N_1, N_2 and N_3.

The simplest quadrature formula is the center of gravity rule

$$\int_K f\,dx \approx f(\bar{x})|K| \tag{3.56}$$

where $\bar{x} = (N_1 + N_2 + N_3)/3$ is the center of gravity and $|K|$ is the area of K. The center of gravity formula is a two dimensional variant of the mid-point rule. There is also a two dimensional analog to the trapezoidal rule, namely, the so-called corner quadrature formula

$$\int_K f\,dx \approx \sum_{i=1}^{3} f(N_i)\frac{|K|}{3} \tag{3.57}$$

A better quadrature formula is the two-dimensional mid-point rule

$$\int_K f\,dx \approx \sum_{1\leq i<j\leq 3}^{3} f(x^{ij})\frac{|K|}{3} \tag{3.58}$$

where $x^{ij} = (N_i + N_j)/2$ is the mid-point of the edge between node number i and j.

As you can imagine there are numerous other quadrature rules. We refer the interested reader to any standard text book on numerical analysis for a thorough description of this subject.

3.6 Computer Implementation

3.6.1 Assembly of the Mass Matrix

We next show how to compute the mass matrix M in two dimensions. This is a quite bit more complicated than in one dimension and we therfore do this by example. To this end, consider the mesh of the rectangle Ω in Figure 3.6.

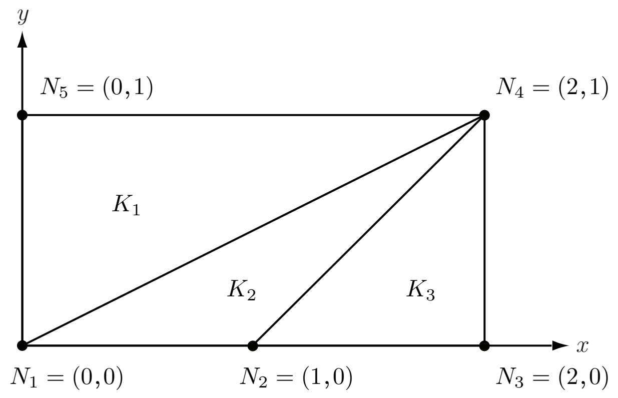


Fig. 3.6 A mesh of the rectangle $\Omega = [0,2] \times [0,1]$.

On this mesh we wish to compute the mass matrix M, given by

$$M = \int_\Omega \begin{bmatrix} \varphi_1\varphi_1 & \varphi_2\varphi_1 & \varphi_3\varphi_1 & \varphi_4\varphi_1 & \varphi_5\varphi_1 \\ \varphi_1\varphi_2 & \varphi_2\varphi_2 & \varphi_3\varphi_2 & \varphi_4\varphi_2 & \varphi_5\varphi_2 \\ \varphi_1\varphi_3 & \varphi_2\varphi_3 & \varphi_3\varphi_3 & \varphi_4\varphi_3 & \varphi_5\varphi_3 \\ \varphi_1\varphi_4 & \varphi_2\varphi_4 & \varphi_3\varphi_4 & \varphi_4\varphi_4 & \varphi_5\varphi_4 \\ \varphi_1\varphi_5 & \varphi_2\varphi_5 & \varphi_3\varphi_5 & \varphi_4\varphi_5 & \varphi_5\varphi_5 \end{bmatrix} dx \tag{3.59}$$

To do so, we first break the integral over the whole domain Ω into a sum of integrals over the triangles K_i, $i = 1,2,3$. We then have

$$M = \sum_{i=1}^{3} \int_{K_i} \begin{bmatrix} \varphi_1\varphi_1 & \varphi_2\varphi_1 & \varphi_3\varphi_1 & \varphi_4\varphi_1 & \varphi_5\varphi_1 \\ \varphi_1\varphi_2 & \varphi_2\varphi_2 & \varphi_3\varphi_2 & \varphi_4\varphi_2 & \varphi_5\varphi_2 \\ \varphi_1\varphi_3 & \varphi_2\varphi_3 & \varphi_3\varphi_3 & \varphi_4\varphi_3 & \varphi_5\varphi_3 \\ \varphi_1\varphi_4 & \varphi_2\varphi_4 & \varphi_3\varphi_4 & \varphi_4\varphi_4 & \varphi_5\varphi_4 \\ \varphi_1\varphi_5 & \varphi_2\varphi_5 & \varphi_3\varphi_5 & \varphi_4\varphi_5 & \varphi_5\varphi_5 \end{bmatrix} dx = \sum_{i=1}^{3} M^{K_i} \tag{3.60}$$

As we know there are only three non-zero hat functions on each triangle. For example, the only non-zero hats on K_1 are φ_1, φ_4, and φ_5. Integrating the product of these we see that K_1, or any triangle for that matter, gives a total of $3 \cdot 3 = 9$ integral contributions to M. Moreover, for a given triangle, the index on the non-zero hat functions are the same as the node numbers for that triangle. Thus, inspecting which hats are non-zero on which triangle, we can therefore beforehand say which rows and columns vanich in each matrix M^{K_i}. For example, the only non-zero entries of M^{K_1} are $M^{K_1}_{11}$, $M^{K_1}_{14}$, $M^{K_1}_{15}$, $M^{K_1}_{41}$, $M^{K_1}_{44}$, $M^{K_1}_{45}$, $M^{K_1}_{51}$, $M^{K_1}_{54}$, and $M^{K_1}_{55}$. Proceeding similarly, we are left with

$$M = \int_{K_1} \begin{bmatrix} \varphi_1\varphi_1 & 0 & 0 & \varphi_4\varphi_1 & \varphi_5\varphi_1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \varphi_1\varphi_4 & 0 & 0 & \varphi_4\varphi_4 & \varphi_5\varphi_4 \\ \varphi_1\varphi_5 & 0 & 0 & \varphi_4\varphi_5 & \varphi_5\varphi_5 \end{bmatrix} dx \tag{3.61}$$

$$+ \int_{K_1} \begin{bmatrix} \varphi_1\varphi_1 & \varphi_2\varphi_1 & 0 & \varphi_4\varphi_1 & 0 \\ \varphi_1\varphi_2 & \varphi_2\varphi_2 & 0 & \varphi_4\varphi_2 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \varphi_1\varphi_4 & \varphi_2\varphi_4 & 0 & \varphi_4\varphi_4 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} dx \tag{3.62}$$

$$+ \int_{K_1} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \varphi_2\varphi_2 & \varphi_3\varphi_2 & \varphi_4\varphi_2 & 0 \\ 0 & \varphi_2\varphi_3 & \varphi_3\varphi_3 & \varphi_4\varphi_3 & 0 \\ 0 & \varphi_2\varphi_4 & \varphi_3\varphi_4 & \varphi_4\varphi_4 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} dx \tag{3.63}$$

$$= M^{K_1} + M^{K_2} + M^{K_3} \tag{3.64}$$

In practice the global element matrices M^{K_i} are never formed, but only the small 3×3 element matrix necessary for storing their non-zero entries.

Having reduced the computation of the mass matrix M to a series of operations on the triangles, we consider a single triangle K with its three nodes N_1, N_2, and N_3, and corresponding hat functions φ_1, φ_2, and φ_3. These nodes will almost certainly have a different node numbering, say N_r, N_s, and N_t, in the mesh as a whole, but let us label them 1, 2, and 3 for now.

The computation of the element masses could of course be done using quadrature. However, there is a much easier way. Using induction it is possible to show the following integration formula for hat functions.

$$\int_K \varphi_1^m \varphi_2^n \varphi_3^p \, dx = \frac{2m!n!p!}{(m+n+p+2)!} |K| \tag{3.65}$$

where $|K|$ is the area of K and m, n, and p are positive integers. From this we immidiately have

$$\int_K \varphi_i \varphi_j \, dx = \frac{1}{12}(1 + \delta_{i,j})|K| \quad i,j = 1,2,3 \tag{3.66}$$

where $\delta_{i,j}$ is 1 if $i = j$ and 0 if $i \neq j$. This gives the following local element mass matrix

$$M^K = \frac{1}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} |K| \tag{3.67}$$

The mapping $\{1,2,3\} \mapsto \{r,s,t\}$ between the global node numbers r, s, and t and the local node numbers 1, 2, and 3 is called the local to global mapping. It is used when adding the entries of the local element mass matrix M^K into their appropriate positions in the global mass matrix M. This is done by cycling the index i and j over 1, 2 and 3 while and adding $M^K_{i,j}$ to $M_{\text{loc2glb}_i,\text{loc2glb}_j}$, where $\text{loc2glb} = [r,s,t]$. This gives a simple yet flexible way of organizing the assembly of the mass matrix. We summarize this assembly technique below.

Algorithm 8 Assembly of the Mass Matrix

```
1:  Let $n_p$ be the number of nodes and $n_t$ the number of elements in a mesh described by its point
    matrix $P$ and connectivity matrix $T$.
2:  Allocate memory for the $n_p \times n_p$ matrix $M$ and initialize all matrix entries to zero.
3:  for $K = 1,2,\dots,n_t$ do
4:    Compute the $3 \times 3$ local element mass matrix $M^K$ given by
          $$M^K = \frac{1}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} |K| \quad (3.68)$$
5:    Set up the local to global mapping, $\text{loc2glb} = [r,s,t]$.
6:    for $i = 1,2,3$ do
7:      for $j = 1,2,3$ do
8:
          $$M_{\text{loc2glb}_i,\text{loc2glb}_j} = M_{\text{loc2glb}_i,\text{loc2glb}_j} + M^K_{i,j} \quad (3.69)$$
9:      end for
10:   end for
11: end for
```

The conversion of this algorithm into MATLAB code is straight forward.

```
function M = MassMat2D(p,t)
np = size(p,2); % number of nodes
nt = size(t,2); % number of elements
M = sparse(n,n); % allocate mass matrix
for K = 1:nt % loop over elements
  loc2glb = t(1:3,K); % local-to-global map
  x = p(1,loc2glb); % node x-coordinates
  y = p(2,loc2glb); %      y
  area = polyarea(x,y); % triangle area
  MK = [2 1 1;
        1 2 1;
        1 1 2]/12*area; % element mass matrix
  M(loc2glb,loc2glb) = M(loc2glb,loc2glb) ...
    + MK; % add element masses to M
end
```

Input to this routine is the point matrix `p` and connectivity matrix `t` given by `initmesh`. Output is the assembled global mass matrix M. Note that the allocation of the mass matrix is done using the `sparse` command, which tells MATLAB to store only non-zero matrix entries. This is important in order to save memory, since the number of nodes and consequently the matrix size might be big.

Running this function on our mesh of the rectangle, which has the point and connectivity matrix

```
p=[0 1 2 2 0;
   0 0 0 1 1]
t=[1 1 2;
   4 2 3;
   5 4 4];
```

we get the 5×5 global mass matrix

```
M =

    0.2500    0.0417         0    0.1250    0.0833
    0.0417    0.1667    0.0417    0.0833         0
         0    0.0417    0.0833    0.0417         0
    0.1250    0.0833    0.0417    0.3333    0.0833
    0.0833         0         0    0.0833    0.1667
```

3.6.2 Assembly of the Load Vector

The load vector b is assembled using the same technique as the mass matrix M, that is, by summing element load vectors b^K over the mesh. On each element K we get a local 3×1 element load vector b^K with entries

$$b_i^K = \int_K f \varphi_i \, dx, \quad i = 1,2,3 \tag{3.70}$$

Using node quadrature, for instance, to compute these integrals we end up with

$$b_i^K \approx \frac{1}{3} f(N_i)|K|, \quad i = 1,2,3 \tag{3.71}$$

We summarize the computation of the load vector with the following algorithm.

Algorithm 9 Assembly of the Load Vector

```
1: Let $n_p$ be the number of nodes and $n_t$ the number of elements in a mesh described by its point
   matrix $P$ and connectivity matrix $T$.
2: Allocate memory for the $n_p \times 1$ vector $b$ and initialize all entries to zero.
3: for $K = 1, 2, \ldots, n_t$ do
4:     Compute the $3 \times 1$ local element load vector $b^K$ given by
           $$b^K = \frac{1}{3} \begin{bmatrix} f(N_1) \\ f(N_2) \\ f(N_3) \end{bmatrix} |K| \qquad (3.72)$$
5:     Set up the local to global mapping, loc2glb $= [r, s, t]$.
6:     for $i = 1, 2, 3$ do
7:         $$b_{\text{loc2glb}_i} = b_{\text{loc2glb}_i} + b_i^K \qquad (3.73)$$
8:     end for
9: end for
```

Translated into MATLAB code the algorithm takes the following form.

```
function b = LoadVec2D(p,t,f)
np = size(p,2);
nt = size(t,2);
b = zeros(n,1);
for K = 1:nt
  loc2glb = t(1:3,K);
  x = p(1,loc2glb);
  y = p(2,loc2glb);
  area = polyarea(x,y);
  bK = [f(x(1),y(1));
        f(x(2),y(2));
        f(x(3),y(3))]/3*area; % element load vector
  b(loc2glb) = b(loc2glb) ...
    + bK; % add element loads to b
end
```

Here, we assume that `f` is a function handle to a routine specifying f, for example,

```
function f = Foo(x,y)
f = x.*y;
```

A main routine for computing the L^2-projection πf of $f = x_1 x_2$ on the unit square $\Omega = [0,1]^2$ is given below.

```
function L2Projector2D()
g = Rectg(0,0,1,1); % unit square
[p,e,t] = initmesh(g,'hmax',0.1); % create mesh
M = MassMat2D(p,t); % assemble mass matrix
b = LoadVec2D(p,t,@Foo); % assemble load vector
```

```
Pf = M\b; % solve linear system
pdesurf(p,t,Pf) % plot projection
```

3.7 Problems

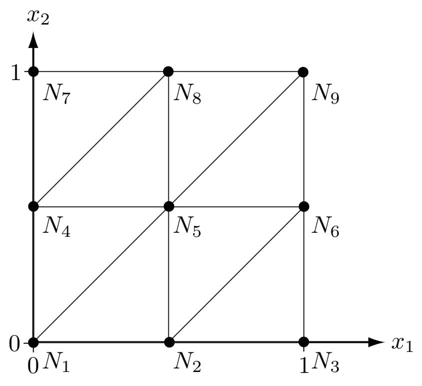
Exercise 3.1. Write down the geometry matrix `geom` for the unit square $\Omega = [0,1]^2$.

Exercise 3.2. Express the area of an arbitrary triangle in terms of its corner coordinates $(x_1^{(1)}, x_2^{(1)})$, $(x_1^{(2)}, x_2^{(2)})$, and $(x_1^{(3)}, x_2^{(3)})$.

Exercise 3.3. Derive the basis functions for piecewise linear functions on the triangle with corners at $(-1,-1)$, $(1,0)$, and $(-1,1)$.

Exercise 3.4. Determine the basis functions for piecewise linear functions on an arbitrary triangle with corner coordinates (x_1,y_1), (x_2,y_2) and (x_3,y_3).

Exercise 3.5. Determine a linear coordinate transform which maps an arbitrary triangle onto the reference triangle $\bar{K}$ with corners at origo, $(1,0)$, and $(0,1)$.



Exercise 3.6. Given the triangulation of Figure 3.7.

(a) Write down the point matrix P and the connectivity matrix T.
(b) Determine the mesh function $h(x)$.

Exercise 3.7. Looking at Figure 3.7, and draw the hat functions φ_1 and φ_5 corresponding to nodes N_1 and N_5, respectively.

Exercise 3.8. Consider again the mesh of the unit square Ω shown in Figure 3.7.

(a) Determine the sparsity pattern of the mass matrix on this mesh.

(b) Compute the integrals $\int_\Omega \phi_1\phi_2\,dx$, $\int_\Omega \phi_7\phi_4\,dx$, $\int_\Omega \phi_7\phi_8\,dx$, and $\int_\Omega x_1\phi_1\,dx$.

Exercise 3.9. Let $f = x_1x_2$ and let $\Omega = [0,1]^2$ be the unitsquare.

(a) Calculate $\int_\Omega f\,dx$ analytically.
(b) Compute $\int_\Omega f\,dx$ by using the center of gravity rule on each triangle of the mesh in Figure 3.7.
(c) Compute $\int_\Omega f\,dx$ by using the corner quadrature rule on each triangle of the mesh in Figure 3.7.

Exercise 3.10. Compute the L^2-projection $P_hf \in V_h$ to $f = x_1^2$ on the mesh shown in Figure 3.7. Use the corner qudrature rule to evaluate the integrals of the mass matrix and the load vector.

Chapter 4
The Finite Element Method in 2D

Abstract In this chapter we develop finite element methods for numerically solving partial differential equations in two dimensions. The approach taken is the same as before, that is, we first rewrite the equation in variational form, and then seek an approximate solution in the space of continuous piecewise linear functions. Although the numerical methods presented are general in nature, we shall focus on linear second order elliptic partial differential equations. The Poisson equation will be our model problem. We prove basic error estimates, discuss the implementation of the involved algorithms, and study some relevant examples of applications.

4.1 Green's Formula

At the outset let us recall a few mathematical preliminaries, which will be of frequent use later on.

Let Ω be a domain in $\mathbb{R}^2$, with boundary $\partial\Omega$ and exterior unit normal n. We recall the following form of the divergence theorem

$$\int_\Omega \frac{\partial f}{\partial x_i}\, dx = \int_{\partial\Omega} f n_i \, ds, \quad i = 1,2 \tag{4.1}$$

where n_i is component i of n.

Setting $f = fg$ we get the partial integration formula

$$\int_\Omega \frac{\partial f}{\partial x_i} g \, dx = -\int_\Omega f \frac{\partial g}{\partial x_i}\, dx + \int_{\partial\Omega} f g n_i \, ds, \quad i = 1,2 \tag{4.2}$$

Applying (4.2) with $f = w_i$, the components of a vector field w on Ω, and $g = v$, and taking the sum over $i = 1,2$ we obtain

$$\int_\Omega (\nabla \cdot w) v \, dx = -\int_\Omega w \cdot \nabla v \, dx + \int_{\partial\Omega} (w \cdot n) v \, ds \tag{4.3}$$

Finally, choosing $w = -\nabla u$ in (4.3) we obtain Green's formula

$$\int_\Omega -\Delta u v \, dx = \int_\Omega \nabla u \cdot \nabla v \, dx - \int_{\partial\Omega} n \cdot \nabla u v \, ds \tag{4.4}$$

4.2 The Finite Element Method for Poisson's Equation

4.2.1 Poisson's Equation

In two dimensions Poisson's equation takes the form: find u such that

$$-\Delta u = f, \quad \text{in } \Omega \tag{4.5a}$$

$$u = 0, \quad \text{on } \partial\Omega \tag{4.5b}$$

where $\Delta = \partial^2/\partial x_1^2 + \partial^2/\partial x_2^2$ is the Laplace operator, and $f \in L^2(\Omega)$ is a given function.

4.2.2 Variational Formulation

To derive a variational formulation of Poisson's equation we multiply $-\Delta u = f$ by a test function v, which is assumed to vanish on the boundary $\partial\Omega$, and integrate using Green's formula (i.e., integration by parts). This yields

$$\int_\Omega f v \, dx = -\int_\Omega \Delta u v \, dx \tag{4.6}$$

$$= \int_\Omega \nabla u \cdot \nabla v \, dx - \int_{\partial\Omega} n \cdot \nabla u v \, ds \tag{4.7}$$

$$= \int_\Omega \nabla u \cdot \nabla v \, dx \tag{4.8}$$

since $v = 0$ on $\partial\Omega$. Further, introducing the space V_0, defined by

$$V_0 = \{ v : \|\nabla v\| + \|v\| < \infty, \, v|_{\partial\Omega} = 0 \} \tag{4.9}$$

we have the following variational formulation of (4.5): find $u \in V_0$ such that

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V_0 \tag{4.10}$$

With this choice of test and trial space V_0 the integrals $\int_\Omega \nabla u \cdot \nabla v \, dx$ and $\int_\Omega f v \, dx$ are well defined. To see this, note that due to the Cauchy-Schwartz inequality we have $\int_\Omega f v \, dx \leq \|f\| \|v\|$, which is less than infinity by assumption. A similar line of reasoning applies to the other integral. Indeed, V_0 is the largest space with the property that the integrals in the variational formulation exist.

In this context we would like to a point out a subtlety that we have not yet touched upon. Even though the solution to Poisson's equation (4.5) is also a solution to the variational formulation (4.10), the opposite is generally not true. To see this it suffice to note that the solution to the variational equation does not have to be twice differentiable. For this reason the variational formulation is sometimes called the weak form, as opposed to the original strong form. Proving that a weak solution is also a strong solution can be tricky, since it depends on the shape of the domain and regularity of the coefficients.

4.2.3 Finite Element Approximation

Now, let $\mathscr{K}$ be a triangulation of Ω, and let V_h be the space of continuous piecewise linears on $\mathscr{K}$. Also, to satisfy the strong boundary conditions, let $V_{h,0} \subset V_h$ be the subspace

$$V_{h,0} = \{v \in V_h : v|_{\partial\Omega} = 0\} \tag{4.11}$$

With this choice of approximation space the finite element method for (4.5) takes the form: find $u_h \in V_{h,0}$ such that

$$\int_\Omega \nabla u_h \cdot \nabla v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V_{h,0} \tag{4.12}$$

4.2.4 Derivation of a Linear System of Equations

To compute the finite element approximation u_h let $\{\varphi_i\}_{i=1}^{n_i}$ be a basis for $V_{h,0}$ with n_i hat functions. Here, n_i is the number of internal nodes in the mesh, since the functions of $V_{h,0}$ vanish on the boundary.

We note that the finite element method (4.12) is equivalent to

$$\int_\Omega \nabla u_h \cdot \nabla \varphi_i \, dx = \int_\Omega f \varphi_i \, dx, \quad i = 1,2,\dots,n_i \tag{4.13}$$

Since u_h belongs to V_h it can be written as

$$u_h = \sum_{j=1}^{n_i} \xi_j \varphi_j \tag{4.14}$$

with N unknowns ξ_j, $j = 1,2,\dots,n_i$, to be determined.

Inserting the ansatz (4.14) into (4.13) we get

$$\int_\Omega f\varphi_i\,dx = \int_\Omega \nabla u_h \cdot \nabla\varphi_i\,dx \tag{4.15}$$

$$= \int_\Omega \nabla\left(\sum_{j=1}^{n_i} \xi_j\varphi_j\right)\cdot\nabla\varphi_i\,dx \tag{4.16}$$

$$= \sum_{j=1}^{n_i} \xi_j \int_\Omega \nabla\varphi_j\cdot\nabla\varphi_i\,dx, \quad i = 1,2,\dots,n_i \tag{4.17}$$

Using the notation

$$A_{ij} = \int_\Omega \nabla\varphi_j\cdot\nabla\varphi_i\,dx, \quad i,j = 1,2,\dots,n_i \tag{4.18}$$

$$b_i = \int_\Omega f\varphi_i\,dx, \quad i = 1,2,\dots,n_i \tag{4.19}$$

we have

$$b_i = \sum_{j=1}^{n_i} A_{ij}\xi_j, \quad i = 1,2,\dots,n_i \tag{4.20}$$

which is a linear system for the unknowns ξ_j. In matrix form we write this

$$b = A\xi \tag{4.21}$$

where the entries of the $n_i \times n_i$ stiffness matrix A, and the $n_i \times 1$ load vector b is defined by (4.18) and (4.19), respectively. Solving the linear system (4.20) we obtain the unknowns ξ_j, and thus u_h.

4.2.5 Basic Algorithm to Compute the Finite Element Solution

The following algorithm summarizes the basic steps for computing the finite element solution u_h.

Algorithm 10 Basic Finite Element Algorithm

```
1: Create a triangulation $\mathscr{K}$ of $\Omega$ and define the corresponding space of continuous piecewise
   linear functions $V_{h,0}$ hat function basis $\{\varphi_i\}_{i=1}^{n_i}$.
2: Assemble the $n_i \times n_i$ stiffness matrix $A$ and the $n_i \times 1$ load vector $b$, with entries
   $$A_{ij} = \int_\Omega \nabla\varphi_j \cdot \nabla\varphi_i \, dx, \quad b_i = \int_\Omega f\varphi_i \, dx \tag{4.22}$$
3: Solve the linear system
   $$A\xi = b \tag{4.23}$$
4: Set
   $$u_h = \sum_{j=1}^{n_i} \xi_j \varphi_j \tag{4.24}$$
```

4.3 Basic Analysis of the Finite Element Method

4.3.1 Existence and Uniqueness of the Finite Element Solution

Theorem 4.1. *The finite element solution u_h, defined by* (4.12) *exists and is unique.*

Proof. We first show the uniqueness claim. The argument is by contradiction. Assume that there are two finite element solutions u_h and $\tilde{u}_h$ satisfying (4.12). Then we have

$$\int_\Omega \nabla u_h \cdot \nabla v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V_{h,0} \tag{4.25}$$

$$\int_\Omega \nabla \tilde{u}_h \cdot \nabla v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V_{h,0} \tag{4.26}$$

Subtracting these equations we get

$$\int_\Omega \nabla(u_h - \tilde{u}_h) \cdot \nabla v \, dx = 0, \quad \forall v \in V_{h,0} \tag{4.27}$$

Next setting $v = u_h - \tilde{u}_h \in V_{h,0}$ we get

$$\int_\Omega |\nabla(u_h - \tilde{u}_h)|^2 \, dx = 0 \tag{4.28}$$

From this identity we conclude that $u_h - \tilde{u}_h$ must be a constant function. However, using the boundary conditions we see that this constant must be zero, since $u_h = \tilde{u}_h = 0$ on the boundary.

To prove existence we recall that the finite element solution is determined by a square $n_i \times n_i$ linear system of equations. The existence of a solution to such a system of linear equations follows from the uniqueness of the solution.

4.3.2 A Priori Error Estimates

In this section we present the basic error estimates for the finite element approximation u_h. The basic goal is to understand in what sense the error $u - u_h$ becomes small when the mesh is refined.

Theorem 4.2 (Galerkin Orthogonality). *The finite element approximation u_h, defined by* (4.12)*, satisfies the orthogonality*

$$\int_\Omega \nabla(u - u_h) \cdot \nabla v \, dx = 0, \quad \forall v \in V_{h,0} \tag{4.29}$$

Proof. From the variational formulation we have

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V_0 \tag{4.30}$$

and from the definition of the finite element method

$$\int_\Omega \nabla u_h \cdot \nabla v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V_{h,0} \tag{4.31}$$

Subtracting these and using the fact that $V_{h,0} \subset V_0$ immediately proves the claim.

To estimate the error we introduce the following norm called the energy norm on V_0

$$|||v|||^2 = \int_\Omega \nabla v \cdot \nabla v \, dx \tag{4.32}$$

Note that $|||v||| = \|\nabla v\|_{L^2(\Omega)}$.

The next theorem is a best approximation result.

Theorem 4.3. *The finite element solution u_h, defined by* (4.12) *satisfies*

$$|||u - u_h||| \leq |||u - v|||, \quad \forall v \in V_{h,0} \tag{4.33}$$

Proof. Writing $u - u - h = u - v + v - u_h$ for any $v \in V_{h,0}$ we have

$$\begin{aligned}
|||u - u_h|||^2 &= \int_\Omega \nabla(u - u_h) \cdot \nabla(u - u_h) \, dx & (4.34)\\
&= \int_\Omega \nabla(u - u_h) \cdot \nabla(u - v) \, dx + \int_\Omega \nabla(u - u_h) \cdot \nabla(v - u_h) \, dx & (4.35)\\
&= \int_\Omega \nabla(u - u_h) \cdot \nabla(u - v) \, dx & (4.36)\\
&\leq |||u - u_h||| \, |||u - v||| & (4.37)
\end{aligned}$$

where we used the Galerkin orthogonality property (4.29) to conclude that

$$\int_\Omega \nabla(u - u_h) \cdot \nabla(v - u_h) \, dx = 0 \tag{4.38}$$

since $v - u_h \in V_{h,0}$. Dividing by $|||u - u_h|||$ concludes the proof.

This shows that the finite element solution u_h is the closest of all functions in V_h to the exact solution u when measuring distance using the energy norm. Next we use this result together with interpolation estimates to study how the error depends on the mesh size.

Theorem 4.4. *The finite element solution u_h, defined by* (4.12) *satisfies the a priori error estimate*

$$|||u - u_h|||^2 \le \sum_{K \in \mathscr{K}} C h_K^2 \|D^2 u\|^2_{L^2(K)} \tag{4.39}$$

with a constant C independent of h_K.

Proof. Starting from the best approximation result (4.33) choosing $v = \pi u$ and using the interpolation estimate (3.19) we have

$$\begin{aligned} |||u - u_h|||^2 &\le |||u - \pi u|||^2 && (4.40)\\ &= \sum_{K \in \mathscr{K}} \|D(u - \pi u)\|^2_{L^2(K)} && (4.41)\\ &\le \sum_{K \in \mathscr{K}} C h_K^2 \|D^2 u\|^2_{L^2(K)} && (4.42) \end{aligned}$$

which proves the estimate. Here, we tacitly assume that u is two times differentiable so that the quantity $D^2 u$ is well defined.

Defining $h = \max_{K \mathscr{K}} h_K$ we conclude that

$$|||u - u_h||| \le Ch \|D^2 u\|_{L^2(\Omega)} \tag{4.43}$$

and thus the gradient of the error tends to zero as the maximum mesh size h tend to zero.

The energy norm $|||\cdot|||$ is useful as it allows a simple derivation of the a priori error estimate (4.39). However, it is not a natural norm such as the L^2-norm, for instance. To deduce a primitive L^2-estimate it is possible to use the Poincaré inequality

$$\|v\| \le C\|\nabla v\| = C|||v||| \tag{4.44}$$

which hold for any function $v \in V_0$. Using Theorem 4.4 we then have

$$\|u - U\| \le C|||u - U||| \le Ch\|D^2 u\| \tag{4.45}$$

which is the desired L^2 estimate. The problem with this estimate is that it is suboptimal in the sense that we expect the L^2 error to be proportional to h^2 and not h since we are using piecewise linears to approximate the solution. The next theorem shows that this is indeed the case.

Theorem 4.5. *The finite element solution u_h, defined by* (4.12) *satisfies the a priori error estimate*

$$\|u - u_h\| \leq Ch^2 \|D^2 u\| \tag{4.46}$$

with a constant C independent of h.

Proof. The proof utilizes a trick called Nitsche's trick, which is really not a trick at all but a rather general technique for deriving error estimates. Anyway, let $e = u - u_h$ denote the error, and let ϕ be the solution of the so-called dual, or adjoint, problem

$$-\Delta\phi = e, \quad \text{in } \Omega \tag{4.47a}$$

$$\phi = 0, \quad \text{on } \partial\Omega \tag{4.47b}$$

Multiplying $-\Delta\phi = e$ by e and integrating using Green's formula as usual we have

$$\int_\Omega e^2\, dx = -\int_\Omega e\Delta\phi\, dx \tag{4.48}$$

$$= \int_\Omega \nabla e \cdot \nabla\phi \cdot dx - \int_{\partial\Omega} en \cdot \nabla\phi\, ds \tag{4.49}$$

$$= \int_\Omega \nabla e \cdot \nabla\phi\, dx \tag{4.50}$$

$$= \int_\Omega \nabla e \cdot \nabla(\phi - \pi\phi)\, dx \tag{4.51}$$

where we have used Galerkin orthogonality (4.29) in the last line to subtract an interpolant $\pi\phi \in V_{h,0}$ to ϕ. Further, using the Cauchy-Schwartz inequality we obtain

$$\|e\|^2 \leq \|\nabla e\| \|\nabla(\phi - \pi\phi)\| \tag{4.52}$$

Now, assuming that the domain Ω does not have any reentrant corners or cusps it can generally be shown that $\|D^2\phi\|$ is proportional to $\|\Delta\phi\|$. Combining this result with the standard interpolation estimate for $\nabla(\phi - \pi\phi)$ and recalling that $-\Delta\phi = e$ we obtain

$$\|\nabla(\phi - \pi\phi)\| \leq Ch\|D^2\phi\| \leq Ch\|\Delta\phi\| = Ch\|e\| \tag{4.53}$$

Thus, by virtue of Theorem (4.4) we have

$$\|e\|^2 \leq \|\nabla e\| \|\nabla(\phi - \pi\phi)\| \tag{4.54}$$

$$\leq Ch\|D^2 u\| Ch\|e\| \tag{4.55}$$

Dividing by $\|e\|$ concludes the proof.

4.3.3 Properties of the Stiffness Matrix

Theorem 4.6. *The stiffness matrix is symmetric and positive definite.*

Proof. A is symmetric since $A_{ij} = A_{ji}$. To prove that A is positive definite we shall show that $\xi^T A\xi > 0$ for all $n_i \times 1$ vectors ξ not equal to zero. Now, straight forward calculations reveals that

$$\begin{aligned}
\xi^T A\xi &= \sum_{i,j=1}^{n_i} \xi_i A_{ij} \xi_j && (4.56)\\
&= \sum_{i,j=1}^{n_i} \xi_i \xi_j \int_\Omega \nabla\varphi_j \cdot \nabla\varphi_i \, dx && (4.57)\\
&= \int_\Omega \nabla\left(\sum_{i=1}^{n_i} \xi_i\varphi_i\right) \cdot \nabla\left(\sum_{j=1}^{n_i} \xi_j\varphi_j\right) dx && (4.58)\\
&= \left\|\nabla\left(\sum_{i=1}^{n_i} \xi_i\varphi_i\right)\right\|^2 && (4.59)
\end{aligned}$$

which is greater than zero as long as the sum $s = \sum_{i=1}^{n_i} \xi_i\varphi_i$ is not a constant function. Now, using the fact that $s \in V_{h,0}$ and that the only constant function in $V_{h,0}$ is the zero function, we see that $\xi^T A\xi = 0$ if and only if $\xi = 0$.

4.4 A Problem with Variable Coefficients

With the aim of writing a general finite element solver we next consider a slightly more challenging model problem involving variable coefficients and more general boundary conditions.

$$\begin{aligned}
-\nabla \cdot (a\nabla u) &= f, && \text{in } \Omega && (4.60a)\\
-n \cdot (a\nabla u) &= \kappa(u - g_D) - g_N, && \text{on } \partial\Omega && (4.60b)
\end{aligned}$$

where $a > 0$, f, κ, g_D, and g_N are given functions.

We shall seek a solution to this problem in the space

$$V = \{v : \|v\| + \|\nabla v\| < \infty\} \qquad (4.61)$$

Multiplying $-\nabla \cdot (a\nabla u) = f$ by a test function $v \in V$ and integrating using Green's formula we have

$$\begin{aligned}
\int_\Omega f v \, dx &= \int_\Omega -\nabla \cdot (a\nabla u) v \, dx && (4.62)\\
&= \int_\Omega a\nabla u \cdot \nabla v \, dx - \int_{\partial\Omega} n \cdot (a\nabla u) v \, ds && (4.63)\\
&= \int_\Omega a\nabla u \cdot \nabla v \, dx + \int_{\partial\Omega} (\kappa(u - g_D) - g_N) v \, ds && (4.64)
\end{aligned}$$

where we used the boundary condition to replace $-n \cdot a\nabla u$ by $\kappa(u - g_D) - g_N$.

Collecting terms we get the following variational formulation: find $u \in V$ such that

$$\int_\Omega a\nabla u \cdot \nabla v\, dx + \int_{\partial\Omega} \kappa u v\, ds = \int_\Omega f v\, dx + \int_{\partial\Omega} (\kappa g_D + g_N) v\, ds, \quad \forall v \in V \tag{4.65}$$

Based on this variational form we may now formulate a finite element method: find $u_h \in V_h \subset V$ such that

$$\int_\Omega a\nabla u_h \cdot \nabla v\, dx + \int_{\partial\Omega} \kappa u_h v\, ds = \int_\Omega f v\, dx + \int_{\partial\Omega} (\kappa g_D + g_N) v\, ds, \quad \forall v \in V_h \tag{4.66}$$

4.5 Computer Implementation

Writing a finite element solver can be quite complicated for higher dimensional problems, and therefore we shall take a moment to go trough the implementation of the finite element method (4.66). The linear system resulting form this discretization process takes the form

$$(A+R)\xi = b + r \tag{4.67}$$

where the entries of the involved matrices and vectors are given by

$$A_{ij} = \int_\Omega a\nabla\varphi_i \cdot \nabla\varphi_j\, dx, \tag{4.68}$$

$$R_{ij} = \int_{\partial\Omega} \kappa\varphi_i\varphi_j\, ds, \tag{4.69}$$

$$b_i = \int_\Omega f\varphi_i\, dx, \tag{4.70}$$

$$r_i = \int_{\partial\Omega} (\kappa g_D + g_N)\varphi_i\, ds \tag{4.71}$$

with indices $i, j = 1, 2, \dots, n_p$ with n_p the number of nodes in the mesh.

4.5.1 Assembly of the Stiffness Matrix

The assembly of the stiffness matrix A is performed in the same manner as shown previously for the mass matrix M. Of course, the matrix entries of A are different then those of M. The local element stiffness matrix is given by

$$A^K_{ij} = \int_K a\nabla\varphi_i \cdot \nabla\varphi_j\, dx, \quad i, j = 1, 2, 3 \tag{4.72}$$

We shall now compute these 9 integrals.

Consider a triangle K with nodes $N_i = (x_1^{(i)}, x_2^{(i)})$, $i = 1,2,3$. To each node N_i there is a hat function φ_i associated, which takes the value 1 at node N_i and 0 at the other two nodes. Each hat function is a linear function on K so it has the form

$$\varphi_i = a_i + b_i x_1 + c_i x_2 \tag{4.73}$$

where the coefficients a_i, b_i, and c_i, are determined by

$$\varphi_i(N_j) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \tag{4.74}$$

The explicit expresssions for the coefficients a_i, b_i, and c_i are given by

$$a_i = \frac{x_1^{(j)} x_2^{(k)} - x_1^{(k)} x_2^{(j)}}{2|K|}, \quad b_i = \frac{x_2^{(j)} - x_2^{(k)}}{2|K|}, \quad c_i = \frac{x_1^{(k)} - x_1^{(j)}}{2|K|}$$

with cyclic permutation of the index i, j, k over $1,2,3$.

The gradient of φ_i is just the constant vector $\nabla\varphi_i = [b_i, c_i]^T$. The gradients $\nabla\varphi_i$ will occur very frequently, so therefore let us write a special routine for computing them.

```
function [area,b,c] = Gradients(x,y)
area=polyarea(x,y);
b=[y(2)-y(3); y(3)-y(1); y(1)-y(2)]/2/area;
c=[x(3)-x(2); x(1)-x(3); x(2)-x(1)]/2/area;
```

Input `x` and `y` are two vectors holding the node coordinates of the triangle. Output are the vectors `b` and `c` holding the coeffcients b_i and c_i of the gradients. Since the area is computed as a by product we also return it in the variable `area`.

Once we have $\nabla\varphi_i$ it is easy to compute the local stiffness matrix. Using the center of gravity quadrature formula we have

$$A_{ij}^K = \int_K a \nabla\varphi_i \cdot \nabla\varphi_j \, dx \tag{4.75}$$

$$= (b_i b_j + c_i c_j) \int_K a \, dx \tag{4.76}$$

$$\approx \bar{a}(b_i b_j + c_i c_j)|K|, \quad i, j = 1,2,3 \tag{4.77}$$

where $\bar{a} = a(\bar{x})$ with $\bar{x} = (N_1 + N_2 + N_3)/3$ the center of gravity of K.

We summarize the assembly of the global stiffness matrix with the following algorithm.

Algorithm 11 Assembly of the Stiffness Matrix

```
1: Let $n$ be the number of nodes and $m$ the number of elements in a mesh, and let the mesh be
   described by its point matrix $P$ and connectivity matrix $T$.
2: Allocate memory for the $n \times n$ matrix $A$ and initialize all matrix entries to zero.
3: for $K = 1,2,\dots,m$ do
4:    Compute the gradients $\nabla\varphi_i = [b_i, c_i]$, $i = 1,2,3$ of the three hat functions $\varphi_i$ on $K$.
5:    Compute the $3 \times 3$ local element mass matrix $A^K$ given by
          $$A^K = \bar{a}\begin{bmatrix} b_1^2+c_1^2 & b_1b_2+c_1c_2 & b_1b_3+c_1c_3 \\ b_2b_1+c_2c_1 & b_2^2+c_2^2 & b_2b_3+c_2c_3 \\ b_3b_1+c_3c_1 & b_3b_2+c_3c_2 & b_3^2+c_3^2 \end{bmatrix}|K| \tag{4.78}$$
6:    Set up the local to global mapping, loc2glb $= [r,s,t]$.
7:    for $i = 1,2,3$ do
8:       for $j = 1,2,3$ do
9:
             $$A_{\text{loc2glb}_i\text{loc2glb}_j} = A_{\text{loc2glb}_i\text{loc2glb}_j} + A_{ij}^K \tag{4.79}$$
10:      end for
11:   end for
12: end for
```

It is straight forward to translate this algorithm into MATLAB code.

```
function A = StiffMat2D(p,t,a)
np = size(p,2);
nt = size(t,2);
A = sparse(np,np);
for K = 1:nt
  loc2glb = t(1:3,K); % local-to-global map
  x = p(1,loc2glb); % node x-coordinates
  y = p(2,loc2glb); % node y-
  [area,b,c] = Gradients(x,y);
  xc = mean(x); yc = mean(y); % element centroid
  abar = a(xc,yc); % value of a(x,y) at centroid
  AK = abar*(b*b'...
    +c*c')*area; % element stiffness matrix
  A(loc2glb,loc2glb) = A(loc2glb,loc2glb) ...
    + AK; % add element stiffnesses to A
end
```

A few comments about this routine are perhaps in order. For each element we compute the area and the gradients $\nabla\varphi_i = [b_i, c_i]^T$ using the `Gradients` routine. The local element stiffness matrix A^K is then the sum of the outer product of these vectors times the element area $|K|$ and $\bar{a}$ (i.e., `AK = abar*(b*b'+c*c')*area`. The function a is assumed to be defined by a separate routine. Finally, A^K is added to the appropriate places in A using the vectorized command `A(loc2glb,loc2glb) = A(loc2glb,loc2glb) + AK`. Input is the point and triangle matrix describing the mesh, and a function handle `a` to the routine specifying a. Output is the assembled global stiffness matrix A.

The load vector b is exactly the same as for the L^2-projection and assembled as shown before.

We remark that the stiffness and mass matrices and the load vector can also be assembled with the built-in routine `assema`. In the simplest case the syntax for doing so is

```
[A,M,b] = assema(p,t,1,1,1);
```

4.5.2 Assembling the Boundary Conditions

We must also assemble the boundary matrix R and the boundary vector r containing line integrals originating from the Robin boundary condition. To do so we observe that if two nodes N_i and N_j of a triangle K are located on the domain boundary $\partial\Omega$, then the edge E between them will contribute to matrix entry R_{ij}, and to vector entries r_i and r_j. In particular, we have the local element boundary matrix and vector

$$R_{ij}^E = \int_E \kappa \varphi_i \varphi_j \, ds = \frac{1}{6}\kappa(1+\delta_{ij})|E|, \quad i,j = 1,2 \tag{4.80}$$

$$r_i^E = \int_E (\kappa g_D + g_N)\varphi_i \, ds = \frac{1}{2}(\kappa g_D + g_N)|E|, \quad i = 1,2 \tag{4.81}$$

where $|E|$ is the length of E. For simplicity, we have assumed that κ, g_D, and g_N are constant on E.

We can think of R as a one-dimensional mass matrix on a mesh with nodes located along $\partial\Omega$ instead of along the x_1-axis. As a consequence, the assembly routines for these matrices are very similar.

MATLAB stores starting and ending nodes for the line segments on the mesh boundary in the first two rows of the edge matrix `e`, which is output from `initmesh`. To assemble R we loop over these edges and for each edge we add the entries of the local element boundary matrix R^K to the appropriate entries in the global boundary matrix R. We list the code for this below.

```
function R = RobinMat2D(p,e,kappa)
np = size(p,2); % number of nodes
ne = size(e,2); % number of boundary segments
R = sparse(np,np);
for E = 1:ne
  loc2glb = e(1:2,E); % boundary nodes
  x = p(1,loc2glb); % node x-coordinates
  y = p(2,loc2glb); % node y-
  len = sqrt((x(1)-x(2))^2+(y(1)-y(2))^2); % edge length
  xc = mean(x); yc = mean(y); % element centroid
  k = kappa(xc,yc); % value of kappa at centroid
```

```
    RK = k/6*[2 1; 1 2]*len; % element boundary matrix
    R(loc2glb,loc2glb) = R(loc2glb,loc2glb) + RK;
  end
```

Input is the point and edge matrix describing the mesh, and a function handle to a routine specifying κ. Output is the assembled global boundary matrix R.
The boundary vector r can be assembled similarly.

```
  function r = RobinVec2D(p,e,kappa,gD,gN)
  np = size(p,2);
  ne = size(e,2);
  r = zeros(np,1);
  for E = 1:ne
    loc2glb = e(1:2,E);
    x = p(1,loc2glb);
    y = p(2,loc2glb);
    len = sqrt((x(1)-x(2))^2+(y(1)-y(2))^2);
    xc = mean(x); yc = mean(y);
    tmp = kappa(xc,yc)*gD(xc,yc)+gN(xc,yc);
    rK = tmp*[1; 1]*len/2;
    r(loc2glb) = r(loc2glb) + rK;
  end
```

4.5.3 A Finite Element Solver for Poisson's Equation

Next we present a physical application that can be simulated with the code written so far.

4.5.3.1 Potential Flow Over a Wing

When designing aircrafts it is very important to know the areodynamical properties of the wings to assess among other things the lift force. Therefore we now simulate the flow of air over a wing. For simplicity we the wing to be very long so that the problem can be reduced to two dimensions. Figure 4.1 shows a rectangular domain surrounding a cross section of the wing. A potential equation for the airflow around the wing follows from the somewhat unphysical assumption that the velocity field u is steady and irrotational, that is, $\partial_t u = 0$ with t time and $\nabla \times u = 0$. Then there exists a scalar function ϕ such that $u = -\nabla\phi$. This is called the flow potential and is given as the solution of the Laplace equation

$$-\Delta\phi = 0 \tag{4.82}$$

We impose the following boundary conditions

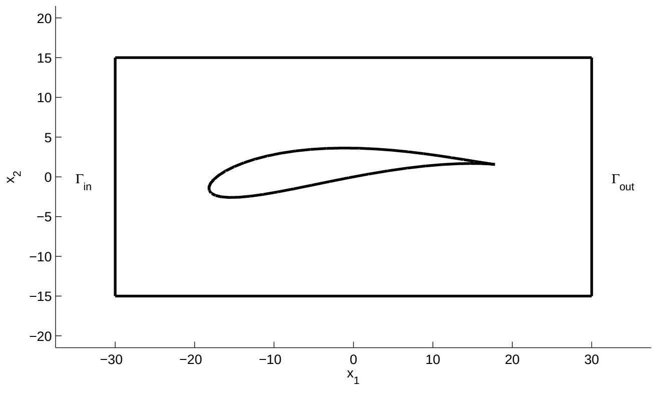


Fig. 4.1 Mesh surrounding a wing profile.

$$n \cdot \nabla\phi = 1, \quad \text{on } \Gamma_{\text{in}} \tag{4.83}$$
$$\phi = 0, \quad \text{on } \Gamma_{\text{out}} \tag{4.84}$$
$$n \cdot \nabla\phi = 0, \quad \text{elsewhere} \tag{4.85}$$

A slight complication with the boundary conditions is that the Dirichlet condition must be approximated with the Robin condition we have implemented. To this end we set $\kappa = 10^6$ on Γ_{out} which penalizes any deviation of the solution from zero on this boundary segment. On Γ_{in} we set $\kappa = 0$ and $g_N = 1$.

We write the following subroutines, which specify κ, g_D and g_N.

```
function z = Kappa(x,y)
z=0;
if (x>29.99), z=1.e+6; end

function z = g_D(x,y)
z=0;

function z = g_N(x,y)
z=0;
if (x<-29.99), z=1; end
```

We also need the following subroutine to specify $a = 1$.

```
function z = One(x,y)
z=1;
```

The velocity potential can now be compted with just a couple of code lines

```
function PoissonSolver2D()
wing = Airfoil();
[p,e,t] = initmesh(wing,'hmax',0.5);
```

```
A = StiffMat2D(p,t,@One);
R = RobinMat2D(p,e,@Kappa);
r = RobinVec2D(p,e,@Kappa,@g_D,@g_N);
phi = (A+R)\r;
pdecont(p,t,phi)
```

Here, `Airfoil` is a subroutine specifying the geometry matrix. It is listed in the Appendix.

Figure 4.2 shows the computed finite element approximation Φ to the velocity potential.

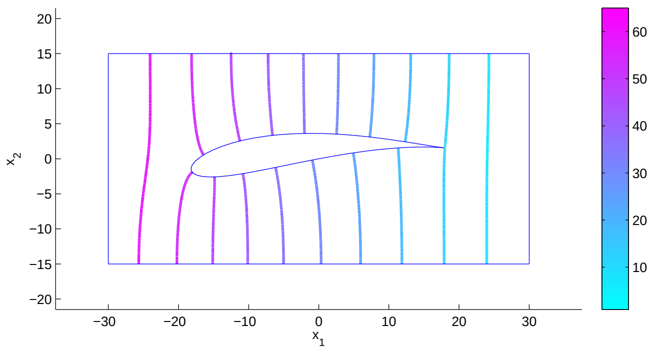


Fig. 4.2 Isocontours of the computed finite element velocity potential Φ.

The velocity field u is defined by $u = -\nabla\phi$. Its computed counterpart can be visualized by typing

```
[phix,phiy] = pdegrad(p,t,phi); % derivatives of 'phi'
u = -phix;
v = -phiy;
pdeplot(p,e,t,'flowdata',[u; v]')
```

Figure 4.3 shows the result.

Fig. 4.3 Velocity glyphs around the wing.

Finally, a pressure around the wing can be defined by $p = -|\nabla\Phi|^2$. In Figure 4.4 we show this pressure.

Fig. 4.4 Pressure isocontours around the wing.

In the next three sections we shall study three problems that demand special attention, namely, the pure Dirichlet problem, the pure Neumann problem and the Eigenvalue problem.

4.6 The Dirichlet Problem

We consider the following model problem with inhomogeneous boundary conditions: find u such that

$$-\Delta u = f, \quad \text{in } \Omega \tag{4.86a}$$

$$u = g, \quad \text{on } \partial\Omega \tag{4.86b}$$

where f and g are given functions.

This problem has different trial and test space due to the inhomogeneous strong boundary condition. The trial space is given by

$$V_g = \{v : \|v\| + \|\nabla v\| < \infty,\ v|_{\partial\Omega} = g\} \tag{4.87}$$

whereas the test space is given by V_0.

Multiplying equation $-\Delta u = f$ by a test function $v \in V_0$ and integrating using Green's formula as usual we obtain

$$\int_\Omega f v \, dx = -\int_\Omega \Delta u v \, dx \tag{4.88}$$

$$= \int_\Omega \nabla u \cdot \nabla v \, dx - \int_{\partial\Omega} n \cdot \nabla u v \, ds \tag{4.89}$$

$$= \int_\Omega \nabla u \cdot \nabla v \, dx \tag{4.90}$$

since $v = 0$ on $\partial\Omega$. Thus, we obtain the variational formulation: find $u \in V_g$ such that

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V_0 \tag{4.91}$$

Now, let us assume that g is the restriction of a continuous piecewise linear function to the boundary. In other words there is a function $u_{h,g} \in V_h$ such that $u_{h,g} = g$ on $\partial\Omega$. If this is not the case we have to first approximate g by such a function, for instance using interpolation on the boundary.

Introducing the affine subspace

$$V_{h,g} = \{v \in V_h : v|_{\partial\Omega} = g\} \tag{4.92}$$

the finite element method reads: find $u_h \in V_{h,g}$ such that

$$\int_\Omega \nabla u_h \cdot \nabla v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V_{h,0} \tag{4.93}$$

To derive an equation for u_h we write it in the form

$$u_h = u_{h,0} + u_{h,g} \tag{4.94}$$

where $u_{h,g}$ is any fixed function in $V_{h,g}$ and $u_{h,0} = 0$ on $\partial\Omega$ and, thus, $u_{h,0} \in V_{h,0}$. This construction of u_h will satisfy the boundary conditions since $u_{h,g} = g$ on the boundary. Further, since $u_{h,g}$ is known it remains to determine $u_{h,0}$. We get the equation: find $u_{h,0} \in V_{h,0}$ such that

$$\int_\Omega \nabla u_{h,0} \cdot \nabla v \, dx = \int_\Omega f v \, dx - \int_\Omega \nabla u_{h,g} \cdot \nabla v \, dx, \quad \forall v \in V_{h,0} \tag{4.95}$$

This is a problem of the same kind as above but with a modified right hand side. One can prove that u_h is independent of the particular choice of $u_{h,g}$. In practice $u_{h,g}$ is often chosen to be zero at all interior nodes.

The implementation of this can be done as follows. Let n_p be the total number of nodes and let us assume that that the first n_i of these are interior, while the remaining $n_b = n_p - n_i$ nodes lie on the boundary. Further, let A and b be the usual $n_p \times n_p$ stiffness matrix and $n_p \times 1$ load vector output from `assema`. The linear system resulting from equation (4.95) can be written as

$$\begin{bmatrix} A_{00} & A_{0g} \\ 0 & I \end{bmatrix} \begin{bmatrix} \xi_0 \\ \xi_g \end{bmatrix} = \begin{bmatrix} b_0 \\ g \end{bmatrix} \tag{4.96}$$

where A_{00} is the upper left $n_i \times n_i$ block of A, A_{0g} the $n_i \times n_b$ upper right block block of A, I the $n_b \times n_b$ identity matrix, b_0 the first n_i entries of b, g the n_b boundary node values, and ξ_0 and ξ_g the nodal values of $u_{h,0}$ and $u_{h,g}$, respectively. Rearranging the first n_i equations we obtain the discrete counterpart of (4.95)

$$A_{00}\xi_0 = b_0 - A_{0g}\xi_g = b_0 - A_{0g}g \tag{4.97}$$

from which the interior values ξ_0 can be determined.

The translation of this to MATLAB is straight forward. Suppose we have a vector `fixed` holding the numbers of all boundary nodes, and another (column) vector `g` holding the corresponding nodal values. Then, we can form and solve equation (4.97) with the following piece of code.

```
[A,unused,b] = assema(p,t,...); % assemble
np = size(p,2); % total number of nodes
```

```
fixed = unique([e(1,:) e(2,:)]); % boundary nodes
free = setdiff([1:np],fixed); % interior nodes
b = b(free)-A(free,fixed)*g; % modify stiffness matrix
A = A(free,free); % modify load vector
xi = zeros(np,1); % allocate solution vector
xi(fixed) = g; % insert fixed node values
xi(free) = A\b; % solve for free node values
```

4.7 The Neumann Problem

Next we consider the following model problem: find u such that

$$-\Delta u = f, \quad \text{in } \Omega \tag{4.98a}$$

$$n \cdot \nabla u = g, \quad \text{on } \partial\Omega \tag{4.98b}$$

where f and g are given functions. This problem is solvable provided f and g satisfies the conservation property

$$\int_\Omega f\,dx + \int_{\partial\Omega} g\,ds = 0 \tag{4.99}$$

Note however that the solution is only uniquely determined up to a constant, since any constant satisfies the problem with $f = g = 0$. A common trick to remedy this is to impose the additional constraint

$$\int_\Omega u\,dx = 0 \tag{4.100}$$

We shall seek a weak solution to (4.98) in the space $V = \{v : \|\nabla v\| + \|v\| < \infty,\ \int_\Omega v\,dx = 0\}$. Multiplying equation $-\Delta u = f$ by a test function $v \in V$ and integrating using Green's formula we have

$$\int_\Omega fv\,dx = -\int_\Omega \Delta uv\,dx \tag{4.101}$$

$$= \int_\Omega \nabla u \cdot \nabla v\,dx - \int_{\partial\Omega} n \cdot \nabla uv\,ds \tag{4.102}$$

$$= \int_\Omega \nabla u \cdot \nabla v\,dx - \int_{\partial\Omega} gv\,ds \tag{4.103}$$

where we used the Neumann boundary condition $n \cdot \nabla u = g$. Thus, we obtain the weak formulation: find $u \in V$ such that

$$\int_\Omega \nabla u \cdot \nabla v\,dx = \int_\Omega fv\,dx + \int_{\partial\Omega} gv\,ds, \quad \forall v \in V \tag{4.104}$$

The finite element method now reads: find $u_h \in V_h$ such that

$$\int_\Omega \nabla u_h \cdot \nabla v \, dx = \int_\Omega f v \, dx + \int_{\partial\Omega} g v \, ds, \quad \forall v \in V_h \tag{4.105}$$

Based on this formulation a linear system $A\xi = b$ may now be derived in the same way as usual. In doing so, the constraint $\int u_h \, dx = 0$ can be enforced using the Lagrange multiplier technique. In doing so, the basic idea is as follows. The solution ξ to an $n_p \times n_p$ linear system $A\xi = b$ with A symmetric and positive definite is also the minimizer of the quadratic form $Q(\xi) = \xi^T A \xi - \xi^T b$. Now, if we have a set of n_c constranits for x, say, $Cx = 0$ with C a given $n_c \times n_p$ matrix, then a fundamental result from optimization says that ξ is found by seeking a stationary point for the Lagrangian

$$L(\xi, \mu) = Q(\xi) - \mu^T C \xi \tag{4.106}$$

where μ is an $n_c \times 1$ vector called the Lagrange multiplier. Differentialing L with respect to ξ and μ and utilizing the first order optimality condition $L_\xi = L_\mu = 0$ leads to the augmented linear system

$$\begin{bmatrix} A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \xi \\ \mu \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix} \tag{4.107}$$

from which ξ can be obtained. Loosely speking we may think of μ as a force which acts to enforce the constraints. In the case that A is the stiffness matrix, ξ a vector holding node values of u_h, and the constraint is a u_h with zero mean value, C is just a $1 \times n_p$ vector with entries $C_i = \int_\Omega \varphi_i \, dx$, where φ_i is a hat function. This is due to the fact that we then have $\int_\Omega u_h \, dx = C\xi$. Moreover, since $C\xi = 0$ is a constraint which does not violate the underlying partial differential equation μ should vanish or at least be very small.

4.8 The Eigenvalue Problem

The last of our model problems is the eigenvalue problem: find the function u and the number λ such that

$$-\Delta u = \lambda u, \quad \text{in } \Omega \tag{4.108a}$$

$$n \cdot \nabla u = 0, \quad \text{on } \partial\Omega \tag{4.108b}$$

Here, we have for simplicity assumed a Neumann condition on the boundary, but Dirichlet conditions may also be applied. All the same the boundary conditions must be homogenous though.

The significant feature of an eigenvalue problem is that the solution u appears in both the left and right hand side of the equation. Moreover, we seek both the function u and the number λ. We say that u is an eigenfunction, or eigenmode, and λ an eigenvalue.

The finite element discretization for eigenvalue problems is, however, similar to standard problems.

Multiplying $-\Delta u = \lambda u$ by a test function $v \in V$ and integrating using Green's formula we have

$$\lambda \int_\Omega uv\,dx = -\int_\Omega \Delta uv\,dx \tag{4.109}$$

$$= \int_\Omega \nabla u \cdot \nabla v\,dx - \int_{\partial\Omega} n \cdot \nabla uv\,ds \tag{4.110}$$

$$= \int_\Omega \nabla u \cdot \nabla v\,dx \tag{4.111}$$

Thus, the weak formulation reads: find $u \in V$ and $\lambda \in \mathbb{R}$ such that

$$\int_\Omega \nabla u \cdot \nabla v\,dx = \lambda \int_\Omega uv\,dx, \quad \forall v \in V \tag{4.112}$$

The finite element method takes the form: find $u_h \in V_h$ and $\Lambda \in \mathbb{R}$ such that

$$\int_\Omega \nabla u_h \cdot \nabla v\,dx = \Lambda \int_\Omega u_h v\,dx, \quad \forall v \in V_h \tag{4.113}$$

The finite element discretization leads not to a linear system, but to a generalized algebraic eigenvalue problem of the form

$$A\xi = \Lambda M\xi \tag{4.114}$$

where A and M are the usual stiffness and mass matrices, and ξ is a vector holding the nodal values of u_h. The existence of a solution to this eigenvalue problem follows from the spectral theorem. The eigenvectors ξ and eigenvalues Λ come in pairs (ξ, Λ), and there are as many pairs $(\xi_i, \Lambda_i)_{i=1}^{n_p}$ as there are nodes n_p in the mesh. Moreover, the eigenvalues Λ_i are real positive and of increasing magnitude. This is a consequence of the fact that both A and M are symmetric. Another consequence is that the corresponding eigenvectors ξ_i are orthogonal with respect to A, and orthonormal with respect to M. In MATLAB generalized sparse eigenvalue problems can be solved using the `eigs` routine. Below we show how to compute the first five eigenmodes with smallest magnitude on a disk. This geometry is predefined in MATLAB. Assembly of the matrices A and M is done using the `assema` routine.

```
g = 'circleg'; % built-in geometry of a cricle
[p,e,t] = initmesh(g,'hmax',0.1); % mesh
[A,M] = assema(p,t,1,1,0); % assemble A and M
[Xi,La] = eigs(A,M,5,'SM'); % solve A*Xi=La*M*Xi
pdesurf(p,t,Xi(5,:)) % plot 5:th eigenmode
```

4.9 Adaptive Finite Element Methods

As we have seen a posteriori error estimates are computable error estimates, which can be used to control adaptive mesh refinement and thereby iteratively increase the accuracy of the finite element solution. In this section we formulate adaptive finite elements for Poisson's equation.

4.9.1 A Posteriori Error Estimates

For the model problem (4.5) we have the following a posteriori error estimate.

Theorem 4.7. *The finite element solution u_h, defined by* (4.12)*, satisfies the a posteriori estimate*

$$|||u-u_h|||^2 \leq C \sum_{K\in\mathcal{K}} \rho_K^2(u_h) \tag{4.115}$$

where the element residual $\rho_K(u_h)$ is defined by

$$\rho_K(u_h) = h_K\|f+\Delta u_h\|_{L^2(K)} + \tfrac{1}{2}h_K^{1/2}\|[n\cdot\nabla u_h]\|_{L^2(\partial K\setminus\partial\Omega)} \tag{4.116}$$

Here, $[n\cdot\nabla u_h]$ denotes the jump in the normal derivative of u_h on the edge $\partial K_1\cap\partial K_2$, shared by any two elements K_1 and K_2, that is,

$$[n\cdot\nabla u_h]|_{\partial K_1\cap\partial K_2} = n_1\cdot\nabla u_h|_{K_1} + n_2\cdot\nabla u_h|_{K_2} \tag{4.117}$$

with n_i the exterior unit normal of K_i.

Proof. Letting $e=u-u_h$ be the error we have

$$|||e|||^2 = \|\nabla e\|^2 = \int_\Omega \nabla e\cdot\nabla e\,dx = \int_\Omega \nabla e\cdot\nabla(e-\pi e)\,dx \tag{4.118}$$

where we have use the Galerkin orthogonality to subtract an interpolant $\pi e\in V_{h,0}$. Splitting this into a sum over the elements and using Green's formula we further have

$$|||e|||^2 = \sum_{K\in\mathcal{K}} \int_K \nabla e\cdot\nabla(e-\pi e)\,dx \tag{4.119}$$

$$= \sum_{K\in\mathcal{K}} -\int_K \Delta e(e-\pi e)\,dx + \int_{\partial K} n\cdot\nabla e(e-\pi e)\,ds \tag{4.120}$$

$$= \sum_{K\in\mathcal{K}} \int_K (f+\Delta u_h)(e-\pi e)\,dx + \int_{\partial K}[n\cdot\nabla u_h]/2(e-\pi e)\,ds \tag{4.121}$$

The result in the last line follows from (4.120) by noting that there are two contributions for each interior edge $\partial K_1\cap\partial K_2$, one from triangle K_1 and one from triangle K_2. Summing these contributions we get

$$\int_{\partial K_1 \cap \partial K_2} (n_1 \cdot \nabla e_1 (e_1 - \pi e_1) + n_2 \cdot \nabla e_2 (e_2 - \pi e_2))\, ds, \tag{4.122}$$

where $e_i = e|_{K_i}$ and n_i is the exterior unit normal of K_i for $i = 1,2$. Using the facts that the exact solution has a continuous normal derivative and that the error and its interpolant are continuous we get

$$\int_{\partial K_1 \cap \partial K_2} [n \cdot \nabla u_h](e - \pi e)\, ds \tag{4.123}$$

We proceed with estimating the right hand side of (4.121). First we estimate the interior contribution using the Cauchy-Schwartz inequality followed by an interpolation error estimate

$$\begin{aligned} \int_K (f + \Delta u_h)(e - \pi e)\, dx &\leq \|f + \Delta u_h\|_K \|e - \pi e\|_K & (4.124) \\ &\leq \|f + \Delta u_h\|_K C h_K \|De\|_K & (4.125) \end{aligned}$$

For the the edge contributions we need the following inequality called the trace inequality

$$\|v\|^2_{L^2(K)} \leq C(h_K^{-1} \|v\|^2_{L^2(K)} + h_K \|\nabla v\|^2_{L^2(K)}) \tag{4.126}$$

We then have, again using Cauchy-Schwartz inequality,

$$\begin{aligned} \int_{\partial K} [n \cdot \nabla u_h]/2(e - \pi e)\, ds &\leq \|[n \cdot \nabla u_h]/2\|_{\partial K} \|e - \pi e\|_{\partial K} & (4.127) \\ &\leq \|[n \cdot \nabla u_h]/2\|_{\partial K} C(h_K^{-1} \|e - \pi e\|^2_K + h_K \|D(e - \pi e)\|^2_K)^{1/2} & (4.128) \\ &\leq \|[n \cdot \nabla u_h]/2\|_{\partial K} C h_K \|De\|_K & (4.129) \end{aligned}$$

Using these estimates together with the Cauchy-Schwartz inequality the estimate follows directly.

4.9.2 Adaptive Mesh Refinement

There are two important issues to consider when constructing a mesh refinement algorithm for a triangulation. First, invalid triangles (e.g., with hanging nodes) are not allowed and we wish to refine as few elements as possible which are not in the list of elements to be refined. Second, it is important that the minimal angle in the triangulation is kept as large as possible. Otherwise the quality of the finite element solution u_h will deteriorate as we successively refine the mesh.

There are a number of refinement algorithms such as:

- Rivara refinement

- Regular refinement

In the Rivara method a triangle is always refined by inserting a new edge from the midpoint of the longest side to the opposite corner. Regular refinement consists of splitting each triangle into four smaller ones. Both methods will typically manufactures invalid triangles, that is, triangles with hanging nodes. To remedy this further refinement using special refinement techniques is usually employed.

4.9.3 Adaptive Finite Elements using MATLAB

It is easy to write an adaptive finite element solver in MATLAB.

First we create a (coarse) initial mesh

```
g = 'cardg'; % predefined geometry of a cardioid
[p,e,t] = initmesh(g,'hmax',0.25);
```

Then we compute the finite solution u_h

```
[A,unused,b] = assema(p,t,...);
% .. application of B.C. etc ..
xi = A\b;
```

The next step is to evaluate the element residuals ρ_K, defined by (4.94). This can be done with the routine `pdejmps`.

```
rho = pdejmps(p,t,...);
```

The `pdejmps` routine was originally designed for computing the element residuals to $-\nabla \cdot (c\nabla u) + au = f$ and its syntax is therefore

```
rho = pdejmps(p,t,c,a,f,xi,1,1,1);
```

where each of the three inputs `c`, `a`, and `f` can be either a constant or a row vector specifying the values of the coefficients c, a, and f at the mid-points of the triangles.

As our refinement criterion we select the 10% most error prone elements to be refined.

```
tol = 0.9*max(rho);
elements = find(rho > tol);
```

After these calls the vector `elements` contains the element numbers of the elements to be refined.

The actual refinement is done with the `refinemesh` routine.

```
[p,e,t] = refinemesh(g,p,e,t,elements,'regular');
```

The mesh refinement algorithm used here is called longest edge bisection. We use the simple stopping criterion that the maximum number of elements in the mesh must not exceed, say, 10000.

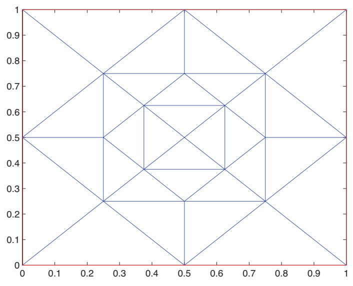
Below we list a complete routine for adaptively solving Poisson's equation $-\Delta u = 1$ on a domain Ω shaped like a cardioid with $u = 0$ on the boundary $\partial\Omega$.

```
function AdaptivePoisson2D()
% set up geometry
g = 'cardg';
% create initial mesh
[p,e,t] = initmesh(g,'hmax',0.25);
% while not too many elements, do
while size(t,2) < 10000
  % assemble stiffness matrix A, and load vector b
  [A,unused,b] = assema(p,t,1,0,1);
  % get the number of nodes
  np = size(p,2);
  % enforce zero Dirichlet BC
  fixed = unique([e(1,:) e(2,:)]);
  free = setdiff([1:np],fixed);
  A = A(free,free);
  b = b(free);
  % solve for finite element solution U
  xi = zeros(np,1);
  xi(free) = A\b;
  figure(1), pdesurf(p,t,U)
  % compute element residuals
  rho = pdejmps(p,t,1,0,1,xi,1,1,1);
  % choose a selection criteria
  tol = 0.9*max(rho);
  % select elements for refinement
  elements = find(rho > tol)';
  % refine elements using regular refinement
  [p,e,t] = refinemesh(g,p,e,t,elements,'regular');
  figure(2), pdemesh(p,e,t)
end
```

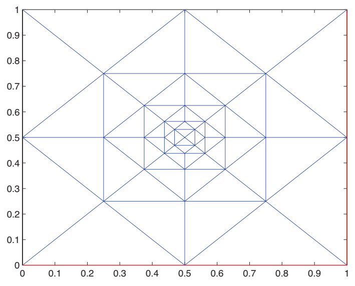
To illustrate adaptive mesh refinement let us solve the problem

$$-\Delta u = 4a^2(1-ar^2)e^{-ar^2}, \quad \text{in } \Omega = [0,1]^2 \tag{4.130a}$$
$$u = 0, \quad \text{on } \partial\Omega \tag{4.130b}$$

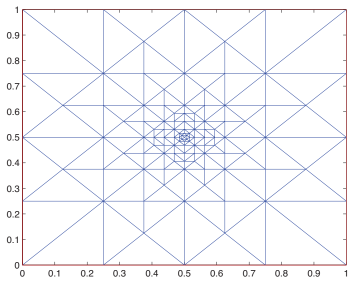
where a is a parameter and $r = \sqrt{(x_1-0.5)^2+(x_2-0.5)}$ is the distance from the center of the unitsquare $\Omega = [0,1]^2$. If a is chosen sufficiently large, say $a = 400$, then the analytical solution is given by $u = ae^{-ar^2}$. This problem is computationally demanding, since the solution is a very narrow pulse, with strong localized gradients, centered at $(0.5, 0.5)$. To obtain a good finite element approximation we thus expect to have to resolve the region around this point by placing many triangles there, but maybe we do not need so many triangles elsewhere. In Figures 4.5 and 4.6 below we show the results of running the adaptive code outlined above for 10 adaptive loops with a 25% refinement rule.



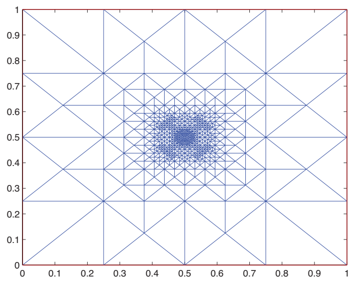
(a) 2 refinements



(b) 4 refinements

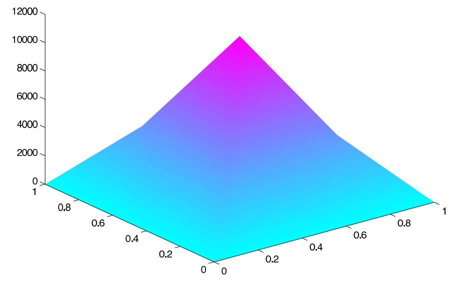


(c) 6 refinements

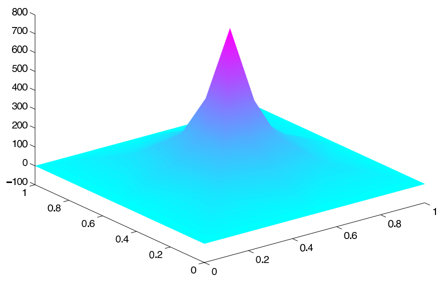


(d) 10 refinements

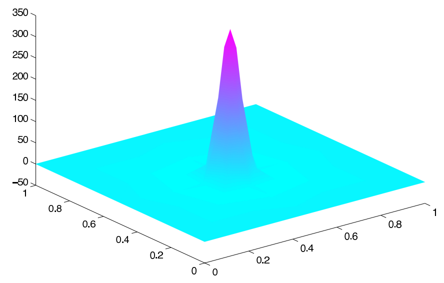
Fig. 4.5 Adaptive meshes for the problem with solution $u = ae^{-ar^2}$.



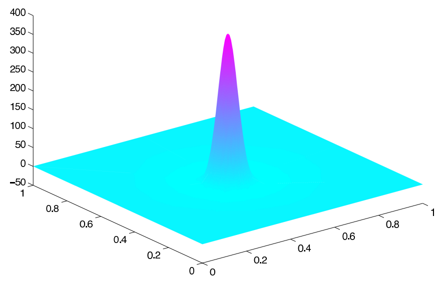
(a) 2 refinements



(b) 4 refinements



(c) 6 refinements



(d) 10 refinements

Fig. 4.6 Adaptively computed approximations to $u = ae^{-ar^2}$.

4.10 Problems

Exercise 4.1. Prove the Cauchy-Schwartz inequality $|\int_\Omega uv\,dx| \leq \|u\|\|v\|$.

Exercise 4.2. Verify that $\|\nabla u\|$ satisfies the requirements of a norm on V_0.

Exercise 4.3. Determine f so that $u = x(1-x)y(1-y)$ is a solution to $-\Delta u = f$ on the unitsquare $\Omega = [0,1]^2$ with $u = 0$ on the boundary $\partial\Omega$. Then compute ∇u, $\|u\|$, and $\|\nabla u\|$.

Exercise 4.4. What are appropriate test and trial spaces for

$$\begin{aligned} -\Delta u &= 0, \quad x \in \Omega \\ u &= 0, \quad x \in \Gamma_D \\ n \cdot \nabla u &= g, \quad x \in \Gamma_N \end{aligned}$$

where Γ_D and Γ_N are two disjunct parts of the boundary and such that $\Gamma_D + \Gamma_N = \partial\Omega$.

Exercise 4.5. Compute the element mass and stiffness matrices on the reference triangle $\bar{K}$ with corners at $(0,0)$, $(1,0)$, and $(0,1)$.

Exercise 4.6. Define the geometry matrix `g` for the domain $\Omega = [-2,3]^2 \setminus [-1,1]^2$ (i.e., a rectangle with a square hole). Use it to make a triangulation of this domain with `initmesh`.

Exercise 4.7. Show that the solution u to

$$\begin{aligned} -\Delta u &= f, \quad x \in \Omega \\ u &= 0, \quad x \in \partial\Omega \end{aligned}$$

satisfies the stability estimate

$$\|\nabla u\| \leq C\|f\|$$

where C is a constant. *Hint:* Multiply with u and integrate by parts. Also, recall the Poincaré inequality $\|w\| \leq C\|\nabla w\|$ which holds for all functions w that are zero at the boundary $\partial\Omega$.

Exercise 4.8. Show, that for

$$\begin{aligned} -\Delta u &= 0, \quad x \in \Omega \\ u &= 0, \quad x \in \Gamma_D \\ n \cdot \nabla u &= g, \quad x \in \Gamma_N \end{aligned}$$

holds the stability estimate

$$\|\nabla u\| \leq C\|g\|_{\Gamma_N}$$

where C is a constant. *Hint:* Use the trace inequality $\|w\|_{\partial\Omega} \leq C(\|w\| + \|\nabla w\|)$.

Exercise 4.9. Write a MATLAB code to assemble the stiffness matrix A on a mesh of a domain of your choice. Use `eigs` to compute the eigenvalues and verify that one eigenvalue is zero. Why?

Exercise 4.10. Prove that

$$\|\nabla(u-u_h)\|^2 = \|\nabla u\|^2 - \|\nabla U\|^2$$

where u is the exact solution and u_h the finite element approximation to

$$\begin{aligned} -\Delta u &= f, \quad x \in \Omega \\ u &= 0, \quad x \in \partial\Omega \end{aligned}$$

Exercise 4.11. Let $\bar{K}$ be the reference triangle with corners at $(0,0)$, $(1,0)$ and $(0,1)$ and let

$$I(r,s) = \int_{\bar{K}} x_1^r x_2^s \, dx$$

where r and s are non-negative integers. Show that

$$\begin{aligned} I(r-1,s-1) &= \frac{s+1}{r} I(r,s) \\ I(r,0) &= \frac{1}{(r+1)(r+2)} \end{aligned}$$

and thus by induction that

$$I(r,s) = \frac{r!s!}{(r+s+2)!}$$

Exercise 4.12. Consider

$$\begin{aligned} -\Delta u + u &= f, \quad x \in \Omega \\ u &= 0, \quad x \in \partial\Omega \end{aligned}$$

(a) Make a variational formulation.
(b) Formulate a finite element method in a suitable piecewise polynomial space V_h.
(c) Deduce the Galerkin orthogonality property

$$\int_\Omega (\nabla(u-u_h)\cdot\nabla v + (u-u_h)v)\, dx = 0, \quad \forall v \in V_h$$

(d) Derive the a priori error estimate

$$\|\nabla(u-u_h)\|^2 + \|u-u_h\|^2 \leq \sum_{K\in\mathscr{K}} Ch_K^2 \|D^2u\|^2_{L^2(K)}$$

Chapter 5
Time-dependent Problems

Abstract Most real-world problems depend on time and in this chapter we shall therefore construct numerical methods for solving time dependent differential equations. We do this by first discretizing in space using finite elements. As a result we obtain a semi-discrete problem in time in the form of a system of ordinary differential equations (ODE). We then discretize in time and solve this ODE system numerically using a finite difference time stepping scheme. As model problems we use two classical equations from mathematical physics, namely, the Heat equation and the Wave equation. To assert the accuracy of the computed solutions we state and prove both stability estimates and a priori error estimates.

5.0.1 Finite Difference Methods for Systems of ODE

We begin this chapter by deriving three simple finite difference methods for systems of ordinary differential equations (ODE).

We wish to find the $n \times 1$ time-dependent solution vector $\xi = \xi(t)$ to the ODE system

$$M\dot{\xi}(t) + A\xi(t) = b(t), \quad 0 < t < T \tag{5.1a}$$

$$\xi(0) = \xi_0 \tag{5.1b}$$

where $\dot{\xi}$ means differentiation with respect to time t, T is the final time, M and A are given constant $n \times n$ matrices, $b(t)$ is a given time-dependent $n \times 1$ vector, and ξ_0 is given $n \times 1$ vector with initial data.

To make a time discretization of (1.5) let $0 = t_0 < t_1 < t_2 < \cdots < t_L = T$ be a time grid on the interval $0 < t < T$ with time steps $k_l = t_l - t_{l-1}$, $l = 1, 2, \dots, L$. Integrating $M\dot{\xi} + A\xi = b$ from t_{l-1} to t_l we have, since M and A are constant matrices,

$$M\int_{t_{l-1}}^{t_l} \dot{\xi}(t)\,dt + A\int_{t_{l-1}}^{t_l} \xi(t)\,dt = \int_{t_{l-1}}^{t_l} b(t)\,dt \tag{5.2}$$

The first integral is trivial to evaluate, yielding

$$M(\xi(t_l) - \xi(t_{l-1})) + A \int_{t_{l-1}}^{t_l} \xi(t)\,dt = \int_{t_{l-1}}^{t_l} b(t)\,dt \tag{5.3}$$

Now, let ξ_l denote an approximation to $\xi(t_l)$, $l = 0, 1, \dots, L$. Given ξ_{l-1}, we may approximate the remaining integrals using quadrature to obtain a time stepping scheme. For instance, using right end-point quadrature we obtain the following equation for ξ_l

$$M(\xi_l - \xi_{l-1}) + k_l A \xi_l = k_l b_l \tag{5.4}$$

or equivalently

$$(M + k_l A)\xi_l = M\xi_{l-1} + k_l b_l \tag{5.5}$$

where we have introduced the notation $b_l = b(t_l)$. Thus, starting with ξ_0 we successively get ξ_l $l = 1, 2, \dots, L$ from the linear system (5.4). This is the so-called backward Euler method.

Algorithm 12 Backward Euler Method

```
1: Create a time grid $0 = t_0 < t_1 < \cdots < t_L = T$ on the interval $0 < t < T$ with $L$ time steps
   $k_l = t_l - t_{l-1}$.
2: Set $\xi_0 = \xi(0)$.
3: for $l = 1, 2, \dots, L$ do
4:     Solve the linear system
           $(M + k_l A)\xi_l = M\xi_{l-1} + k_l b_l$    (5.6)
5: end for
```

Rearranging the terms of (5.4) we obtain

$$M\frac{\xi_l - \xi_{l-1}}{k_l} + A\xi_l = b_l \tag{5.7}$$

from which it is obvious that the time derivative is approximated by the difference quotient

$$\dot{\xi}(t_l) \approx \frac{\xi_l - \xi_{l-1}}{k_l} \tag{5.8}$$

Hence, Euler's method is a finite difference formula.

Had we instead of right end-point quadrature used left dito when approximating the integrals of (5.3) we would have obtained

$$M\xi_l = (M - k_l A)\xi_{l-1} + k_l b_{l-1} \tag{5.9}$$

which gives the so-called forward Euler method. Similarly, use of the Trapezoidal rule leads to

$$\left(M+\frac{k_l}{2}A\right)\xi_l=\left(M-\frac{k_l}{2}A\right)\xi_{l-1}+\frac{k_l}{2}\left(b_l+b_{l-1}\right) \tag{5.10}$$

and the so-called Crank-Nicolson method.

Each of these three method have its own characteristics regarding accuracy, stability, and computational cost. Loosely speaking, forward Euler is very fast, backward Euler numerically stable, and Crank-Nicolson the most accurate.

5.1 The Heat Equation

Having studied numerical methods for ordinary differential equations we shall now do the same for partial differential equations.

5.1.1 Derivation of the Time-dependent Heat Equation

We have already studied the derivation of the Heat equation, but under the assumption of steady state. We now revisit this derivation taking also the dynamics of the heat transfer process into account. Thus, let us consider the same one-dimensional geometry as before with a thin metal rod of length L and cross section area A occupying the interval $0 < x < L$. Let f be a heat source intensity, q the heat flux along the direction of increasing x, and e the internal energy per unit length of the rod. The principle of conservation of energy now states that the rate of change of internal energy equals the sum of net heat flux and produced heat. Thus, we get

$$\int_0^L \dot{e}\,dx=\int_0^L f\,dx+A(0)q(0)-A(L)q(L) \tag{5.11}$$

which can be rewritten as

$$\int_0^L \dot{e}+(Aq)'\,dx=\int_0^L f\,dx \tag{5.12}$$

Assuming that the internal energy e is proportional to temperature T we have

$$e=cT \tag{5.13}$$

where c is a constant of propotionallity called the heat conductivity. As before we also assume that Fourier's law, $q=-kT'$, is valid. Combining these equations we arrive at

$$\int_0^L c\dot{T} + (AkT')'\,dx = \int f\,dx \tag{5.14}$$

from which we infer the time-dependent Heat equation

$$c\dot{T} + (AkT')' = f \tag{5.15}$$

As usual this equation needs to be supplemented by boundary conditions at $x = 0$ and $x = L$ of either Neumann, Dirichlet, or Robin type. These boundary conditions should hold for all times. However, this is not enough to yield a unique solution T. An initial condition of the form $T(x,0) = T_0(x)$, where $T_0(x)$, $0 < x < L$, a given function is also required to specify the solution at the initial time $t = 0$.

5.1.2 Model Problem

Thus, we consider the model problem

$$\dot{u} - (au')' = f, \quad 0 < x < 1, \quad 0 < t \leq T \tag{5.16a}$$
$$u(0,t) = u(1,t) = 0 \tag{5.16b}$$
$$u(x,0) = u_0(x) \tag{5.16c}$$

where $u = u(x,t)$ is the unknown solution that we wish to find, $a = a(x) \geq a_0 > 0$ is a given positive function, $f = f(x,t)$ a given source function, and $u_0(x)$ a given initial condition.

5.1.3 Variational Formulation

Multiplying (5.16) by a function v and integrating by parts over $0 < x < 1$ we have

$$\int_0^1 fv\,dx = \int_0^1 \dot{u}v\,dx - \int_0^1 (au')'v\,dx \tag{5.17}$$
$$= \int_0^1 \dot{u}v\,dx - au'(1)v(1) + au'(0)v(0) + \int_0^1 au'v'\,dx \tag{5.18}$$
$$= \int_0^1 \dot{u}v\,dx + \int_0^1 au'v'\,dx \tag{5.19}$$

where we assumed that $v(0) = v(1) = 0$. Recalling the space $V_0 = \{v : \|v'\| + \|v\| < \infty,\ v(0) = v(1) = 0\}$ we obtain the following variational formulation of (5.11): find u such that, for every fixed t, $u \in V_0$ and

$$\int_0^1 \dot{u}v\,dx + \int_0^1 au'v'\,dx = \int_0^1 fv\,dx, \quad \forall v \in V_0, \quad 0 < t < T \tag{5.20}$$

5.1.4 Spatial Discretization

In order to discretize the variational formulation in space, let $0 = x_0 < x_1 < \cdots < x_n = 1$ be mesh on the interval $0 < x < 1$, and let $V_{h,0} \subset V_0$ be the corresponding subspace of continuous piecewise linears vanishing at $x = 0$ and $x = 1$. The space discrete counterpart of (5.20) takes the form: find u_h such that, for every fixed fixed t, $u_h \in V_{h,0}$ and

$$\int_0^1 \dot{u}_h v\,dx + \int_0^1 a u_h' v'\,dx = \int_0^1 f v\,dx, \quad \forall v \in V_{h,0}, \quad 0 < t < T \tag{5.21}$$

We note that (5.21) is equivalent to

$$\int_0^1 \dot{u}_h \varphi_i\,dx + \int_0^1 a u_h' \varphi_i'\,dx = \int_0^1 f \varphi_i\,dx, \quad i = 1,2,\dots,n-1, \quad 0 < t < T \tag{5.22}$$

where φ_i, $i = 1,2,\dots,n-1$ are the usual hat basis functions for $V_{h,0}$. Note that φ_0 and φ_n do not belong to the basis, since all functions in $V_{h,0}$ are zero at the interval end-points.

We now seek a solution u_h to (5.22) expressed for every fixed t as a linear combination of hat functions $\varphi_j(x)$, $j = 1,2,\dots,n-1$, and time-dependent coefficients $\xi_j(t)$. That is, we make the ansatz

$$u_h(x,t) = \sum_{j=1}^{n-1} \xi_j(t)\varphi_j(x) \tag{5.23}$$

and seek to determine the time-dependent coefficient vector

$$\xi(t) = \begin{bmatrix} \xi_1(t) \\ \xi_2(t) \\ \vdots \\ \xi_{n-1}(t) \end{bmatrix} = \begin{bmatrix} u_h(x_1,t) \\ u_h(x_2,t) \\ \vdots \\ u_h(x_{n-1},t) \end{bmatrix} \tag{5.24}$$

of nodal values of u_h in such a way that (5.22) is satisfied.

We consider carefully the construction of u_h. For every fixed time t, u_h is a continuous piecewise linear function of x with time-dependent nodal values $\xi_j(t)$.

Substituting (5.23) into (5.22) we have

$$\begin{aligned}\int_0^1 f\varphi_i\,dx = {} & \sum_{j=1}^{n-1} \dot{\xi}_j(t) \int_0^1 \varphi_j\varphi_i\,dx \\ & + \sum_{j=1}^{n-1} \xi_j(t) \int_0^1 a\varphi_j'\varphi_i'\,dx, \quad i = 1,2,\dots,n-1, \quad 0 < t < T\end{aligned} \tag{5.25}$$

Using the notation

$$M_{ij} = \int_0^1 \varphi_j \varphi_i \, dx, \quad i, j = 1, 2, \dots, n-1 \tag{5.26}$$

$$A_{ij} = \int_0^1 a \varphi_j' \varphi_i' \, dx, \quad i, j = 1, 2, \dots, n-1 \tag{5.27}$$

$$b_i(t) = \int_0^1 f(t) \varphi_i \, dx, \quad i = 1, 2, \dots, n-1 \tag{5.28}$$

we have

$$b_i(t) = \sum_{j=1}^{n-1} M_{ij} \dot{\xi}_j(t) + \sum_{j=1}^{n-1} A_{ij} \xi_j(t), \quad i = 1, 2, \dots, n-1, \quad 0 < t < T \tag{5.29}$$

which is a system of $n-1$ ODE for the $n-1$ coefficients $\xi_j(t)$, $j = 1, 2, \dots, n-1$. In matrix form we write this

$$M\dot{\xi}(t) + A\xi(t) = b(t), \quad 0 < t < T \tag{5.30}$$

where the entries of the $(n-1) \times (n-1)$ matrices M and A, and the $(n-1) \times 1$ vector b are defined by (5.22), (5.23), and (5.24), respectively. We recognize M, as the mass matrix, A as the stiffness matrix, and $b(t)$ as a time-dependent load vector.

The ODE system (5.30) is sometimes called a spatial semi-discretization of the Heat equation, since the dependence on the space coordinate x has been eliminated.

We thus conclude that the coefficients $\xi_j(t)$, $j = 0, 1, \dots, n$, in the ansatz (5.23) satisfy a system of ODE, which must be solved in order to obtain the space discrete, or semi-discrete, solution u_h.

5.1.5 Time Discretization

To discretize also in time, let $0 = t_0 < t_1 < t_2 < \cdots < t_L = T$ be a time grid on $0 < t < T$ with time steps $k_l = t_l - t_{l-1}$, $l = 1, 2, \dots, L$. Also, let ξ_l denote an approximation to $\xi(t_l)$. Applying the backward Euler method to the ODE system (5.30) we obtain the following algorithm for numerically solving the Heat equation.

Algorithm 13 Backward Euler Metod for the Heat Equation

```
1: Create a mesh with $n$ elements on the interval $0 < x < 1$ and define the corresponding space of
   continuous piecewise linear functions $V_{h,0}$.
2: Create a time grid $0 = t_0 < t_1 < \cdots < t_L = T$ on the interval $0 < t < T$ with $L$ time steps
   $k_l = t_l - t_{l-1}$.
3: Choose $\xi_0$.
4: for $l = 1, 2, \dots, L$ do
5:     Compute the $(n-1) \times (n-1)$ mass and stiffness matrices $M$ and $A$, and the $(n-1) \times 1$ load
       vector $b_l = b(t_l)$ with entries
           $M_{ij} = \int_0^1 \varphi_j \varphi_i \, dx, \quad A_{ij} = \int_0^1 a \varphi_j' \varphi_i' \, dx, \quad (b_l)_i = \int_0^1 f(t_l) \varphi_i \, dx$   (5.31)
6:     Solve the linear system
           $(M + k_l A)\xi_l = M\xi_{l-1} + k_l b_l$   (5.32)
7: end for
```

Here, we observe that it is possible to define a solution approximation U_l at the end of each time step by

$$U_l(x) = \sum_{j=1}^{n-1} (\xi_l)_j \varphi_j(x), \quad l = 0, 1, \dots, L \tag{5.33}$$

This solution approximation is fully discrete in the sense that it is only defined for the discrete times t_l, in which case it is a continuous piecewise linear function on $0 < x < 1$.

Regarding the starting vector ξ_0 there are a few different possible choices of initial data. The simplest choice is to let $\xi_0 = \pi u_0$, that is, the interpolant of u_0 on the mesh. Alternatively, we could let ξ_0 be the nodal vector of the L^2-projection of u_0, but this is of course more computationally expensive. As we shall see there are also other choices for ξ_0, for example, the Ritz projection of u_0 to be presented in the next section.

5.2 Stability Estimates

It is generally of interest to know something about the long term behaviors of the solution to a time-dependent equation. In particular, one would like to know if the solution grows with time or if it can be bounded by the data (e.g., the initial condition and the right hand side) of the equation. For this purpose stability estimates are used.

5.2.1 A Space Discrete Estimate

We first derive a stability estimate for the space discrete solution u_h to the Heat equation (5.16). Recall that $u_h = u_h(t,x)$ is continuous when viewed as a function of time t, but has only a discrete set of degrees of freedom when viewed as a function of the space coordinate x.

Choosing $v = u_h$ in the variational formulation (5.20) we have

$$\int_0^1 \dot{u}_h u_h + a u_h'^2 \, dx = \int_0^1 f u_h \, dx \tag{5.34}$$

Noting that the first term can be written

$$\int_0^1 \dot{u}_h u_h \, dx = \int_0^1 \tfrac{1}{2}\partial_t (u_h^2)\, dx = \tfrac{1}{2}\partial_t \|u_h\|^2 = \|u_h\|\partial_t \|u_h\| \tag{5.35}$$

and using the Cauchy Schwartz inequality we have

$$\|u_h\|\partial_t\|u_h\| + \|\sqrt{a}u_h'\|^2 \leq \|f\|\|u_h\| \tag{5.36}$$

Here, we observe that $\sqrt{a}$ is well defined since by assumption a has minimum value $a_0 > 0$. Thus, dropping the positive term $\|\sqrt{a}u_h'\|^2$ and dividing by $\|u_h\|$ we further have

$$\partial_t \|u_h\| \leq \|f\| \tag{5.37}$$

Finally, integrating this result with respect to time from 0 to t we obtain the stability estimate

$$\|u_h(\cdot,t)\| = \|u_h(\cdot,0)\| + \int_0^t \|f(\cdot,s)\|\, ds \tag{5.38}$$

which shows that the size of u_h is bounded in time by the initial condition $u_h(\cdot,0)$ and the source function f.

5.2.2 A Fully Discrete Estimate

Let us also derive a stability estimate for the fully discrete solution U_l, defined for each discrete time t_l, $l = 0,1,\dots,L$, by (5.33). To do so we multiply the backward Euler method with the vector ξ_l, which gives

$$\xi_l^T (M + k_l A)\xi_n = \xi_l^T (M\xi_{l-1} + b_l) \tag{5.39}$$

This is equivalent to

$$\|U_l\|^2 + k_l \|\sqrt{a}U_l'\|^2 = \int_0^1 U_{l-1}U_l\,dx + k_l \int_0^1 f_l U_l\,dx \tag{5.40}$$

where $f_l = f(t_l)$. Using again the Cauchy-Schwartz inequality we have

$$\|U_l\|^2 + k_l \|\sqrt{a}U_l'\|^2 \leq \|U_{l-1}\|\|U_l\| + k_l\|f_l\|\|U_l\| \tag{5.41}$$

Now, dropping first the positive term $\|\sqrt{a}U_l'\|^2$ and then dividing by $\|U_l\|$, we get

$$\|U_l\| \leq \|U_{l-1}\| + k_l\|f_l\| \tag{5.42}$$

Iterated used of this result implies that

$$\|U_l\| \leq \|U_0\| + \sum_{i=1}^{l} k_i\|f_i\| \tag{5.43}$$

which is our stability estimate.

This shows that the size of U_l is bounded for all times by the timestep k_l, the initial condition U_0, and the source function f.

5.3 A Priori Error Estimates

Loosely speaking error estimates for time-dependent problems can be derived by combining error estimates for the corresponding stationary problem with stability estimates. We shall use this approach to derive error estimates for the Heat equation.

5.3.1 Ritz projection

Ritz projection is a technique for approximating a given function u, and is very similar to L^2-projection. Both L^2- and Ritz projection compute the orthogonal projection of u onto a finite dimensional subspace with respect to a certain scalar product. For L^2-projection the subspace is V_h and the scalar product the usual L^2-product $\int uv\,dx$. However, for Ritz projection the subspace is $V_{h,0}$ and the scalar product $\int au'v'\,dx$, where $a \geq a_0 > 0$ is a positive weight function. The practical consequence of this is that the mass matrix should be replaced by the stiffness matrix when switching from computing L^2- to Ritz projections. We shall not study Ritz projection in depth, but only state its definition and approximation properties.

The Ritz projection $R_h u \in V_{h,0}$ to a given function $u \in V_0$ is defined by

$$\int_0^1 a(u - R_h u)'v'\,dx = 0, \quad \forall v \in V_{h,0} \tag{5.44}$$

With this definition we have the following approximation result.

Proposition 5.1. *The following estimate holds.*

$$\|u - R_h u\| \leq Ch^2 \|D^2 u\| \tag{5.45}$$

Proof. The proof follows from a duality argument using Nitsche's trick and the dual problem $-(a\phi')' = u - R_h u$ with boundary conditions $\phi(0) = \phi(1) = 0$. We omit the details.

5.3.2 A Space Discrete Estimate

Theorem 5.1. *The space discrete solution u_h defined by* (5.23) *satisfies the a priori estimate*

$$\|u(t) - u_h(t)\| \leq Ch^2 (\|u_0''\| + \int_0^t \|\dot{u}''(\cdot, s)\| \, ds) \tag{5.46}$$

Proof. We use the Ritz projection $R_h u$ to rewrite the error $u - u_h$ as the sum

$$u - u_h = u - R_h u + R_h u - u_h = \rho + \theta \tag{5.47}$$

We can bound the first term $\rho = u - R_h u$ by observing that

$$\begin{aligned} \|\rho(\cdot, t)\| &\leq Ch^2 \|u''(\cdot, t)\| & (5.48) \\ &\leq Ch^2 \|u''(\cdot, 0) + \int_0^t \dot{u}''(\cdot, s) \, ds\| & (5.49) \\ &\leq Ch^2 (\|u_0''\| + \int_0^t \|\dot{u}''(\cdot, s)\| \, ds) & (5.50) \end{aligned}$$

To bound the second term $\theta = R_h u - u_h$ we insert it into the variational formulation (5.20), yielding

$$\begin{aligned} \int_0^1 \dot{\theta} v \, dx + \int_0^1 a\theta' v' \, dx &= \int_0^1 \dot{(R_h u - u_h)} v \, dx + \int_0^1 a(R_h u - u_h)' v' \, dx & (5.51) \\ &= \int_0^1 \dot{R_h u} v \, dx - \int_0^1 \dot{u}_h v \, dx - \int_0^1 a u_h' v' \, dx + \int_0^1 a R_h u' v' \, dx & (5.52) \\ &= \int_0^1 \dot{R_h u} v \, dx + \int_0^1 f v \, dx + \int_0^1 a u' v' \, dx & (5.53) \\ &= \int_0^1 \dot{R_h u} v \, dx - \dot{u} v \, dx & (5.54) \\ &= -\int_0^1 \dot{\rho} v \, dx & (5.55) \end{aligned}$$

From this we make the key observation that θ satisfies the Heat equation (5.16) with $-\dot{\rho}$ as right hand side. The unspoken hope is now that θ will be small since we know that ρ is of order h^2. To show that this is indeed the case we use the space discrete stability estimate (5.38). We get

$$\|\theta(\cdot,t)\| \leq \|\theta(\cdot,0)\| + \int_0^t \|\dot{\rho}(\cdot,s)\|\, ds \tag{5.56}$$

By choosing $u_h(x,0) = R_h u(x,0) = R_h u_0(x)$, we can eliminate $\|\theta(\cdot,0) = R_h u(\cdot,0) - u_h(\cdot,0)\|$. Finally, we note that

$$\|\dot{\rho}(\cdot,t)\| = \partial_t \|u(\cdot,t) - R_h u(\cdot,t)\| \leq Ch^2 \partial_t \|u''(\cdot,t)\| = Ch^2 \|\dot{u}''(\cdot,t)\| \tag{5.57}$$

5.3.3 A Fully Discrete Estimate

Theorem 5.2. *The fully discrete solution U_l defined by* (5.33) *satisfies the a priori estimate*

$$\|u(t) - U_l\| \leq Ch^2(\|u_0''\| + \int_0^t \|\dot{u}''(\cdot,s)\|\, ds) + Ck \int_0^t \|\ddot{u}''(\cdot,s)\|\, ds \tag{5.58}$$

where k is a uniform time step on $0 < t < T$.

Proof. We assume that the time grid is uniform with a time step k. Again we write the error $u(t_l) - U_l = (u(t_l) - R_h u(t_l)) + (R_h u(t_l) - U_l) = \rho_l + \theta_l$. As before ρ_l can be bounded by

$$\|\rho_l\| \leq Ch^2(\|u_0''\| + \int_0^{t_l} \|\dot{u}''(\cdot,s)\|\, ds) \tag{5.59}$$

To bound also θ_l we insert it into the backward Euler method, which after some elementary manipulations

$$\int_0^1 \frac{\theta_l - \theta_{l-1}}{k} v\, dx + \int_0^1 a\theta_l' v'\, dx = -\int_0^1 \omega_l v\, dx \tag{5.60}$$

where

$$\omega_l = \dot{u}(t_l) - \frac{R_h u(t_l) - R_h u(t_{l-1})}{k} \tag{5.61}$$

Adding and subtracting $k^{-1}(u(t_l) - u(t_{l-1}))$ from ω_l we have

$$\omega_l = \left(\dot{u}(t_l) - \frac{u(t_l) - u(t_{l-1})}{k}\right) \tag{5.62}$$
$$+ \left(\frac{u(t_l) - R_h u(t_l)}{k} - \frac{u(t_{l-1}) - R_h u(t_{l-1})}{k}\right)$$
$$= \omega_l^1 + \omega_l^2 \tag{5.63}$$

Applying the fully discrete stability estimate (5.43) we infer

$$\|\theta_l\| \leq \|\theta_0\| + k\sum_{i=1}^{l} \|\omega_i^1\| + k\sum_{i=1}^{l} \|\omega_i^2\| \tag{5.64}$$

As before, θ_0 and can be eliminated by choosing $U_0 = R_h u_0$. Now, from Taylors formula it follows that

$$\dot{u}(t_l) - \frac{u(t_l) - u(t_{l-1})}{k} = -\frac{1}{k}\int_{t_{l-1}}^{t_l} (t_{l-1} - s)\ddot{u}(\cdot, s)\, ds \tag{5.65}$$

which gives

$$k\sum_{i=1}^{l} \|\omega_i^1\| \leq \sum_{i=1}^{l} \|\int_{t_{n-1}}^{t_l} (t_{l-1} - s)\ddot{u}(\cdot, s)\, ds\| \leq k\int_0^{t_l} \|\ddot{u}(\cdot, s)\|\, ds \tag{5.66}$$

Furthermore, noting that

$$\frac{u(t_l) - R_h u(t_l)}{k} - \frac{u(t_{l-1}) - R_h u(t_{l-1})}{k} = \frac{1}{k}\int_{t_{l-1}}^{t_l} (I - R_h)\dot{u}(x, s)\, ds \tag{5.67}$$

and using that $\|u - R_h u\| \leq Ch^2 \|u''\|$ we have

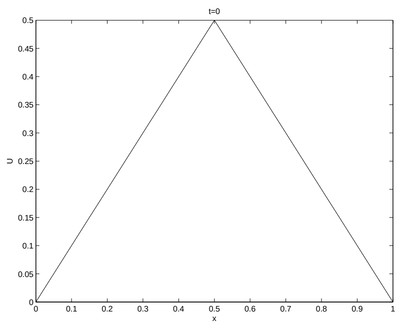
$$k\sum_{i=1}^{l} \|\omega_i^2\| \leq \sum_{i=1}^{l} \int_{t_{l-1}}^{t_l} Ch^2 \|\dot{u}''(\cdot, s)\|\, ds \leq Ch^2 \int_0^{t_l} \|\dot{u}''(\cdot, s)\|\, ds \tag{5.68}$$

Together these estimates prove the theorem.

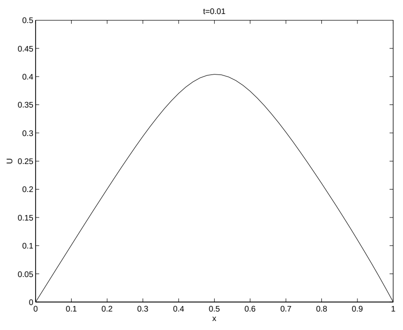
5.4 Computer Implementation

Heat is always spreading through the process of diffusion, which means that the entire volume of a body will eventually become warm even if the body is heated only in a single spot. This is built into the Heat equation, which is said to have smoothing properties. Let us illustrate this property with a numerical example. The amount of diffusion is given by the coefficient a. Consider the model problem with $a = 1$, $f = 2x$, and $u_0 = 0.5 - |x - 0.5|$. The initial condition looks like a triangle with its peak at $x = 0.5$. The steady state solution is given by $u(x, \infty) = \frac{3}{2}x(x^2 - x)$, which is assumed after roughly 0.5 time units. Figure 5.1 shows a series of snapshots of the computed solution as it evolves towards steady state. From the figure we see that the peak of the triangle quickly diffuses and disappears, which shows the smoothing property of the equation. The code for this simulation is listed below. We reuse the assembly routines written for computing L^2-projections and two-point boundary value problems. Note that the assembly of the mass and stiffness matrices and also the load vector can be done outside the time loop since neither of them are time-dependent in this case.

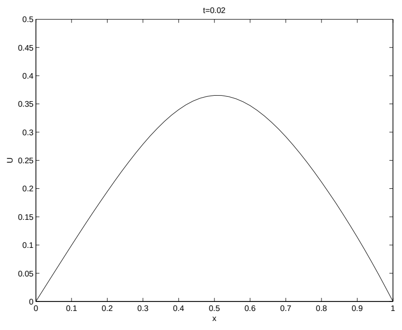
```
function HeatSolver1D()
h = 0.01; % mesh size
x = 0:h:1; % mesh
L = 100; % number of time levels
T = 0.5; % final time
t = linspace(0,T,L+1) % time grid
U = 0.5-abs(0.5-x)'; % inital condition
kappa = [1.e+6 1.e+6]; % Robin BC parameters
g = [0 0];
A = StiffMat1D(x,@One,kappa); % stiffness matrix
M =  MassMat1D(x); % mass matrix
b =  LoadVec1D(x,@Twox,kappa,g); % load vector
for l = 1:L % time loop
  k = t(l+1) - t(l); % time step;
  U = (M + k*A)\(M*U + k*b); % backward Euler method
  plot(x,U), axis([0 1 0 1]), pause(0.1) % plot
end

function y = One(x)
y = 1; % coefficient a=1

function y = Twox(x)
y = 2*x; % function f=2x
```



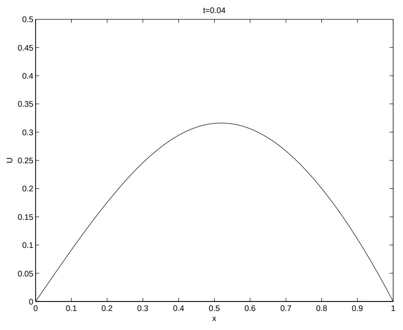
(a) $t = 0$



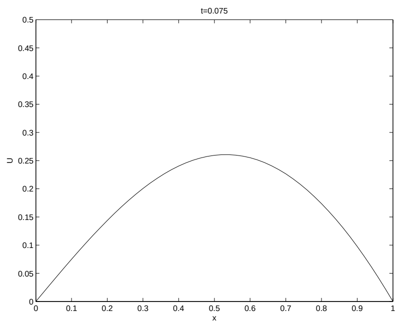
(b) $t = 0.01$



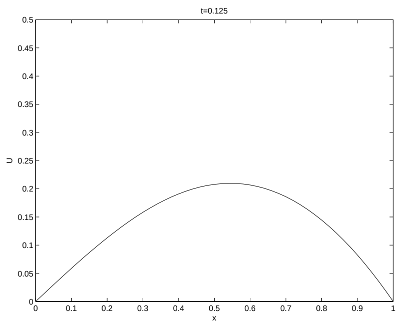
(c) $t = 0.02$



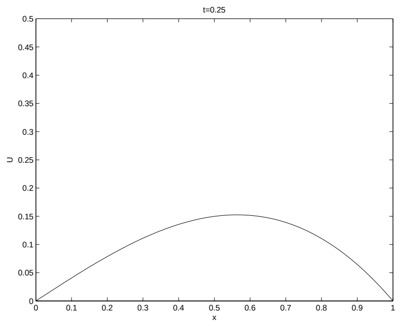
(d) $t = 0.04$



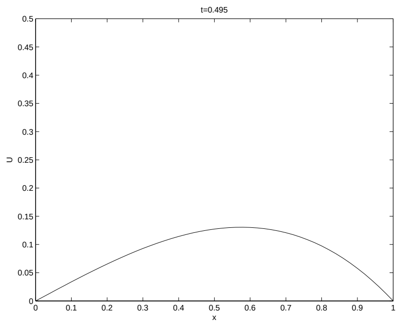
(e) $t = 0.075$



(f) $t = 0.125$



(g) $t = 0.25$



(h) $t = 0.5$

Fig. 5.1 Snapshots showing transient solution evolving to steady state. Note the fast smoothing of the initial peak.

5.5 The Wave Equation

As we have seen the Heat equation quickly diffuses any high gradients to produce smooth solutions at steady state. This is a typical feature for equations involving just one time derivative. We shall now study what happens if the number of time derivatives is increased from one to two, because it turns out that this seemingly small change dramatically alters the behavior of the solutions. Indeed this new equation allows for oscillating solutions and does not have a steady state. The equation is called the Wave equation.

5.5.1 Derivation of the Acoustic Wave Equation

The Wave equation is a frequently occurring partial differential equation in engineering and scientific applications and can be derived in many ways. Below we derive it from the point of view of acoustics. The acoustic Wave equation describes sound waves in a continuum (i.e., liquid or gas), and in this context sound is interpreted as a pressure disturbance. To this end let Ω be a domain occupied by a continuum with density ρ, pressure p, and velocity u. Our basic hypothesis is that any instantaneous movement of a small volume of matter within the continuum is counteracted by the built up of a pressure gradient. Thus, Newton's second law, that is, net force equals mass times acceleration, gives us

$$\rho \dot{u} = -\nabla p \tag{5.69}$$

Further, if this movement leads to expansion of the small volume of matter, then a pressure drop must occur to preserve energy. Similarly, volume contraction leads to a rise in pressure. Now, a local measure of expansion and contraction is the divergence of u, which suggests the relation

$$\dot{p} = -K\nabla \cdot u \tag{5.70}$$

where K is a constant of proportionality indicating the incompressibility (i.e., resistance to compression) of the continuum. Differentiating (5.70) with respect to time and using (5.69) we obtain

$$\ddot{p} = -K\nabla \cdot \dot{u} = K\nabla \cdot \frac{\nabla p}{\rho} \tag{5.71}$$

If ρ abd K are constant this simplifies to

$$\ddot{p} = c^2 \Delta p \tag{5.72}$$

where $c^2 = K/\rho$. This is the acoustic Wave equation.

The boundary conditions for the Wave equation are the same as for any equation involving the Laplace term $-\Delta p$, and can be of either Dirichlet, Neumann, or Robin type. However, since the Wave equation also involves a term $\ddot{p}$ with two time derivatives there must be two initial conditions, one for p and one for $\dot{p}$. These take the form $p(x,t_0) = u_0$ and $\dot{p}(x,t_0) = v_0$ at the staring time t_0.

Needless to say solutions to the Wave equation look like waves. Hence, the name.

5.5.2 Model Problem

We consider the following model problem

$$\ddot{u} - c^2 \Delta u = f, \quad \text{in } \Omega \times I \tag{5.73a}$$

$$u = 0, \quad \text{on } \Omega \times I \tag{5.73b}$$

$$u = u_0, \quad \text{in } \Omega, \text{ for } t = 0 \tag{5.73c}$$

$$\dot{u} = v_0, \quad \text{in } \Omega, \text{ for } t = 0 \tag{5.73d}$$

where $I = (0,T]$ is the time interval, c^2 is a parameter, f is a given source function, and u_0 and v_0 given initial conditions.

5.5.3 Variational Formulation

Multiplying $\ddot{u} - c^2 \Delta u = f$ by a test function v, which is zero on the boundary, and integrating using Green's formula we have

$$\int_\Omega f v \, dx = \int_\Omega \ddot{u} v \, dx - c^2 \int_\Omega \Delta u \, dx \tag{5.74}$$

$$= \int_\Omega \ddot{u} v \, dx + c^2 \int_\Omega \nabla u \cdot \nabla v \, dx - c^2 \int_{\partial\Omega} n \cdot \nabla u v \, ds \tag{5.75}$$

$$= \int_\Omega \ddot{u} v \, dx + c^2 \int_\Omega \nabla u \cdot \nabla v \, dx, \quad t \in I \tag{5.76}$$

Using the familiar space $V_0 = \{v : \|\nabla v\| + \|v\| < \infty,\ v|_{\partial\Omega} = 0\}$ the variational formulation of (5.73) reads: find u such that for every fixed t, $u \in V_0$ and

$$\int_\Omega \ddot{u} v \, dx + c^2 \int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V_0, \quad t \in I \tag{5.77}$$

5.5.4 Spatial Discretization

Let $V_{h,0} \subset V_0$ be the subspace of continuous piecewise linears on a mesh $\mathscr{K}$ of Ω. The space discrete counterpart of (5.77) reads: find u_h such that for every fixed t, $u_h \in V_{h,0}$ and

$$\int_\Omega \ddot{u}_h v \, dx + c^2 \int_\Omega \nabla u_h \cdot \nabla v \, dx = \int_\Omega f v \, dx, \quad \forall v \in V_{h,0}, \quad t \in I \tag{5.78}$$

We note that (5.78) is equivalent to

$$\int_\Omega \ddot{u}_h \varphi_i \, dx + c^2 \int_\Omega \nabla u_h \cdot \nabla \varphi_i \, dx = \int_\Omega f \varphi_i \, dx, \quad i = 1, 2, \ldots, n_i, \quad t \in I \tag{5.79}$$

where φ_i, $i = 1, 2, \ldots, n_i$ are the usual hat basis functions for $V_{h,0}$ and n_i the number of internal nodes in the mesh.

Next we make a space discrete ansatz

$$u_h = \sum_{j=1}^{n_i} \xi_j(t) \varphi_j \tag{5.80}$$

where ξ_j are n_i time-dependent coefficients to be determined.

Substituting (5.80) into (5.79) we have

$$\begin{aligned} \sum_{j=1}^{n_i} \ddot{\xi}_j(t) \int_\Omega \varphi_j \varphi_i \, dx + \sum_{j=1}^{n_i} \xi_j(t) c^2 \int_\Omega \nabla \varphi_j \cdot \nabla \varphi_i \, dx \\ = \int_\Omega f \varphi_i \, dx, \quad i = 1, 2, \ldots, n_i, \quad t \in I \end{aligned} \tag{5.81}$$

We recognize this as an $n_i \times n_i$ system of ODE

$$M \ddot{\xi}(t) + c^2 A \xi(t) = b(t), \quad t \in I \tag{5.82}$$

where M, A, and b are the usual mass matrix, stiffness matrix, and load vector, respectively.

5.5.5 Time Discretization

Looking at the ODE system (5.82) we see that it is of second order, which is kind of problematic since all our finite difference time stepping schemes are designed to handle first order systems only. The solution is to introduce a new variable $\eta = \dot{\xi}$ and rewrite the second order system as two first order systems. In doing so we end up with

$$M\dot{\xi}(t) = M\eta(t) \tag{5.83}$$

$$M\dot{\eta}(t) + c^2 A\xi(t) = b(t) \tag{5.84}$$

Now, application the Crank-Nicolson method to each of these two systems gives us

$$M\frac{\xi_l - \xi_{l-1}}{k_l} = M\frac{\eta_l + \eta_{l-1}}{2} \tag{5.85}$$

$$M\frac{\eta_l - \eta_{l-1}}{k_l} + c^2 A\frac{\xi_l + \xi_{l-1}}{2} = \frac{b_l + b_{l-1}}{2} \tag{5.86}$$

In block matrix form this can be written more compactly as

$$\begin{bmatrix} M & -\frac{k_l}{2}M \\ \frac{c^2 k_l}{2}A & M \end{bmatrix} \begin{bmatrix} \xi_l \\ \eta_l \end{bmatrix} = \begin{bmatrix} M & \frac{k_l}{2}M \\ -\frac{c^2 k_l}{2}A & M \end{bmatrix} \begin{bmatrix} \xi_{l-1} \\ \eta_{l-1} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{k_l}{2}(b_l + b_{l-1}) \end{bmatrix} \tag{5.87}$$

Here, the starting iterates ξ_0 and η_0 can be chosen either by nodal interpolation of u_0 and v_0, or as their Ritz projections, for example.

The reason for choosing the Crank-Nicolson time stepping method is that it is more accurate than the Euler methods and that it has the property of conserving energy, which loosely speaking means that the computed solution will not get numerically smeared out. Thus, it is a suitable method for the Wave equation.

We summarize the Crank-Nicolson method for solving the Wave equation with the following algorithm.

Algorithm 14 The Crank-Nicolson Method for the Wave Equation

```
1: Create a triangulation $\mathscr{K}$ of $\Omega$ and define the corresponding space of continuous piecewise
   linear functions $V_{h,0}$ hat function basis $\{\varphi_i\}_{i=1}^{n_i}$.
2: Create a time grid $0 = t_0 < t_1 < \cdots < t_L = T$ on the interval $I = (0,T]$ with $L$ time steps
   $k_l = t_l - t_{l-1}$.
3: Choose $\xi_0$ and $\eta_0$.
4: for $l = 1,2,\dots,L$ do
5:     Compute the $n_i \times n_i$ mass and stiffness matrices $M$ and $A$, and the $n_i \times 1$ load vector $b_l$, with
       entries
           $M_{ij} = \int_0^1 \varphi_j \varphi_i \, dx, \quad A_{ij} = \int_0^1 \varphi_j' \varphi_i' \, dx, \quad (b_l)_i = \int_0^1 f(t_l)\varphi_i \, dx$   (5.88)
6:     Solve the linear system
           $\begin{bmatrix} M & -\frac{k_l}{2}M \\ \frac{c^2 k_l}{2}A & M \end{bmatrix} \begin{bmatrix} \xi_l \\ \eta_l \end{bmatrix} = \begin{bmatrix} M & \frac{k_l}{2}M \\ -\frac{c^2 k_l}{2}A & M \end{bmatrix} \begin{bmatrix} \xi_{l-1} \\ \eta_{l-1} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{k_l}{2}(b_l + b_{l-1}) \end{bmatrix}$   (5.89)
7: end for
```

5.6 Stability Estimates

5.6.1 Energy Conservation

In the absence of external forces or damping the solution u to the Wave equation (5.73) is some kind of traveling wave, which move back and forth over the domain eternally, and although the wave may disperse the energy content (i.e., the sum of kinetic and potential energy) of the initial condition is not diminished. This is the content of the next estimate.

Theorem 5.3. *With $f = 0$ the solution u to* (5.73) *satisfies the estimate*

$$\|\dot{u}(\cdot,t)\|^2 + \|\nabla u(\cdot,t)\|^2 = C \tag{5.90}$$

with constant C independent of time.

Proof. *Choosing $v = \dot{u}$ in the variational formulation* (5.77) *we have*

$$0 = \int_\Omega \ddot{u}\dot{u}\,dx + \int_\Omega \nabla u \cdot \nabla \dot{u}\,dx \tag{5.91}$$

$$= \int_\Omega \tfrac{1}{2}\partial_t(\dot{u})^2\,dx + \int_\Omega \tfrac{1}{2}\partial_t(\nabla u)^2\,dx \tag{5.92}$$

$$= \tfrac{1}{2}\partial_t(\|\dot{u}\|^2 + \|\nabla u\|^2) \tag{5.93}$$

Integrating this result with respect to to time t from 0 *to T we have*

$$\|\dot{u}(\cdot,T)\|^2 + \|\nabla u(\cdot,T)\|^2 = \|v_0\|^2 + \|\nabla u_0\|^2 \tag{5.94}$$

The proof ends by noting that the right hand side is independent of time t.

5.7 A Priori Estimate

Theorem 5.4. *The space discrete solution u_h defined by* (5.80) *satisfies the a priori estimate*

$$\|u(t) - u_h(t)\| \leq Ch^2(\|u''(t)\| + \int_0^t \|\ddot{u}(\cdot,s)\|\,ds) \tag{5.95}$$

Proof. *The proof follows by writing the error $u - u_h = u - R_h u + R_h u - u_h$, inserting it into the variational formulation, and using a variant of the stability estimate* (5.90)*. We omit the details.*

5.8 Computer Implementation

A MATLAB code for solving the Wave equation is given below. The specific problem under consideration is $\ddot{u} - \Delta u = 0$ on a square domain with two columns added on one side. The boundary conditions are $u = 0.1\sin(8\pi t)$ on the line segments $x = -0.25$, and $n \cdot \nabla u = 0$ on the rest of the boundary. Thus, we have both Dirichlet and Neumann conditions. Zero initial conditions are assumed. This set up corresponds to a situation where coherent light in the form of a sine wave impinges on a screen with two narrow slits. This creates interference on the other side of the screen. The explanation for this has to do with the distance traveled by the wave from the two slits. As the light passes the screen the waves from the two sources are in phase. However, as we move away from the screen, the path traveled by the light from one slit is larger than that traveled by the light from the other slit. When the difference in path is equal to half a wavelength the waves extinguish each other and the amplitude of their sum vanish. Similarly, when the difference in path length is equal to a wavelength, the waves interact to enhance each other.

```
function WaveSolver2D()
g = Dslit(); % double slit geometry
h = 0.025; % mesh size
k = 0.005; % time step
T = 2; % final time
[p,e,t] = initmesh(g,'hmax',h);
np = size(p,2); % number of nodes
x = p(1,:)'; y = p(2,:)'; % node coordinates
fixed = find(x < -0.24999); % Dirichlet nodes
xi = zeros(np,1); % set zero IC
eta = zeros(np,1);
[A,M,b] = assema(p,t,1,1,0); % assemble A, M, and b
for l = 1:round(T/k) % time loop
  time = l*k;
  LHS = [M -0.5*k*M;  0.5*k*A M]; % Crank-Nicholson
  rhs = [M  0.5*k*M; -0.5*k*A M]*[xi; eta] ...
        + [zeros(np,1); k*b];
  sol = LHS\rhs;
  xi = sol(1:np);
  eta = sol(np+1:end);
  xi(fixed) = 0.1*sin(8*pi*time); % set BC the ugly way
  pdesurf(p,t,xi), axis([-1 1 -1 1 -.5 .5])
  pause(0.1)
end
```

The enforcement of the Dirichlet boundary condition demands some explanation. Since these are time-dependent we have to evaluate and set them inside the time loop at every time step. This can be done by counting the free and the fixed nodes and reducing the linear system resulting from the Crank-Nicholson scheme. However, we use a quick and dirty way instead. At each time step we first apply the Dirichlet boundary conditions to the solution from the previous time step, and then solve the linear system with Neumann boundary conditions. This is simple and works if the time step is small.

The geometry matrix for the double slit domain is given by the routine `Dslit` listed in the Appendix.

In Figure 5.2 we show snapshots of the amplitude of the light wave at a few time steps. The evolution of the interference pattern is clearly seen.

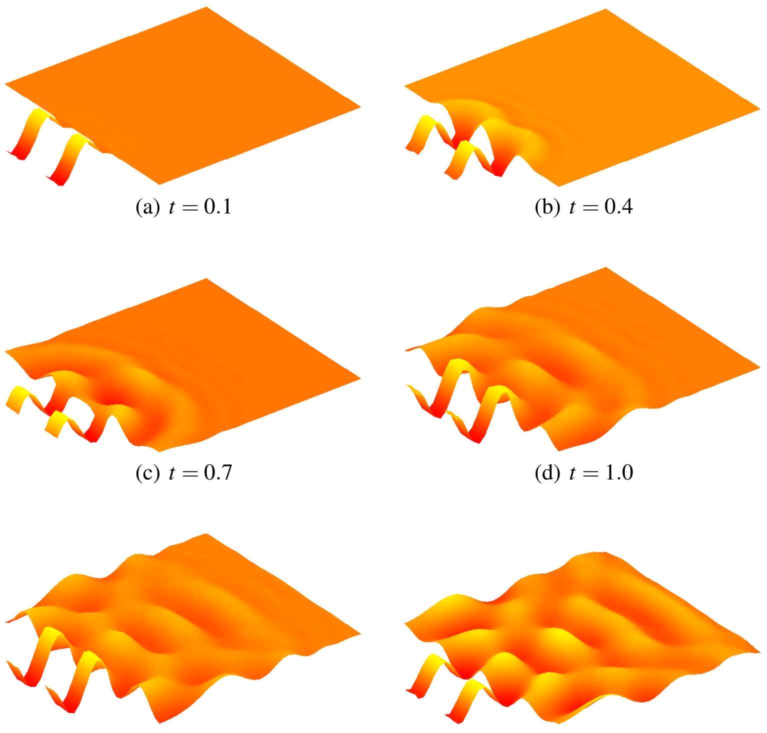

(a) $t = 0.1$ (b) $t = 0.4$ (c) $t = 0.7$ (d) $t = 1.0$ (e) $t = 1.3$ (f) $t = 1.6$

Fig. 5.2 Simulation of light interference with the double slit experiment. Light wave amplitude at a various times.

5.9 Problems

Exercise 5.1. Make two iterations using backward Euler on the ODE system

$$\dot{c}(t) + Ac(t) = f, \quad t > 0, \quad c(0) = c_0,$$

where

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}, \qquad f = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \qquad c_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Assume time step $k = 1/2$.

Exercise 5.2. Show that a space discretization of the problem

$$\begin{aligned} \dot{u} - \Delta u + u = f, &\quad x \in \Omega,\, t > 0 \\ u = 0, &\quad x \in \partial\Omega,\, t > 0 \\ u = u_0, &\quad x \in \Omega,\, t = 0 \end{aligned}$$

leads to a system of ODE of the form

$$M\dot{\xi}(t) + A\xi(t) + M\xi(t) = b(t)$$

Also, identify the entries of the involved matrices and vectors.

Exercise 5.3. Show that the homogenous Heat equation $\dot{u} - \Delta u = 0$, with boundary condition $u = 0$, and intitial condition $u(x,0) = u_0(x)$ obeys the stability estimate

$$\|u(\cdot,t)\| \leq \|u_0\|$$

Interpret this result. *Hint:* Multiply by u and integrate.

Exercise 5.4. Modify `HeatSolver1D` and solve the heat problem

$$\begin{aligned} \dot{u} = u''/10, &\quad 0 < x < 1, \quad t > 0 \\ u(0) = u(1) &= 0 \\ u(x,0) &= x(1-x) \end{aligned}$$

Use a mesh with 100 elements, final time 0.1 and timestep 0.001. Plot the finite element solution at each time step. Compare with the exact solution, given by the infinite sum

$$u(x,t) = \frac{4}{\pi^3} \sum_{n=1}^{\infty} \frac{(-1)^n - 1}{n^2} e^{-n^2\pi^2 t/10} \sin(n\pi x)$$

Trunkate the sum after, say, 25 terms.

Exercise 5.5. Show that the Ritz projector $R_h u$ satisfies the estimate $\|e'\| \leq Ch\|u''\|$, where $e = u - R_h u$.

Hint: Start from $\|\sqrt{a}e'\|^2 = \int_0^1 ae'^2\,dx$ and write $e = u - \pi u - \pi u - R_h u$, where $\pi u \in V_{h,0}$ is the usual node interpolant of u. Then use the definition of the Ritz projector, the following variant of Cauchy-Schwartz inequality $\int_0^1 ae'(u - \pi u)'\,dx \leq \|\sqrt{a}e'\|\|\sqrt{a}(u - \pi u)'\|$, and a standard interpolation estimate.

Chapter 6
Iterative Methods for Large Sparse Linear Systems

Abstract In the previous chapters we have seen how finite element discretization give rise to linear systems, which must be solved in order to obtain the finite element solution. These linear systems are generally very large since they are direct proportional to the number of nodes in the finite element mesh. Recall that is not unusual to have millions of nodes in large meshes. This puts high demands on the linear algebra algorithms and software that is used to solve the linear systems regarding the computational complexity (i.e., number of floating point operations needed), memory requirements, and time consumption. In this context a good thing is that matrices stemming from finite element discretization are generally sparse meaning that they have very few non-zero entries. This is due to the fact that since the hat functions have very limited support they only interact with their nearest neighbors. In this chapter we review the most common iterative methods for solving large sparse linear systems.

6.1 Introduction

6.1.1 Linear Systems

Throughout this chapter we shall consider the problem of solving the linear system of algebraic equations

$$Ax = b \tag{6.1}$$

where A is a given $n \times n$ matrix, b is a given $n \times 1$ vector, and x the sought $n \times 1$ solution vector.

Our basic assumption is that n is large, say, 10^5, and that A is sparse. A sparse matrix is somewhat vaguely defined as one with very few non-zero entries A_{ij}. The prime example of such a matrix is the stiffness matrix resulting from finite element discretization of the Laplace operator $-\Delta$.

We recall that if A is invertible, which by the way is the usual case when the underlying differential equation is well posed, the solution x to (6.1) can formally be found by first computing the inverse A^{-1} to A and then multiply it with b to obtain $x = A^{-1}b$. However, this requires the computation of the $n \times n$ matrix A^{-1} which might seem wasteful since our aim is to find the $n \times 1$ vector x. This is especially true if n is large. Indeed, as we shall see it is almost never necessary to compute any matrix inverse to solve a linear system.

There are two broad classes of solution methods for linear systems, namely:

- Direct methods
- Iterative methods

6.1.2 Direct Methods

Direct methods refers to Gaussian elimination, or, LU factorization, and its variants. The common feature of direct methods are that the solution x is retrieved after a fixed number of floating point operations. Unfortunately, for a linear system with n unknowns this operation count is proportional to n^3, which is way too expensive even for modern supercomputers if n happens to be large. As a result direct methods are not particularly well suited for solving liner systems from finite element applications, and we shall not discuss them further here. Instead we now focus on the other class of iterative solution methods for linear systems.

6.1.3 Iterative Methods

Unlike direct methods, iterative methods do not have a fixed number of floating point operations attached to them for computing the solution x to a linear system. Instead, a solution approximation x^k is sought iteratively, such that $x^k \to x$ in the limit $k \to \infty$. Of course the unspoken hope is that this iteration process will converge and with a small number of iterations k. As we shall see these hopes do certainly not always come true, but when they do iterative methods are cheap, fast, and the preferred choice for solving large sparse linear systems.

6.2 Basic Iterative Methods

It is simple to create a framework for a basic iterative method. Consider again the linear system $Ax = b$.

Let us first split A into

$$A = M - K \tag{6.2}$$

where M is any non-singular matrix and K the remainder $K = M - A$. We then have

$$(M - K)x = b \tag{6.3}$$
$$Mx = Kx + b \tag{6.4}$$
$$x = M^{-1}Kx + M^{-1}b \tag{6.5}$$

If we have a starting guess x^0 for x, this suggests the following iteration scheme:

$$x^{k+1} = M^{-1}Kx^k + M^{-1}b \tag{6.6}$$

Although we do not know for which linear systems this iteration converges, if any, we tacitly summarize it as our basic iterative method for solving linear systems. We shall return to the convergence issue shortly.

Algorithm 15 Basic Iterative Method for a Linear System

```
1: Choose a staring guess $x^0$.
2: for $k = 0, 1, 2$ until convergence do
3:
       $$x^{k+1} = M^{-1}Kx^k + M^{-1}b \tag{6.7}$$
4: end for
```

For this iteration to be computationally practical, it is important that the splitting of A is chosen such that $M^{-1}K$ and $M^{-1}b$ are easy to calculate, or at least their action on any given vector. Recall that we do not want to compute inverses.

In the following we shall study splittings of A of the form

$$A = D - U - L \tag{6.8}$$

where D is the diagonal of A, and $-U$ and $-L$ the strict upper and lower triangular part of A, respectively. This leads to two classical iterative methods, known as the Jacobi and the Gauss-Seidel methods.

6.2.1 *Jacobi's Method*

Jacobi iteration is defined by choosing $M = D$ and $K = L + U$, which gives the iteration scheme

$$x^{k+1} = D^{-1}(L + U)x^k + D^{-1}b \tag{6.9}$$

Here, we note that D is easy to invert since it is a diagonal matrix.

6.2.2 The Gauss-Seidel Method

In the Gauss-Seidel method $M = D - L$ and $K = U$, which gives the iteration scheme

$$x^{k+1} = (D-L)^{-1}(Ux^k + b) \tag{6.10}$$

We note that since $D-L$ is lower triangular the effect of $(D-L)^{-1}$ can be computed by forward elimination.

A naive implementation of the Gauss-Seidel method takes only a few lines of code.

```
D = diag(diag(A)) % diagonal
L = -tril(A,-1) % minus lower triangle
U = -triu(A, 1) % minus upper triangle
for k = 1:maxiter
  y = (D-L)\(U*x + b); % Gauss-Seidel iteration scheme
  x = y
end
```

6.2.3 Convergence Analysis

We now return to the question of convergence of the basic iterative method (6.7). Inspecting it we see that it and all the above methods can be written

$$x^{k+1} = Rx^k + c \tag{6.11}$$

where R is called the iteration matrix and given by $R = M^{-1}K$, and $c = M^{-1}b$.

A relation between the errors in successive approximations can be be derived by subtracting $x = Rx + c$ from (6.11)

$$x^{k+1} - x = R(x^k - x) = \ldots = R^{k+1}(x^0 - x) \tag{6.12}$$

Taking norms and using the Cauchy inequality we have

$$||x^{k+1} - x|| \leq ||R^{k+1}||||x^0 - x|| \leq ||R||^{k+1}||x^0 - x|| \tag{6.13}$$

From this we see that a sufficient condition for convergence is that $||R|| < 1$ in any norm.

Based on the small error analysis above it is clear that $||R||$ should be as small as possible since this is the amplification factor for the error in each iteration. Hence, the splitting of A should be chosen such that:

- $Rx = M^{-1}Kx$ and $x = M^{-1}b$ are easy to evaluate.
- $||R||$ is small.

Unfortunately, these goals are contradictory, and a balance has to be struck. For example,

- $M = I$ makes M^{-1} trivial, but probably not $||A - I|| < 1$.
- $M = A$ gives $K = 0$ and thus $||R|| = ||M^{-1}K|| = 0$, but then $M^{-1} = A^{-1}$ is expensive to compute.

Before stating a general convergence criterion for Jacobi's and the Gauss-Seidel methods let us pause for a moment to introduce the concept of a diagonally dominant matrix.

A square $n \times n$ matrix A is said to be (strictly) diagonally dominant if the absolute value of each diagonal element is greater than the sum of the absolute values of the other elements in its row. That is, if

$$|A_{ii}| > \sum_{j=0, j\neq i}^{n} |A_{ij}|, \quad \forall i = 1, 2, \ldots, n \tag{6.14}$$

For example, the matrix

$$A = \begin{bmatrix} 4 & 1 & 0 \\ -2 & -5 & 1 \\ 6 & 0 & -7 \end{bmatrix} \tag{6.15}$$

is diagonally dominant.

Thus, by now we understand that the success of an iterative method depends on the type of linear system $Ax = b$ it is applied to. More formally we have following convergence criteria.

Theorem 6.1.

- *Jacobi's method converges if A is strictly diagonally dominant.*
- *The Gauss-Seidel method converges if A is symmetric and positive definite (SPD).*

Proof. Let us prove the first part of the theorem as the second part is somewhat technical.

In Jacobi's method the iteration matrix R has the elements

$$R_{i,j} = \frac{A_{ij}}{A_{ii}}, \quad i \neq j, \quad R_{i,i} = 0 \tag{6.16}$$

Taking the infinity norm gives

$$||R||_\infty = \max_{1\leq i\leq n} \sum_{j=1,\ j\neq i}^{n} \frac{|A_{ij}|}{|A_{ii}|} \tag{6.17}$$

which shows that $||R|| < 1$ if A is strictly diagonal dominant, and we are done.

We mention that the Jacobi method sometimes converges even if its convergence criterion is not satisfied. This is just how iterative methods work. They are somewhat unpredictable.

So what if the matrix A is non-singular, but unsymmetric or indefinite? Well, in these cases it is possible to apply the Gauss-Seidel method to the familiar normal equations

$$A^T Ax = A^T b \tag{6.18}$$

Since $A^T A$ is a symmetric and positive definite matrix if A is non-singular the Gauss-Seidel method will converge. However, the rate of convergence can be very slow.

6.3 Projection Methods

The basic iterative methods are cheap but generally slow to converge. To remedy this a great deal of work has been devoted to developing fast iterative methods. This has lead to the development of iterative methods for solving linear systems $Ax = b$, which are based on the requirement that the residual $r = b - Ax$ should be orthogonal to subspaces of $\mathbb{R}^n$, just like the finite element method requires the residual of a partial differential equation to be orthogonal to V_h. In fact, modern iterative methods for linear systems share many features with the Galerkin method. But let us not rush ahead, but start from the beginning. Suppose we seek a solution approximation $\tilde{x}$ to $Ax = b$ from a (small) m-dimensional subspace $\mathscr{K} \subset R^n$, such that the residual

$$r = b - A\tilde{x} \tag{6.19}$$

is orthogonal to another m-dimensional subspace $\mathscr{L} \subset R^n$, that is,

$$b - A\tilde{x} \quad \perp \quad \mathscr{L} \tag{6.20}$$

The subspace $\mathscr{K}$ is called trial space, and the subspace $\mathscr{L}$ is called test space.

There are two classes of projection methods:

- Orthogonal, where $\mathscr{L} = \mathscr{K}$.
- Oblique, where $\mathscr{K}$ and $\mathscr{L}$ are (more or less) unrelated.

As we shall see this distinction gives rise to different types of iterative methods.

If we have a starting guess x_0 for x, then we seek the solution in the affine space

$$x_0 + \mathscr{K} \tag{6.21}$$

instead of just $\mathscr{K}$. That is, we let

$$\tilde{x} = x_0 + \delta$$

where δ is some vector in $\mathscr{K}$.

Our problem is thus to find $\tilde{x} \in x_0 + \mathscr{K}$ such that

$$r = b - A\tilde{x} = b - A(x_0 + \delta) = r_0 - A\delta \quad \perp \quad \mathscr{L}$$

where we have introduced the initial residual $r_0 = b - Ax_0$.

Now, suppose that $V = [v_1, v_2, \dots, v_m]$ and $W = [w_1, w_2, \dots, w_m]$ are two $n \times m$ matrices whose columns $\{v_i\}_{i=1}^m$ and $\{w_i\}_{i=1}^m$ form a basis for $\mathscr{K}$ and $\mathscr{L}$, respectively. Then we can write

$$\tilde{x} = x_0 + \delta = x_0 + Vy \tag{6.22}$$

for some $m \times 1$ vector y to be determined.

Next we note that the orthogonality $r = r_0 - A\delta \perp \mathscr{L}$ means that

$$w^T(r_0 - AVy) = 0, \quad \forall w \in \mathscr{L} \tag{6.23}$$

and since W is a basis for $\mathscr{L}$ this is equivalent to

$$W^T(r_0 - AVy) = 0 \tag{6.24}$$

or

$$W^T AVy = W^T r_0 \tag{6.25}$$

Hence, if the $m \times m$ matrix $W^T AV$ can be inverted then we end up with the expression

$$\tilde{x} = x_0 + Vy = x_0 + V(W^T AV)^{-1} W^T r_0 \tag{6.26}$$

for the approximate solution $\tilde{x}$.

There are two instances when it is guaranteed that $W^T AV$ can be inverted:

- If A is SPD and $\mathscr{L} = \mathscr{K}$.
- If A is non-singular and $\mathscr{L} = A\mathscr{K}$.

We omit the proof of this.

Equation (6.26) is a basic projection step. Most modern methods use a succession of such projections. Typically, a new projection step uses a new pair of subspaces $\mathscr{K}$ and $\mathscr{L}$ with the initial guess x_0 equal to the most recent approximation obtained.

6.3.1 One-dimensional Projection Methods

The simplest choice of trial and test space is to let $\mathscr{K}$ and $\mathscr{L}$ be one-dimensional, that is,

$$\mathscr{K} = \text{ span } \{v\}, \qquad \mathscr{W} = \text{ span } \{w\} \tag{6.27}$$

where v and w are two $n \times 1$ vectors. In this case $\tilde{x}$ is given by

$$\tilde{x} = x_0 + \alpha v \tag{6.28}$$

where the scalar α is given by

$$\alpha = \frac{w^T r_0}{w^T A v} \tag{6.29}$$

A classic choice is to set $v = w = r$ (i.e., $\mathscr{K} = \mathscr{L}$). This yields the Steepest Descent algorithm.

Algorithm 16 Steepest Descent

```
1: Choose a starting guess $x^0$.
2: for $k = 0, 1, 2, \dots$ until convergence do
3:     $r^k = b - Ax^k$
4:     $\alpha = r^{kT} r^k / r^{kT} A r^k$
5:     $x^{k+1} = x^k + \alpha r^k$
6: end for
```

Since $\mathscr{L} = \mathscr{K}$ steepest decent works for cases where A is SPD.

Other choices of v and w include $v = r$ and $w = Ar$, which is the minimal residual method (MINRES).

One-dimensional projection methods are simple, but as one might expect not very efficient.

6.3.2 Krylov Subspaces

The most important iterative methods for sparse linear systems uses projection onto so-called Krylov subspaces. We shall now study these.

The m-th Krylov subspace $\mathscr{K}_m(A; v) \subset \mathbb{R}^n$ is defined by

$$\mathscr{K}_m(A; v) = \text{span}\{v, Av, A^2 v, \dots, A^{m-1} v\} \tag{6.30}$$

where A is a given $n \times n$ matrix and v is a given $n \times 1$ vector. We say that v generates $\mathscr{K}_m$. Often $v = b$.

Let us try to motivate why the Krylov subspaces are defined as they are. Consider a linear system with

$$A = \begin{bmatrix} 5 & 1 \\ 0 & 2 \end{bmatrix}, \qquad b = \begin{bmatrix} 20 \\ 10 \end{bmatrix} \tag{6.31}$$

The characteristic polynomial $p(\lambda)$ of A is given by

$$p(\lambda) = \det(A - \lambda I) = \lambda^2 - 7\lambda + 10 \tag{6.32}$$

Now, according to the Cayley-Hamilton theorem a matrix satisfies its characteristic equation, $p(A) = 0$. That is,

$$0 = A^2 - 7A + 10I \tag{6.33}$$

Multiplying with A^{-1} and rearranging the terms we end up with

$$A^{-1} = \tfrac{7}{10}I - \tfrac{1}{10}A \tag{6.34}$$

Hence, we have

$$x = A^{-1}b = (\tfrac{7}{10}I - \tfrac{1}{10}A)b = \tfrac{7}{10}b - \tfrac{1}{10}Ab = \begin{bmatrix}3\\5\end{bmatrix} \tag{6.35}$$

The key observation here is that the solution x to $Ax = b$ is a linear combination of the vectors b and Ab, which make up the Krylov subspace $\mathscr{K}_2(A,b)$. In other words the solution to $Ax = b$ has a natural representation as a member of a Krylov space, and therefore we can understand why one would construct approximations to x from this space. Of course if the dimension m of $\mathscr{K}_m$ is small, then a Krylov method has the opportunity to find a good approximation x in a few iterations.

Because the Krylov vectors $\{A^j v\}_{j=0}^{m-1}$ tend very quickly to become almost linearly dependent, methods relying on Krylov subspaces frequently involve some orthogonalization procedure. The most general of these is the Arnoldi procedure, which is an algorithm for building an orthonormal basis $\{q_j\}_{j=1}^{m}$ to $\mathscr{K}_m(A,v)$. One variant of the algorithm is given below:

Algorithm 17 Arnoldi's Orthogonalization Procedure

```
 1: Choose a vector $v$ and set $q_1 = v/\|v\|$
 2: **for** $j = 1,2,\dots,m$ **do**
 3:     Compute $z = Aq_j$
 4:     **for** $i = 1,2,\dots,j$ **do**
 5:         $H_{ij} = q_i^T z$
 6:         $z = z - H_{i,j}q_i$
 7:     **end for**
 8:     $H_{j+1j} = \|z\|$
 9:     **if** $H_{j+1j} = 0$ **then**
10:         quit
11:     **end if**
12:     $q_{j+1} = z/H_{j+1j}$
13: **end for**
```

At each step the algorithm multiplies the previous Arnoldi vector q_j by A and then orthonormalizes the resulting vector $z = Aq_j$ against all the previous q_i, $i = 1,2,\dots,j$ by a standard Gram-Schmidt procedure. Inspecting the algorithm, we see that $z = Aq_j$ is a linear combination of the Arnoldi vectors q_i, $i = 1,2,\dots j+1$. The coefficients of this linear combination are the numbers H_{ij}.

The MATLAB realization of the Arnoldi algorithm is given below.

```
function [Q,H] = Arnoldi(A,q,m)
n=size(A,1);
```

```
Q=zeros(n,m+1);
H=zeros(m+1,m);
Q(:,1)=q/norm(q);
for j=1:m
  z=A*Q(:,j);
  for i=1:j
    H(i,j)=dot(z,Q(:,i));
    z=z-H(i,j)*Q(:,i);
  end
  H(j+1,j)=norm(z);
  if H(j+1,j)==0, break, end
  Q(:,j+1)=z/H(j+1,j);
end
```

Loosely speaking the Arnoldi procedure gives a factorization of the matrix A. Indeed, at stage m of the Arnoldi algorithm, it computes the decomposition

$$AQ_m = Q_{m+1}\bar{H}_m \tag{6.36}$$

where

$$Q_m = \begin{bmatrix} \\ q_1\ q_2\ \dots\ q_m \\ \\ \end{bmatrix} \tag{6.37}$$

is the $n \times m$ orthonormal matrix containing the Arnoldi vectors q_i, $i = 1,2,\dots,m$, and where $\bar{H}_m$ is the $(m+1) \times m$ matrix

$$\bar{H}_m = \begin{bmatrix} H_{11} & H_{12} & H_{13} & & & H_{1m} \\ H_{21} & H_{22} & H_{23} & & & H_{2m} \\ 0 & H_{32} & H_{33} & & & H_{3m} \\ 0 & 0 & H_{43} & & & H_{4m} \\ \vdots & & & \ddots & & \vdots \\ 0 & 0 & \dots & 0 & H_{m+1m-1} & H_{m+1m} \end{bmatrix} \tag{6.38}$$

We remark that a $\bar{H}$ is called an upper Hessenberg matrix. By definition such a matrix has all zero entries below the first subdiagonal.

Since the columns of Q_m are orthonormal it is easy to confirm that

$$Q_m^T A Q_m = H_m \tag{6.39}$$

where H_m is the $m \times m$ matrix obtained by deleting the last row from $\bar{H}_m$.

If A is symmetric the Arnoldi algorithm simplifies and is then called the Lanczos algorithm. In this case the Hessenberg matrix H_m reduces to tridiagonal form.

6.3.3 CG

We shall now combine the prototype projection method (6.26) with the Krylov subspace $\mathscr{K}_m(A;v)$ to derive the Conjugate Gradient (CG) algorithm, which is the most famous Krylov method.

Given a linear system $Ax = b$ with a symmetric positive definite system matrix A, and a starting guess x_0 for its solution we now consider a projection method with test and trial space $\mathscr{L} = \mathscr{K} = \mathscr{K}_m(A, r_0)$, where $r_0 = b - Ax_0$ is the initial residual vector. As we have seen this method seeks an approximation x_m to x from the space $x_0 + \mathscr{K}_m$ by imposing the orthogonality condition

$$b - Ax_m \quad \perp \quad \mathscr{K}_m(A; r_0) \tag{6.40}$$

To generate a basis Q_m for the test and trial spaces we can do m steps of the Arnoldi procedure on r_0 with the initial Arnoldi vector chosen as $q_1 = r_0/\|r_0\|$. Substituting $Q_m = V = W$ into the left hand side of (6.26) we have by virtue of (6.39)

$$W^T AV = Q_m^T A Q_m = H_m \tag{6.41}$$

Furthermore, setting $\beta = \|r_0\|$ and substituting $Q_m = W$ into the right hand side of (6.26) we have, since all columns of Q_m except the first are orthogonal against r_0,

$$Q_m^T r_0 = Q_m^T(\beta q_1) = \beta e_1 = \beta[1, 0, \ldots, 0]^T \tag{6.42}$$

Thus, (6.26) reduces to

$$H_m y_m = \beta e_1 \tag{6.43}$$

and, as a result, the approximate solution x_m is given by

$$x_m = x_0 + Q_m y_m = x_0 + Q_m H_m^{-1} \beta e_1 \tag{6.44}$$

The quality of x_m depends on the dimension m of the Krylov space. In practice we would like to be able to improve x_m by choosing m in a dynamic fashion. This line of reasoning leads to the following algorithm called the Full Orthogonalization Method (FOM), and which is the ancestor of, and mathematically equivalent to, the CG algorithm.

Algorithm 18 Full Orthogonalization Method

```
1: Choose a starting guess $x^0$.
2: for $m = 1, 2, 3, ...$ until convergence do
3:     $\beta = \|r_0\|$
4:     Compute $Q_m$ by doing $m$ steps of Arnoldi's procedure.
5:     $y_m = H^{-1}\beta e_1$
6:     $x_m = x_0 + Q_m y_m$
7: end for
```

Let us very swiftly show how FOM can be improved to yield the very elegant CG algorithm. Since we assume that A is symmetric the Hessenberg matrix H_m reduces to tridiagonal form, and the linear system $H_m y_m = \beta e_1$ can therefore be computed efficiently using LU factorization, since both the L and U factors only consists of the diagonal and a sub or superdiagonal, respectively. To this end, let $L_m U_m = H_m$ be the LU factorization of H_m. This gives us

$$x_m = x_0 + Q_m U_m^{-1} L_m^{-1} \beta e_1 = x_0 + P_m z_m \tag{6.45}$$

where we have introduced the $n \times m$ matrix $P_m = Q_m U_m^{-1}$ and the $m \times 1$ vector $z_m = L_m^{-1}\beta e_1$. Now, the difference between the $(m-1) \times (m-1)$ matrix H_{m-1} and the $m \times m$ matrix H_m is the addition of a new last row and column. All other matrix entries are the same. This in turn implies that the difference between L_{m-1}^{-1} and L_m^{-1} is the addition of a last row l_m and a zero column except the diagonal entry which is always unity, viz.

$$L_m^{-1} = \begin{bmatrix} L_{m-1}^{-1} & 0 \\ l_m & 1 \end{bmatrix} \tag{6.46}$$

From this it is easy to see that

$$z_m = \begin{bmatrix} z_{m-1} \\ \zeta_m \end{bmatrix} \tag{6.47}$$

where the $(m-1) \times 1$ vector z_{m-1} stems from the previous iteration and ζ_m is a scalar. As a consequence, we get

$$x_m = x_0 + P_m z_m \tag{6.48}$$

$$= x_0 + \begin{bmatrix} P_{m-1} & p_m \end{bmatrix} \begin{bmatrix} z_{m-1} \\ \zeta_m \end{bmatrix} \tag{6.49}$$

$$= x_0 + P_{m-1} z_{m-1} + \zeta_m p_m \tag{6.50}$$

where p_m is the last column of P_m. Noting that $x_0 + P_{m-1} z_{m-1} = x_{m-1}$ it follows that

$$x_m = x_{m-1} + \zeta_m p_m \tag{6.51}$$

We conclude that, at stage m, x_m is formed simply by updating x_{m-1} by a number, ζ_m, times a search direction (i.e., a gradient) p_m. It turns out that similar updating relations holds for the residuals r_m, and the search directions p_m. Moreover, the search directions are A conjugate in the sense that $p_i^T A p_j = 0$ if $i \neq j$. Hence, the name of the algorithm.

With a little more work it is possible to formulate this projection method as the very compact algorithm, the Conjugate Gradient algorithm.

Algorithm 19 Conjugate Gradient algorithm

```
1: Compute $r_0 = b - Ax_0$ and set $p_0 = r_0$.
2: **for** $j = 0, 1, 2, \ldots$ until convergence **do**
3:     $\alpha_j = r_j^T r_j / p_j A p_j$
4:     $x_{j+1} = x_j + \alpha_j p_j$
5:     $r_{j+1} = r_j - \alpha A p_j$
6:     $\beta_j = r_{j+1}^T r_{j+1} / r_j^T r_j$
7:     $p_{j+1} = r_{j+1} + \beta_j p_j$
8: **end for**
```

This is a very cheap algorithm both regarding computational cost and memory:

- Only one matrix-vector multiplication Ap_j per iteration needed.
- Only requires storage of a few vectors, and not all the m vectors in Q_m.

The rate of convergence of the CG method depends on the condition number $\kappa = |\lambda_{\max}/\lambda_{\min}|$ of A, that is, the quotient of the smallest and largest eigenvalues of A. It is a tedious task to show that the error $e_m = x - x_m$ decreases as

$$\frac{\|e_m\|_A}{\|e_0\|_A} \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^m \tag{6.52}$$

where we have introduced the energy norm $\|v\|_A = \sqrt{v^T A v}$.

The errors are also monotonically decreasing

$$\|e_{m+1}\|_A \leq \|e_m\|_A, \quad m \leq n \tag{6.53}$$

This is due to the fact that the Krylov spaces are nested (i.e., $\mathscr{K}_m \subset \mathscr{K}_{m+1}$).

In Matlab conjugate gradients are implemented as a black box solver

```
x = pcg(A, b, tol, maxit)
```

Input is the system matrix A, the right hand side vector b, a desired tolerance related to the size of the relative residual, and the maximum number of iterations. Output is the solution approximation x_m.

6.3.4 GMRES

Conjugate gradients only works for linear systems with a symmetric positive definite system matrix A. This is somewhat limiting and we shall therefore now study a Krylov method which works for a general linear system without any assumptions on A.

The Generalized Minimum Residual method (GMRES) is a projection method based on taking $\mathscr{K} = \mathscr{K}_m$ and $\mathscr{L} = A\mathscr{K}_m$. This choice of trial and test spaces has the property that the solution x_m minimizes the residual norm $\|r_m\|$ over all vectors in $x_0 + \mathscr{K}_m$. In other words, GMRES is a least squares method. We will exploit this optimality property together with the Arnoldi factorization to derive the GMRES algorithm.

We recall that any vector x in $x_0 + \mathscr{K}_m$ can be written as

$$x = x_0 + Q_m y \tag{6.54}$$

for some $m \times 1$ vector y to be determined. Defining now the least squares functional

$$J(y) = \|b - Ax_m\|^2 = \|b - A(x_0 + Q_m y_m)\|^2 \tag{6.55}$$

the result (6.39) implies

$$\begin{aligned} b - Ax &= b - A(x_0 + Q_m y) && (6.56)\\ &= r_0 - AQ_m y && (6.57)\\ &= \beta q_1 - Q_{m+1}\bar{H}_m y && (6.58)\\ &= Q_{m+1}(\beta e_1 - \bar{H}_m y) && (6.59) \end{aligned}$$

Further, since Q_m is orthonormal we have

$$J(y) = \|b - A(x_0 + Q_m y)\|^2 = \|\beta e_1 - \bar{H}_m y\|^2 \tag{6.60}$$

Now, the GMRES approximation x_m is defined as $x_m = x_0 + Q_m y_m$, where y_m is the minimizer of $J(y)$ over $x_0 + \mathscr{K}_m$. This minimizer is inexpensive to compute since it only requires the solution of a $(m+1) \times m$ linear least squares problem with m typically small. Standard methods from dense linear algebra (i.e., QR factorization) are used to do this.

Algorithm 20 Generalized Minimum Residual (GMRES)

```
1: Compute $r_0 = b - Ax_0$ and set $q_1 = r_0/\|r_0\|$.
2: for $m = 1, 2, \dots$ until convergence do
3:     Compute $Q_m$ with Arnoldi.
4:     Compute $y_m$, the minimizer of $J(y) = \|\beta e_1 - \bar{H}_m y\|$.
5:     Set $x_m = x_0 + Q_m y_m$.
6: end for
```

In GMRES the computational cost per iteration is not fixed but increases because all the m Arnoldi vectors q_j are required for computing $\bar{H}_m$. The memory cost is therefore $\mathscr{O}(mn)$. This cost can limit the largest affordable value of m for large n. One remedy is to restart the algorithm periodically with the latest solution approximation as starting guess.

6.3.5 Other Krylov Methods

The is a plethora of Krylov methods. Below we characterize a few of these.

- CG on the Normal Equations (CGNE)
  - Solve $A^TAx = A^Tb$ using conjugate gradients.
  - Matrix A need not be SPD.
  - Poor convergence, squared condition number $\kappa(A^TA) = \kappa(A)^2$.
- Bi-Conjugate Gradients (BiCG)
  - Makes residuals orthogonal to another Krylov subspace, based on A^T.
  - Memory requirements are small.
  - Convergence sometimes comparable to GMRES, but unpredictable.
- Conjugate Gradients Squared (CGS)
  - Avoids multiplication by A^T, sometimes twice as fast convergence as BiCG.

6.4 Preconditioning

The convergence rate of all Krylov methods depend on the condition number κ of the matrix A. To accelerate convergence it is customary to try to transform the linear system $Ax = b$ into one that has the same solution, but a more favorable (i.e., smaller) condition number. This is accomplished through so-called preconditioning. A preconditioner M is a matrix that approximates A in some sense, but is more easy to invert.

Multiplying by M^{-1} from the left we have the transformed linear system

$$M^{-1}Ax = M^{-1}b \tag{6.61}$$

which has the same solution as $Ax = b$, but the condition number of the matrix $M^{-1}A$ may be better. If M is a good preconditioner, then $M^{-1}A \approx I$ with a condition number close to 1. Note that $M = I$ is a useless preconditioner, while $M = A$ is the most expensive, since it imples inverting A. Hence, we seek a middle route in constructing a good preconditioner.

6.4.1 Jacobi Preconditioning

The simplest preconditioner consists of just the diagonal of the matrix A, that is, $M = \text{diag}(A)$. This is known as Jacobi preconditioning and can sucessfully be applied to linear systems with A SPD. The Jacobi preconditioner need very little storage, and is easy to implement. On the other hand, more sophisticated preconditioners usually yield a faster rate of convergence.

6.4.2 Incomplete Factorizations

Many modern preconditioners are based on incomplete LU factorization (ILU). The basic idea is simple. One computes the ordinary LU factorization $A = LU$, but in doing so entries of L and U that are too small are discarded to save memory. The preconditioner M is then defined by $M = LU$. This type of preconditioning has proven to be very efficient and is commonly used in combination with CG or GMRES to solve linear systems arising form finite element discretizations. The difficulty is to choose a good drop tolerance, that is, the level below which the matrix entries of M are ignored. A high drop tolerance yields a dense M, while a low drop tolerance might make M inefficient.

Matlab has a built-in function called `luinc` for computing the incomplete LU factorization with a user defined drop tolerance. To solve a linear system with ILU preconditioning and GMRES we type

```
[L,U] = luinc(A,1.e-3) % drop tolerance = 0.001
x = gmres(A,b,[],tol,m,L,U)
```

6.5 A Note on Iterative Methods for Eigenvalue Problems

We end this chaper by briefly describe how the Arnoldi algorithm can also be used to find a few eigenvalues of a large sparse $n \times n$ matrix A.

Recall that the Hessenberg matrix H_m and the orthonormal matrix Q_m is computed at stage m of Arnoldi's procedure. Due to the fact that H_m is a projection, that is, approximation, of A onto the Krylov space spanned by the columns of Q_m, a natural idea is to use the eigenvalues of H_m to approximate the eigenvalues of A. Thus, at each step m, or at occasional steps, the eigenvalues of H_m are computed by standard methods such as QR iteration. These are the Ritz values. Since m is much smaller then n for feasible computations, one cannot expect to compute all the eigenvalues of A by this process. Typically, it finds the extreme eigenvalues with either largest or smallest magnitude. This line of reasoning gives the following simple algorithm called Arnoldi iteration.

Algorithm 21 Arnoldi Iteration

```
1: for $m = 1, 2, \ldots$ until convergence do
2:     Compute $H_m$ with Arnoldi.
3:     Compute the $m$ eigenvalues $\theta_i$, $i = 1, \ldots, m$, the Ritz values, to $H_m$.
4:     Use the Ritz values as approximations to the $m$ largest eigenvalues $\lambda_i$ of $A$.
5: end for
```

Of course, this is only a basic sketch of the algorithm. The practical implementation is very elaborate.

Let $V_m^T \Theta_m V_m = H_m$ with $\Theta_m = \mathrm{diag}([\theta_1, \ldots, \theta_m])$ be the eigendecomposition of H_m obtained from the QR iteration. Approximations to the eigenvectors m of A corresponding to the m largest eigenvalues are given by the columns of $Q_m V_m$. These are the Ritz vectors.

Arnoldi iteration is particularly efficient when the matrix A is symmetric, since this simplifies the Arnoldi algorithm, makes H_m tridiagonal, and allows error estimates for the Ritz values and vectors to be rigoruosly proved. For a deeper discussion of iterative methods for large sparse eigenvalue problems we refer the reader to any textbook on sparse linear algebra.

6.6 Problems

Exercise 6.1. Write two Matlab routines `Jacobi.m` and `GS.m` implementing Jacobi and Gauss-Seidel iteration. Let the syntax for calling the routines be given by

```
[x,k] = Jacobi(A,b,tol)
[x,k] = GS(A,b,tol)
```

where `tol` is a number specifying the desired relative residual $\|r^k\|/\|r^0\|$, and `k` is the number of iterations k performed. Test your codes by solving the linear system with

$$A = \begin{bmatrix} 12 & 1 & 0 & 0 & 0 & -1 \\ 1 & 10 & 1 & 0 & 0 & 0 \\ 2 & 0 & 20 & 2 & 0 & 0 \\ 0 & 0 & 1 & 12 & -1 & 0 \\ 0 & 3 & 0 & 0 & 30 & 3 \\ 0 & 0 & 0 & 2 & -2 & 24 \end{bmatrix}, \qquad b = \begin{bmatrix} 8 \\ 24 \\ 70 \\ 46 \\ 174 \\ 142 \end{bmatrix}$$

Exercise 6.2. Use `Jacobi.m` and `GS.m` to compare the number of iterations required by these methods to converge to a given accuracy from a zero starting guess. Let A and b be defined by

```
e = ones(n,1);
A = spdiags([-e 2*e -e], -1:1, n, n);
b = rand(n,1);
```

Record the number of iterations needed to achieve the tolerance 0.1, 0.01, 0.001, and 0.0001 for a few different values of n, say 10, and 100. How many times faster is Gauss-Seidel than Jacobi?

Exercise 6.3. Show that Jacobi iteration may take the form

$$x^{k+1} = x^k + Hr^k$$

where H is a matrix to be defined by you and $r^k = b - Ax^k$ is the residual at stage k. Can you interpret this result from the point of view of one-dimensional projection methods for $Ax = b$.

Exercise 6.4. Consider the m-th Krylov space $\mathscr{K}_m(A;b)$, and the corresponding Krylov matrix

$$K_m = \left[b \; Ab \; A^2b \; \ldots \; A^{m-1}b\right]$$

Let `A=diag([1 2 3 4])` and `b=[1 1 1 1]'`.

(a) Compute the Krylov matrix K_4. Then express the vector $x = A^{-1}b = [1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}]^T$ as a linear combination of the columns of K_4.
(b) Use the routine `arnoldi.m` to compute the 4×4 matrices Q and H in the Arnoldi factorization of A, (i.e., such that $AQ = QH$). Use $q_1 = b/\|b\|$ as starting vector. (Note that since the Arnoldi algorithm stops at stage 3, the last column of H is not actually computed. It comes from a final command `H(:,4) = Q'*A*Q(:,4)`.)
(c) Assume that we have run Arnoldi's algorithm for 2 steps so that we have access to the orthogonal basis $Q_2 = [q_1, q_2]$ that span the Krylov subspace $\mathscr{K}_2(A;b)$. Show how the matrix H_2 can be used to get a Galerkin solution x_2, that is, such that the residual $r_2 = b - Ax_2$ is orthogonal to the span of the basis vectors q_1 and q_2. Compute x_2. What is the residual r_2?

Chapter 7
Abstract Finite Element Analysis

Abstract In this chapter we study the mathematical theory of finite element methods from a broader perspective by introducing a general theory for linear second order elliptic partial differential equations. This allows us to handle a large class of problems with the same analytical techniques. We do this by first introducing a general elliptic problem and its abstract weak form posed on a so-called Hilbert space. We show that this weak problem has a solution by proving the Lax-Milgram Lemma, and that this solution is unique. Knowing that the solution exist we then show how to approximate it by finite elements. Finally, we prove basic a priori and a posteriori error estimates for the finite element approximation.

7.1 Elliptic Problems

Let Ω be a simply connected bounded domain in $\mathbb{R}^d$, with $d = 2$ the number of space dimensions, and with smooth boundary $\partial\Omega$. Although the analysis to be presented is very generell and holds without changes also for $d = 1$ or 3 let us stick to two dimensions for simplicity.

We shall study partial differential equations of the form

$$Lu = f, \quad \text{in } \Omega \tag{7.1a}$$

$$u = 0, \quad \text{on } \partial\Omega \tag{7.1b}$$

where L is the linear second order differential operator

$$Lu = \sum_{i,j=1}^{d} -\frac{\partial}{\partial x_i}\left(a_{ij}\frac{\partial u}{\partial x_j}\right) + cu \tag{7.2}$$

with a_{ij}, $i, j = 1, \ldots, d$, and c are given coefficients depending only on the space coordinates x_i. We shall assume that there is a value a_0 such that $a_{ij} > a_0$ for all $i, j = 1, \ldots, d$, and that these coefficients are symmetric in the sense that $a_{ij} = a_{ji}$.

Further, we also assume that $c \geq 0$. This kind of partial differential equation is a called elliptic.

Using vector notation (7.2) can be written

$$Lu = -\nabla \cdot (a\nabla u) + cu \tag{7.3}$$

where a is a $d \times d$ matrix with the coefficients a_{ij} as entries.

The class of elliptic equations is large and includes many important equations. For example:

- The Laplace equation $\Delta u = 0$.
- The Poisson equation $-\nabla \cdot (a\nabla u) = f$.
- The Diffusion-Reaction equation $-\Delta u + cu = f$.

We shall now present a general theory to handle all these equations using the same analytical tools.

7.2 Abstract Weak Form

Multiplying $Lu = f$ by a test function v in a suitable space V that satisfies the zero boundary conditions and integrating by parts we obtain the abstract weak form of (7.1): find $u \in V$ such that

$$a(u,v) = l(v), \quad v \in V \tag{7.4}$$

where, with a slight abuse of notation,

$$a(u,v) = (Lu,v) \tag{7.5}$$

$$l(v) = (f,v) \tag{7.6}$$

The function space V on which the abstract weak form (7.4) is posed on is generally called a Hilbert space. Hilbert spaces are linear spaces characterized by the fact that they have a scalar product and a norm, so that it is possible to measure the angle between two functions and the size of a function in these spaces. Hilbert spaces are also complete, which means that every Cauchy sequence in them converges. From this on we shall let V denote a Hilbert space.

The left hand side $l(v)$ in (7.4) is a linear form. A linear form $l(\cdot)$ is a mapping $V \to \mathbb{R}$ such that for any $u, v \in V$

1. $l(u+v) = l(u) + l(v)$.
2. $l(\alpha v) = \alpha l(v)$, $\alpha \in \mathbb{R}$.

Similarly, the left hand side $a(u,v)$ in (7.4) is called a bilinear form. A bilinear form $a(\cdot,\cdot)$ is a mapping $V \times V \to \mathbb{R}$ such that for any $u, v \in V$

1. $a(u+v,w) = a(u,w) + (v,w)$.

2. $(\alpha u, v) = \alpha(u, v)$, $\alpha \in \mathbb{R}$.

In the following $l(\cdot)$ and $a(\cdot,\cdot)$ shall always denote a linear and bilinear form, respectively.

A bilinear form is said to be symmetric if $a(u,v) = a(v,u)$. If also

$$a(u,u) \geq 0 \tag{7.7}$$

with equality if and only if $u = 0$, then $a(\cdot,\cdot)$ defines a scalar product on V.

Given a scalar product $a(\cdot,\cdot)$ on V we can easily define an associated norm $\|\cdot\|_V$ on V by

$$\|u\|_V = \sqrt{a(u,u)} \tag{7.8}$$

This norm is called the energy norm.

For the energy norm holds the ubiquitous Cauchy-Schwartz inequality

$$|a(u,v)| \leq \|u\|_V \|v\|_V \tag{7.9}$$

the Triangle inequality

$$\|u+v\|_V \leq \|u\|_V + \|v\|_V \tag{7.10}$$

and the Parallelogram law

$$\|u+v\|_V^2 + \|u-v\|_V^2 \leq 2(\|u\|_V^2 + \|v\|_V^2) \tag{7.11}$$

7.2.1 Three Common Hilbert Spaces

There are three Hilbert spaces that frequently occur when dealing with weak forms of partial differential equations. First there is the familiar space of square integrable functions $L^2 = L^2(\Omega)$, defined by

$$L^2(\Omega) = \{v : \int_\Omega v^2\, dx < \infty\} \tag{7.12}$$

The functions in L^2 are regular in the sense that they can be squared and still have a bounded integral. However, this is generally not the case for their derivatives. This is bad news since the weak form of a partial differential equation usually requires integration of derivatives of either the test function v or the trial function u, or both. Thus, we need spaces in which both a function and its derivatives are bounded. This leads us to introduce the Hilbert space $H^1 = H^1(\Omega)$, defined by

$$H^1(\Omega) = \{v : v \in L^2,\ \partial v/\partial x_i \in L^2,\ i = 1,\dots,d\} \tag{7.13}$$

or equivalently

$$H^1 = \{v : \|\nabla v\| + \|v\| < \infty\} \tag{7.14}$$

We see that the functions in H^1 are those in L^2 that also have all their partial derivatives in L^2. Thus, we have the inclusion $H^1 \subset L^2$. In other words L^2 contains more functions and is a bigger space than H^1. For us H^1 or some variant of it will be the usual space to pose our abstract weak form on.

As said before, a Hilbert space have a scalar product and norm. For L^2 the scalar product $(u,v)_{L^2(\Omega)}$ is the usual integral

$$(u,v)_{L^2(\Omega)} = \int_\Omega uv\,dx, \quad u,v \in L^2(\Omega) \tag{7.15}$$

and the associated norm $\|\cdot\|_{L^2(\Omega)}$ is defined by

$$\|u\|_{L^2(\Omega)} = \sqrt{(u,u)_{L^2(\Omega)}}, \quad u \in L^2(\Omega) \tag{7.16}$$

As is customary we shall often omit the subscript and write simply $(\cdot,\cdot)$ and $\|\cdot\|$ to denote the L^2 scalar product and norm, respectively.

The scalar product and norm on H^1 is defined by

$$(u,v)_{H^1(\Omega)} = (\nabla u, \nabla v) + (u,v), \quad u,v \in H^1(\Omega) \tag{7.17}$$

$$\|u\|_{H^1(\Omega)} = (\|\nabla u\|^2 + \|u\|^2)^{1/2}, \quad u \in H^1(\Omega) \tag{7.18}$$

Note that the H^1 norm contains both $\|\nabla v\|$ and $\|v\|$ which is necessary to assert that $\|v\|_{H^1} = 0$ if, and only if, $v = 0$. If we for some reason would try to use only $\|\nabla v\|$ as norm on H^1 then we would get the so-called semi-norm, which has all the characteristics of a real norm except for the fact that it is not only zero for the zero function $v = 0$. To see this just think of $v = C$ with C a constant. The H^1 semi-norm is denoted $|v|_1 = \|\nabla v\|$. However, there is is one exception to this. On the subspace

$$H_0^1 = \{v \in H^1,\ v|_{\partial\Omega} = 0\} \tag{7.19}$$

the semi-norm $|v|_1$ actually defines a norm. This has to do with the fact that the only constant function in this subspace is the zero function.

On H_0^1 holds the useful Poincaré inequality

$$\|v\| \le C|v|_1 \tag{7.20}$$

In fact this inequality holds on any subspace of H^1 as long as v is zero on some part of the boundary.

The reason for introducing H^1 was to gain more control over the derivatives of L^2 functions. Even so it turns out that H^1 contains many highly irregular functions. In fact a H^1 function need not have well defined point values in two and three dimensions. This is a discovery with far reaching implications as it forces us to redefine what we mean by a derivative. Recall that to define the derivative of a function it is necessary to evaluate the function at certain points. However, we shall not dwell on this matter. Suffice it to say that there exist a concept called weak derivatives, which can be used to give the derivative of a H^1 function a precise meaning. The basic idea is to interpret the derivatives in a distributional sense.

If the boundary $\partial\Omega$ is smooth or polygonal, then the following trace inequality holds

$$\|v\|_{L^2(\partial\Omega)} \leq C\|v\|_{H^1(\Omega)}, \quad v \in H^1(\Omega) \tag{7.21}$$

Conversely, the functions on $\partial\Omega$ that can be extended as H^1 functions into Ω is denoted by $H^{1/2}(\partial\Omega)$.

7.3 Equivalent Minimization Problem

A key observation is that the abstract weak form (7.4) can be interpreted as the minimization problem: find $u \in v$ such that

$$F(u) = \min_{v\in V} F(v) \tag{7.22}$$

where the functional $F(v)$ is given by

$$F(v) = \tfrac{1}{2}a(v,v) - l(v) \tag{7.23}$$

We shall now actually prove that the abstract weak problem is equivalent to the above minimization problem. We begin by showing that the minimization problem implies the weak problem. To this end suppose that u solves the minimization problem (7.22) and consider the auxiliary function

$$g(\varepsilon) = F(u+\varepsilon v) \tag{7.24}$$

for a fixed, but arbitrary, function $v \in V$. Note that $g(\varepsilon)$ is a scalar function of the single variable ε, and attains its minimum for $\varepsilon = 0$. Obviously, this means that $g'(0) = 0$. Expanding $g(\varepsilon) = F(u+\varepsilon v)$ and using the symmetry $a(u,v) = a(v,u)$ we have

$$\begin{aligned} g(\varepsilon) &= F(u+\varepsilon v) && (7.25)\\ &= \tfrac{1}{2}a(u+\varepsilon v, u+\varepsilon v) - l(u+\varepsilon v) && (7.26)\\ &= \tfrac{1}{2}(a(u,u) + 2\varepsilon a(u,v) + \varepsilon^2 a(v,v)) - l(u) - \varepsilon l(v) && (7.27)\end{aligned}$$

Differentiating this result with respect to ε we obtain

$$g'(\varepsilon) = a(u,v)) - \varepsilon a(v,v) - l(v) \tag{7.28}$$

which gives

$$g'(0) = a(u,v)) - l(v) = 0 \tag{7.29}$$

for each $v \in V$. This is exactly the abstract weak problem.

Let us now instead suppose that u solves the weak form (7.4) and show that this implies the minimization problem. To do so we observe that for any $w \in V$ we have

$$\begin{aligned} F(u+w) &= \tfrac{1}{2}a(u+w,u+w) - l(u+w) && (7.30)\\ &= \tfrac{1}{2}(a(u,u) + 2a(u,w) + a(w,w)) - l(u) - l(w) && (7.31)\\ &= F(u) + a(u,w) - l(w) + \tfrac{1}{2}a(w,w) && (7.32)\\ &= F(u) + \tfrac{1}{2}a(w,w) && (7.33)\\ &\geq F(u) && (7.34)\end{aligned}$$

where we have used that $a(u,w) = l(w)$. From this we conclude that $F(u+w)$ attains its minimum value for $w = 0$. This proves the claim.

7.4 The Lax-Milgram Lemma

Next we prove that the solution to the minimization problem (7.22) exist and is unique. This result is known as the Lax-Milgram Lemma and is one of the most important theorems in applied mathematics. However, for the Lax-Milgram Lemma to hold there are a couple of requirements that must be satisfied. First, the linear form $l(\cdot)$ and the bilinear form $a(\cdot,\cdot)$ must be continuous on the space V. That is, there must exist constants C_1 and C_2 such that

$$|l(v)| \leq C_1 \|v\|_V \tag{7.35}$$

$$|a(u,v)| \leq C_2 \|u\|_V \|v\|_V \tag{7.36}$$

Second, the bilinear form $a(\cdot,\cdot)$ must be coercive, which means that there must exist a constant $\alpha > 0$ such that

$$a(u,u) \geq \alpha \|u\|_V^2 \tag{7.37}$$

Third, we require $a(\cdot,\cdot)$ to be symmetric. Under these three assumptions we then have the following lemma.

Theorem 7.1 (Lax-Milgram Lemma). *Let $a(\cdot,\cdot)$ be a continues coercive bilinear form, and let $l(\cdot)$ be a continuous linear form on the Hilbert space V. Then there exist a solution $u \in V$ to the minimization problem*

$$F(u) = \min_{v \in V} F(v) \tag{7.38}$$

where $F(v) = \frac{1}{2}a(v,v) - l(v)$.

Proof. The idea is to show first that F is bounded from below. Otherwise there will not exist a minimum. Second, we take a sequence $\{u_i\}$ in V such that $F(u_i) \to \min F(u)$. Third, we then show that this sequence is a Cauchy sequence, and thus that it converges since V is a Hilbert space. That is, there exist a $u \in V$ such that $u_i \to u$ when $i \to \infty$. Finally, the continuity of F implies that $\lim_{i\to\infty} F(u_i) = F(u)$. But let us begin from the beginning.

The functional $F(\cdot)$ is bounded from below, since from the coercivity of $a(\cdot,\cdot)$ and the continuity of $l(\cdot)$ it follows that

$$F(v) = \tfrac{1}{2}a(v,v) - l(v) \geq \tfrac{1}{2}\alpha\|v\|_V^2 - c\|v\|_V \tag{7.39}$$

This is clearly a quadratic function of the variable v, and the minimum is attained for $v = -\frac{1}{2}C^2/\alpha$. Therefore we conclude that there is a minimum value β such that

$$\beta = \min_{v\in V} F(v) \tag{7.40}$$

We can now choose a sequence $\{u_i\}_1^\infty$ in V such that

$$F(u_i) \to \beta, \tag{7.41}$$

as $i \to \infty$. Further, this sequence is in fact a Cauchy sequence since due to the coercivity of $a(\cdot,\cdot)$ and the parallelogram law we have

$$\begin{aligned}
\alpha\|u_i - u_j\|_V^2 &\leq a(u_i - u_j, u_i - u_j) &(7.42)\\
&= 2a(u_i,u_i) + 2a(u_j,u_j) - 4a(\frac{u_i+u_j}{2}, \frac{u_i+u_j}{2}) &(7.43)\\
&= 4(F(u_i)+l(u_i)) + 4(F(u_j)+l(u_j)) - 8(F(\frac{u_i+u_j}{2}) - l(\frac{u_i+u_j}{2})) &(7.44)\\
&= 4F(u_i) + 4F(u_j) - 8F(\frac{u_i+u_j}{2}) + 4l(u_i) + 4l(u_j) - 8l(\frac{u_i+u_j}{2}) &(7.45)\\
&\leq 4F(u_i) + 4F(u_j) - 8\beta &(7.46)
\end{aligned}$$

But, both $F(u_i)$ and $F(u_j)$ tend to β as i and j tend to infinity, which implies that $\|u_i - u_j\| \to 0$ as $i, j \to \infty$. We thus conclude that that $\{u_i\}_1^\infty$ is a Cauchy sequence. Now, sine V is a Hilbert space it follows by definition that every Cauchy sequence has a limit in V and thus that there exist a $u \in V$ such that $u_i \to u$ as $i \to \infty$.

We must finally show that the found limit u is the minimizer of F. We know that $u_i \to u$ and that $F(u_i) \to \beta$ as $i \to \infty$. Because F is continuous it follows that

$$\lim_{i\to\infty} F(u_i) = F(\lim_{i\to\infty} u_i) = F(u) = \beta \tag{7.47}$$

and the proof is complete.

We remark that there exist also a variant of the Lax-Milgram Lemma for problems with a non-symmetric bilinear form.

The Lax-Milgram lemma asserts that the minimization problem, and thus also the abstract weak problem, has a solution. We next show that this solution is unique.

Theorem 7.2. *If $a(\cdot,\cdot)$ is a continuous coercive bilinear form, and $l(\cdot)$ a continuous linear form on the Hilbert space V, then the abstract weak problem: find $u \in V$ such that*

$$a(u,v) = l(v), \quad \forall v \in V \tag{7.48}$$

has a unique solution u.

Proof. The proof is by contradiction. Suppose there are two solutions $u_1 \in V$ and $u_2 \in V$ satisfying

$$a(u_1,v) = l(v), \quad \forall v \in V \tag{7.49}$$

$$a(u_2,v) = l(v), \quad \forall v \in V \tag{7.50}$$

Subtraction of the equations yields

$$a(u_1 - u_2, v) = 0, \quad \forall v \in V \tag{7.51}$$

Now, choosing $v = u_1 - u_2$ we have

$$a(u_1 - u_2, u_1 - u_2) = 0, \quad \forall v \in V \tag{7.52}$$

Using the coercivity of $a(\cdot,\cdot)$ we find that

$$a(u_1 - u_2, u_1 - u_2) \geq \alpha \|u_1 - u_2\|_V^2 = 0 \tag{7.53}$$

Thus, $\|u_1 - u_2\|_V^2 = 0$, and hence $u_1 = u_2$. We are done.

Let us demonstrate the usability of the Lax-Milgram Lemma by working trough some examples.

Let us first revisit Poisson's equation

$$-\Delta u = f, \quad x \in \Omega, \qquad u = 0, \quad x \in \partial\Omega \tag{7.54}$$

As we have seen the bilinear and linear forms of this equation are given by

$$a(u,v) = (\nabla u, \nabla v) \tag{7.55}$$

$$l(v) = (f, v) \tag{7.56}$$

and the appropriate Hilbert space is $V = H_0^1$ with norm $\|v\|_V = \|v\|_{H_0^1} = \|\nabla v\|$. To show that the weak form of this equation has a unique solution we must show that $a(\cdot,\cdot)$ is continuous and coercive, and that $l(\cdot)$ is continuous on V. The continuity and coercivity of $a(\cdot,\cdot)$ follows from the Cauchy-Schwartz inequality, since we have

$$a(u,v) = (\nabla u, \nabla v) \leq \|\nabla u\| \|\nabla v\| \leq \|u\|_V \|v\|_V \tag{7.57}$$

and

$$a(u,u) = (\nabla u, \nabla u) = \|\nabla u\|^2 \geq \alpha \|u\|_V^2 \tag{7.58}$$

with $\alpha = 1$. The continuity of $l(\cdot)$ follows from the Cauchy-Schwartz inequality again and the Poincaré inequality, which hold on H_0^1, since both u and v are zero on the boundary. We have

$$l(v) = (f,v) \leq \|f\| \|v\| \leq C\|f\| \|\nabla v\| \leq C\|f\| \|v\|_V \leq C\|v\|_V \tag{7.59}$$

where we have absorbed the norm of f into the constant C in the last line. This shows why it is natural to demand $f \in L^2$, since otherwise the norm $\|f\|$ might not be well defined. Based on these findings we thus conclude the the requirements for the Lax-Milgram Lemma are satisfied and that there exist a solution to the weak form.

As a second example we consider the problem

$$-\Delta u + cu = f, \quad x \in \Omega, \qquad n \cdot \nabla u = 0, \quad x \in \partial\Omega \tag{7.60}$$

where $c \in L^2$ is a given positive function with minimum value $c_0 > 0$ on Ω, and $f \in L^2$ a given function. The bilinear and linear forms of this equation are given by

$$a(u,v) = (\nabla u, \nabla v) + (cu, v) \tag{7.61}$$

$$l(v) = (f,v) \tag{7.62}$$

and the appropriate Hilbert space is $V = H^1$ because of the boundary conditions on the normal derivative. We recall that the H^1 norm is given by $\|v\|_{H^1}^2 = \|v\|_V^2 = \|\nabla v\|^2 + \|v\|^2$. To show that the requirements for the Lax-Milgram Lemma are fulfilled in this case we make repeated use of the Cauchy-Schwartz inequality. The coercivity can be established in the following way

$$a(u,u) = (\nabla u, \nabla u) + (cu, u) \tag{7.63}$$

$$\geq \|\nabla u\|^2 + c_0 \|u\|^2 \tag{7.64}$$

$$\geq \min(1, c_0)(\|\nabla u\|^2 + \|u\|^2) \tag{7.65}$$

$$\geq \alpha \|u\|_V^2 \tag{7.66}$$

with the coercivity constant $\alpha = \min(1, c_0)$. The continuity of $a(\cdot,\cdot)$ follows from the Cauchy-Schwartz inequality.

$$a(u,v) = (\nabla u, \nabla v) + (cu, v) \tag{7.67}$$
$$\leq \|\nabla u\|\|\nabla v\| + \|c\|\|u\|\|v\| \tag{7.68}$$
$$\leq C(\|\nabla u\|\|\nabla v\| + \|u\|\|v\|) \tag{7.69}$$
$$\leq C(\|\nabla u\|^2 + \|u\|^2)^{1/2}(\|\nabla v\|^2 + \|v\|^2)^{1/2} \tag{7.70}$$
$$\leq C\|u\|_V\|v\|_V \tag{7.71}$$

The continuity of $l(\cdot)$ is shown in a similar manner.

$$l(v) = (f,v) \leq \|f\|\|v\| \leq C\|f\|\|\nabla v\| \leq C\|f\|(\|\nabla v\| + \|v\|) \leq C\|v\|_V \tag{7.72}$$

As our final example we consider

$$-\Delta u = 0, \quad x \in \Omega, \qquad u = 0, \quad x \in \Gamma_D, \qquad n \cdot \nabla u = g_N, \quad x \in \Gamma_N \tag{7.73}$$

where $g_N \in L^2(\Gamma_N)$ is a given function, and Γ_D and Γ_N are two disjoint segments of the boundary associated with the Dirichlet and Neumann boundary conditions, respectively. The bilinear and linear forms of this equation are given by

$$a(u,v) = (\nabla u, \nabla v) \tag{7.74}$$
$$l(v) = (g_N, v)_{\Gamma_N} \tag{7.75}$$

Due to the boundary conditions the Hilbert space on which the weak form is posed is given by $V = \{v \in H^1 : v|_{\Gamma_D} = 0\}$ with norm $\|v\|_V = \|\nabla v\|$. The coercivity and continuity of $a(\cdot,\cdot)$ is easy to establish. However, the continuity of $l(\cdot)$ requires us to estimate the norm of v on the boundary segment Γ_N. To do so, we use the trace inequality (7.21), which yields

$$l(v) = (g_N, v)_{\Gamma_N} \leq \|g_N\|_{\Gamma_N}\|v\|_{\Gamma_N} \leq C\|g_N\|_{\Gamma_N}\|v\|_V \leq C\|v\|_V \tag{7.76}$$

which shows that $l(\cdot)$ is continuous.

7.5 Abstract Finite Element Approximation

7.5.1 Abstract Finite Element Method

From the Lax-Milgram Lemma we know that the solution u to the abstract weak problem (7.4) exist and is unique. We can now approximate it using finite elements. To this end let $V_h \subset V$ be a finite dimensional subspace of V typically consisting of continuous piecewise linear polynomials on a mesh $\mathscr{K}$ of Ω with global mesh size h. The finite element approximation of the weak problem takes the form: find $u_h \in V_h$ such that

$$a(u_h, v) = l(v), \quad \forall v \in V_h \tag{7.77}$$

7.5.2 Galerkin Orthogonality

To extract information about the error $e = u - u_h$ we subtract the finite element approximation (7.77) from the weak form (7.4). We then obtain the following Galerkin orthogonality property

$$a(e, v) = 0, \quad \forall v \in V_h \tag{7.78}$$

We interpret this as the error e being orthogonal to V_h with respect to the scalar product $a(\cdot, \cdot)$.

7.5.3 A Priori Error Estimates

We now have the following abstract best approximation result know as Cea's Lemma.

Theorem 7.3 (Cea's Lemma). *For the error $e = u - u_h$ it holds that*

$$\|e\|_V \leq \frac{C_2}{\alpha}\|u - v\|_V, \quad \forall v \in V_h \tag{7.79}$$

where α is the coercivity and C the continuity constant of $a(\cdot, \cdot)$.

Proof. *Starting from the coercivity of $a(\cdot, \cdot)$ we have for any $v \in V_h$*

$$\begin{aligned}
\alpha\|e\|_V^2 &\leq a(e, e) &(7.80)\\
&= a(e, u - u_h) &(7.81)\\
&= a(e, u - v + v - u_h) &(7.82)\\
&= a(e, u - v) + a(e, v - u_h) &(7.83)\\
&= a(e, u - v) + 0 &(7.84)\\
&\leq C_2\|e\|_V\|u - v\|_V &(7.85)
\end{aligned}$$

where we have used the Galerkin orthogonality to deduce that $a(e, v - u_h) = 0$. In the last line we have also used the continuity of $a(\cdot, \cdot)$. The claim follows by dividing by $\|e\|_V$.

We can extend Cea's Lemma by choosing $v = \pi u \in V_h$ the interpolant of u, and recalling a standard interpolation estimate. In doing so we immediately have the following a priori error estimate.

Theorem 7.4. *The error $e = u - u_h$ satisfies the a priori estimate*

$$\|e\|_V \leq Ch^2\|D^2u\| \tag{7.86}$$

This shows that the error will tend to zero as the mesh size h tend to zero.

7.5.4 A Posteriori Error Estimate

A posteriori estimates can not be derived so elegantly as a priori estimates in the abstract setting. All the same, to derive a formal a posteriori estimate we observe that for any $v \in V_h$ we have

$$\alpha\|e\|_V^2 \leq a(e,e) \tag{7.87}$$
$$= a(e,e-v) \tag{7.88}$$
$$= a(u,e-v) - a(u_h,e-v) \tag{7.89}$$
$$= l(e-v) - a(u_h,e-v) \tag{7.90}$$

Now, introducing the weak residual $R(u_h)$, defined by

$$(R(u_h),w) = l(w) - a(u_h,w), \quad \forall w \in V \tag{7.91}$$

we infer the following error representation formula

$$\alpha\|e\|_V^2 \leq (R(u_h),e-v) \tag{7.92}$$

which is the starting point for deriving a posteriori error estimates for elliptic equations.

By defining the following so-called dual norm of $R(u_h)$

$$\|R(u_h)\|_{V^*} = \sup_{w\in V} \frac{(R(u_h),w)}{\|w\|_V} \tag{7.93}$$

and using (7.92) with $v = 0$ we have

$$\alpha\|e\|_V^2 = \frac{(R(u_h),e)}{\|e\|_V}\|e\|_V \leq \sup_{w\in V}\frac{(R(u_h),w)}{\|w\|_V}\|e\|_V = \|R(u_h)\|_{V^*}\|e\|_V \tag{7.94}$$

Dividing by $\|e\|_V$ we formally have the a posteriori estimate

$$\|e\|_V \leq \frac{1}{\alpha}\|R(u_h)\|_{V^*} \tag{7.95}$$

As simple as its looks the dual norm is still complicated to compute due to the supremum. Therefore the error representation formula is usually instead used as is for the particular equation under consideration with $v = \pi e \in V_h$, the interpolant of e. The unspoken hope is to extract factors of type $\|e\|_V$ to divide with and also to obtain something that is simple to computable. Fortunately, this is often possible. For example, for the general equation $Lu = f$ with zero boundary conditions and L the general second order elliptic operator defined by (7.2), we have the error representation formula

$$\|\nabla e\|^2 \leq (f, e - \pi e) - (a\nabla u_h, \nabla(e - \pi e)) - (cu_h, e - \pi e) \tag{7.96}$$

which can be further manipulated to yield

$$\|\nabla e\|^2 \leq C \left(\sum_{K \in \mathscr{K}} h_K^2 \|f + \nabla \cdot (a\nabla u_h) - cu_h\|_K^2 + \tfrac{1}{4} h_K \|[n \cdot (a\nabla u_h)]\|_{\partial K}^2 \right)^{1/2} \|\nabla e\| \tag{7.97}$$

which is our desired a posteriori error estimate.

7.6 Problems

Exercise 7.1. Show that the solution u to the abstract weak problem (7.4) satisfies the stability estimate $\|u\|_V \leq C/\alpha$. *Hint:* Use the coercivity and continuity of $a(\cdot,\cdot)$.

Exercise 7.2. Show that if u satisfies $a(u,v) = l(v)$ for all $v \in V$ then u also minimizes the functional $J(v) = \frac{1}{2}a(v,v) - l(v)$ on V.

Exercise 7.3. Use the Poincaré inequality to show that $\|\nabla v\|$ and $\|v\|_{H^1} = \|\nabla v\| + \|v\|$ are equivalent norms on $H_0^1(\Omega)$. In particular, verify that $\|\nabla v\| = 0$ implies $v = 0$ on $H_0^1(\Omega)$.

Exercise 7.4. What numerical values do the constants α, C_1, and C_2 have for the problem $-\Delta u = xy^2$ on the square $\Omega = [-1,2] \times [0,3]$ assuming a zero boundary condition? *Hint:* The relevant space is $H_0^1(\Omega)$ with norm $\|v\|_{H_0^1} = \|\nabla v\|$.

Exercise 7.5. Consider

$$a(u,v) = v^T A u, \qquad l(v) = v^T b, \qquad V = \mathbb{R}^n$$

where A is a real $n \times n$ matrix, b is a real $n \times 1$ vector, and $\|\cdot\|_V$ the usual Euclidean norm.

(a) Show by a simple argument from linear algebra that there exist a unique solution $u \in V$ to (7.4) assuming that $a(\cdot,\cdot)$ is coercive on V.
(b) Show that the coerciveness of $a(\cdot,\cdot)$ is not really necessary in this case when V has finite dimension, and that it suffice that $a(v,v) > 0$.

Exercise 7.6. Verify the trace inequality (7.21) for the particular choice $v = x$ on the square $\Omega = [0,L]^2$ with side length L. How does the constant C in the inequality depend on L?

Chapter 8
The Finite Element

Abstract In this chapter we study the concept of a finite element in some depth. We begin with the classical definition of a finite element as the triplet of a simplex, a polynomial space, and a set of functionals. We then show how to derive shape functions for the most common Lagrange elements on the reference triangle. The isoparametric mapping is introduced as a tool to allow for curved elements, and to simplify the computation of the element stiffness matrix and load vector. We finish by presenting some more exotic elements, such as the Raviart Thomas and Nedelec vector elements.

8.1 Different Types of Finite Elements

8.1.1 Formal Definition of a Finite Element

Formally, a finite element consists of the following triplet:

- A geometric simplex K.
- A polynomial function space P on K.
- A set of $n = \dim(P)$ functionals $L_i(\cdot)$, $i = 1, 2, \ldots, n$, defining the degrees of freedom.

The standard choice of geometric simplex are triangles, or tetrahedrons, but quadrilaterals, prisms, and bricks are also quite common. Triangle and tetrahedron meshes have the advantage of being able to represent geometries with curved boundaries. On the other hand, quadrilaterals and bricks might be more easy to implement in software. For example, if the mesh consists of uniformly shaped squares or cubes, then the element stiffness matrix can be precomputed and stored away once and for all. This obviously helps writing clean and correct code. Prisms are primarily used for geometries with cylindrical symmetries, such as pipes, for instance. Powerful mesh generators have been developed over the years for these simplex types. In the following we shall concentrate on triangular simplex for the sake of simplicity. However, the ideas presented are quite general and the reader should have no difficulties extending them to three dimensions.

To facilitate working with the space P let us equip it with a basis $\{N_j\}_{j=1}^{n}$. The basis functions S_j are generally called shape functions.

The n functionals $L_j(\cdot)$, $j = 1,2,\dots,n$, can be used to uniquely define the shape functions S_j by requiring them to satisfy

$$L_i(S_j) = \delta_{ij}, \quad i,j = 1,2,\dots,n \tag{8.1}$$

The set of shape functions is then said to be a nodal basis for P. The ability of the functionals to uniquely determine the shape functions is called unisolvency, and can be thought of as a compatibility condition for $L_i(\cdot)$ and P. From the strict mathematical point of view a finite element is called unisolvent if, and only if, $L_i(v) = 0$ implies $v = 0$ for all $v \in P$ and i. As we shall see the actual calculation of the shape functions is easy as it amounts to solving a linear system of size $n \times n$.

However, there is a more important but also more subtle task for the functionals $L_i(\cdot)$, namely, that of specifying the behavior of the shape functions between adjacent simplex. To see this let us say we want our finite element functions to be continuous on the whole domain $\Omega = \cup K$. We must then take care when choosing the functionals $L_i(\cdot)$ so that the corresponding shape functions also become continuous, especially across simplex boundaries. In other words, the functionals ultimately determine the smoothness and approximation properties of the finite element space V_h.

The particular choice of functionals $L_i(\cdot)$ give rise to families of finite elements sharing similar properties, although they might have different polynomial order, for instance. The Lagrange family is the most popular and widely used. In two dimensions the defining functionals are

$$L_i(v) = v(N_i), \quad i = 1,2,\dots,n \tag{8.2}$$

where $N_i = (x_1^{(i)}, x_2^{(i)})$ are a set of n carefully selected node points. Notice that the functionals are the simplest possible in the sense that they only consist of point evaluation of v at the nodes. In the linear case $P = P^1(K)$ and with K a triangle these node points are the triangle vertices, and the shape functions N_j, $j = 1,2,3$, are the familiar hat functions.

The Lagrange shape functions are continuous, but have discontinuous derivatives across element boundaries. Thus, it is a C^0 element, which suffice to approximate H^1 space. In some applications it is, however, necessary to use more regular (i.e., smoother) elements. An example of a C^1 element is the triangular Argyris element, which is a quintic polynomial with continuous derivatives. This element was invented to approximate the Hilbert space $H^2 = \{v : v \in L^2, Dv \in L^2, D^2v \in L^2\}$, which is the appropriate space for some fourth order problems, such as $\Delta^2 u = 0$ for instance. Not surprisingly, construction of the Argyris elements is more elaborate than for the Lagrange element. Indeed, there are 21 defining functionals involving point evaluation of first, normal, and second order derivatives for the Argyris element.

On the other hand, it is also possible to have completely discontinuous finite elements with no continuity between adjacent simplex. However, this kind of element requires modification of the variational formulation to work.

In the next section we shall see how the shape functions can be computed for a few different finite elements.

8.1.2 Shape Functions for the Linear Lagrange Triangle

Let us derive the shape functions for the linear Lagrange finite element. To this end, let $\bar{K}$ be the domain $\bar{K} = \{(r,s) : 0 < r,s < 1, r+s < 1\}$, that is, the triangle with vertices at origo, $(1,0)$, and $(0,1)$. This triangle is often called the reference triangle, see Figure (8.1). For reasons soon to become clear we use r and s as coordinates rather than x_1 and x_2.

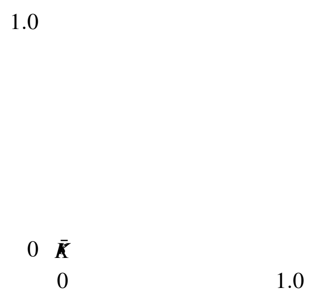


Fig. 8.1 Node points for the linear Lagrange element on the reference triangle $\bar{K}$.

By definition, the appropriate space P is the space of linear polynomials $P^1(\bar{K})$ on $\bar{K}$, and the defining functionals are given by

$$L_1(v) = v(0,0), \quad L_2(v) = v(1,0), \quad L_3(v) = v(0,1) \tag{8.3}$$

Perhaps the simplest basis for $P^1(K)$ is the canonical basis $\{1, r, s\}$, so anyone of the three shape functions S_j, $j = 1,2,3$, can be expressed as a linear combination of 1, r, and s. For example, S_1 can be written $S_1 = c_1 + c_2 r + c_3 s$, where c_i, $i = 1,2,3$ are coefficients to be determined. To do so, we demand that $L_i(S_1) = \delta_{i1}$, which gives the 3×3 linear system

$$e_1 = \begin{bmatrix}1\\0\\0\end{bmatrix} = \begin{bmatrix}L_1(1) & L_1(r) & L_1(s)\\ L_2(1) & L_2(r) & L_2(s)\\ L_3(1) & L_3(r) & L_3(s)\end{bmatrix} \begin{bmatrix}c_1\\c_2\\c_3\end{bmatrix} = \begin{bmatrix}1&0&0\\1&1&0\\1&0&1\end{bmatrix} \begin{bmatrix}c_1\\c_2\\c_3\end{bmatrix} = Vc \tag{8.4}$$

for the unknown coefficients c_i. Note that the entries of V are simple to evaluate. For example, the first row is point evaluation of the functions 1, r, and s at origo. This immediately gives us $V_{11} = L_1(1) = 1$, $V_{12} = L_1(r) = 0$, $V_{13} = L_1(s) = 0$, and so on. The matrix V is generally called a Vandermonde matrix. Computing $V^{-1}e_1$ we readily obtain $c = [1, -1, -1]^T$, from which we deduce that $S_1 = c_1 + c_2 r + c_3 s = 1 - r - s$. Proceeding similarly for the shape functions S_2 and S_3 we eventually find that

$$S_1 = 1 - r - s \tag{8.5}$$
$$S_2 = r \tag{8.6}$$
$$S_3 = s \tag{8.7}$$

which we recognize as the usual hat functions on $\bar{K}$.

We summarize by listing a routine for evaluating the linear shape functions and their partial derivatives at a point (r,s) in $\bar{K}$.

```
function [S,dSdr,dSds] = P1shapes(r,s)
S=[1-r-s; r; s];
dSdr=[-1; 1; 0];
dSds=[-1; 0; 1];
```

8.1.3 Shape Functions for the Quadratic Lagrange Triangle

For the quadratic Lagrange shape functions on the reference triangle $\bar{K}$, the polynomial space P is $P^2(\bar{K})$, and the defining functionals are given by

$$L_1(v) = v(0,0), \quad L_2(v) = v(1,0), \quad L_3(v) = v(0,1) \tag{8.8}$$
$$L_4(v) = v(0.5,0.5), \quad L_5(v) = v(0,0.5), \quad L_6(v) = v(0.5,0) \tag{8.9}$$

In other words the nodes are the triangle vertices and the mid-points of the edges. See Figure 8.2.

Since a general polynomial of two variables has six coefficients, there must be six shape functions S_j, $j = 1,2,\dots,6$. To see this note that the canonical basis for $P^2(\bar{K})$ is $\{1, r, s, r^2, rs, s^2\}$, and that N_j is a linear combination of these monomials. Thus, we have $S_1 = c_1 + c_2 r + c_3 s + c_4 r^2 + c_5 rs + c_6 s^2$ for example. To determine the coefficients c_i, $i = 1,2,\dots,6$, we again demand that $L_i(S_1) = \delta_{i1}$, which gives us the 6×6 linear system

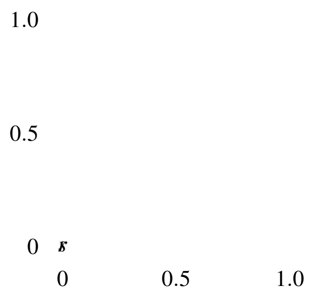


Fig. 8.2 Node points for the quadratic Lagrange element.

$$e_1 = \begin{bmatrix}1\\0\\0\\0\\0\\0\end{bmatrix} = \begin{bmatrix} L_1(1) & L_1(r) & L_1(s) & L_1(r^2) & L_1(rs) & L_1(s^2) \\ L_2(1) & L_2(r) & L_2(s) & L_2(r^2) & L_2(rs) & L_2(s^2) \\ L_3(1) & L_3(r) & L_3(s) & L_3(r^2) & L_3(rs) & L_3(s^2) \\ L_4(1) & L_4(r) & L_4(s) & L_4(r^2) & L_4(rs) & L_4(s^2) \\ L_5(1) & L_5(r) & L_5(s) & L_5(r^2) & L_5(rs) & L_5(s^2) \\ L_6(1) & L_6(r) & L_6(s) & L_6(r^2) & L_6(rs) & L_6(s^2) \end{bmatrix} \begin{bmatrix}c_1\\c_2\\c_3\\c_4\\c_5\\c_6\end{bmatrix} \tag{8.10}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0.5 & 0.5 & 0.25 & 0.25 & 0.25 \\ 1 & 0 & 0.5 & 0 & 0 & 0.25 \\ 1 & 0.5 & 0 & 0.25 & 0 & 0 \end{bmatrix} \begin{bmatrix}c_1\\c_2\\c_3\\c_4\\c_5\\c_6\end{bmatrix} = Vc \tag{8.11}$$

from which it follows that $c = [1,\ -3,\ -4,\ 2,\ 4,\ 2]^T$.

In a similar fashion for the other shape functions can be found. Their explicit formulas are

$$S_1 = 1 - 3r - 3s + 2r^2 + 4rs + 2s^2 \tag{8.12}$$

$$S_2 = 2r^2 - r \tag{8.13}$$

$$S_3 = 2s^2 - s \tag{8.14}$$

$$S_4 = 4rs \tag{8.15}$$

$$S_5 = 4s - 4rs - 4s^2 \tag{8.16}$$

$$S_6 = 4r - 4r^2 - 4rs \tag{8.17}$$

We summarize by listing a routine for evaluating the quadratic shape functions and their partial derivatives at a point (r,s) in $\bar{K}$.

```
function [S,dSdr,dSds] = P2shapes(r,s)
S=[1-3*r-3*s+2*r^2+4*r*s+2*s^2;
   2*r^2-r;
   2*s^2-s;
   4*r*s;
   4*s-4*r*s-4*s^2;
   4*r-4*r^2-4*r*s];
dSdr=[-3+4*r+4*s; 4*r-1; 0; 4*s; -4*s; 4-8*r-4*s];
dSds=[-3+4*r+4*s; 0; 4*s-1; 4*r; 4-4*r-8*s; -4*r];
```

8.1.4 Higher Order Triangular Lagrange Elements

The procedure for computing Lagrange shape functions on the reference triangle $\bar{K}$ generalizes to higher order. If there order of the polynomials space P is o, then there are $n = (o+1)(o+2)/2$ nodes and shape functions. The nodes are positioned in a lattice called the principal lattice of the reference triangle $\bar{K}$. We have already seen this lattice for $o = 1$ and 2. Figure 8.3 shows it also for $o = 3$ and 4. The generalization to any higher order should be obvious.

$o = 3$ $o = 4$

Fig. 8.3 Node points for the cubic and quartic Lagrange elements.

8.1.5 Shape Functions for the Bilinear Elements

Shape functions can also be constructed on quadrilaterals. To do so, let $\bar{Q}$ be the reference square $\bar{Q} = \{(r,s) : -1 < r,s < 1\}$, and let $P(\bar{Q})$ be the space of bilinear functions spanned by the canonical basis $\{1,r,s,rs\}$. The nodes $(x_1^{(i)},x_2^{(i)})$, $i = 1,2,3,4$ are the four corners of $\bar{Q}$, and the defining functionals are again given by $L_i(v) = v(x_1^{(i)},x_2^{(i)})$. We leave it as a simple exercise to the reader to verify that the shape functions take the form

$$S_1 = (1-r)(1-s) \tag{8.18}$$

$$S_2 = (1+r)(1-s) \tag{8.19}$$

$$S_3 = (1+r)(1+s) \tag{8.20}$$

$$S_4 = (1-r)(1+s) \tag{8.21}$$

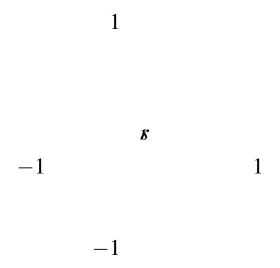


Fig. 8.4 Node points for the bilinear element on the reference square $\bar{Q}$.

8.2 The Isoparametric Mapping

Up to now we have used various tricks to integrate the entries of the element stiffness matrix and load vector. However, this approach quickly gets cumbersome for higher order elements. Also, to improve the geometry representation of the computational domain and not only the solution approximation we would like to be able to work with elements with curved boundaries. Fortunately, it turns out that these two obstacles can be overcome through the concepts of numerical quadrature and isoparametric elements, respectively. The combination of these two ideas allows for a simple and uniform treatment of the elemental assembly procedure. We shall present the isoparametric mapping for triangle elements, although the ideas directly carry over to other element types, such as tetrahedrons for instance.

The setting up the isoparametric map is easily described. Suppose we have a mesh triangle K with nodes at $N_i = (x_1^{(i)}, x_2^{(i)})$, $i = 1, 2, \dots, n$. We will refer to K this as the physical element, as opposed to the reference element $\bar{K}$. Now, the basic idea is to use the shape functions S_j on $\bar{K}$ to describe the geometry of K through the formulas

$$x_1(r,s) = \sum_{i=1}^{n} x_1^{(i)} S_i(r,s) \tag{8.22}$$

$$x_2(r,s) = \sum_{i=1}^{n} x_2^{(i)} S_i(r,s) \tag{8.23}$$

In other words given a point (r,s) in $\bar{K}$ the above formulas maps it to the physical point (x_1, x_2) in K. Thus, the coordinates x_1 and x_2 are parameterized by r and s. This is the isoparametric mapping. Observe that this yields curved boundaries on K whenever the node coordinates $(x_1^{(i)}, x_2^{(i)})$ lying on triangle edges do not lie on a straight lines between the vertices.

Of course, any finite element function v on K is also expressed using the shape functions.

$$v(r,s) = \sum_{i=1}^{n} v_i S_i(r,s) \tag{8.24}$$

Since the stiffness matrix involves partial derivatives of v we use the chain rule to differentiate with respect to r and s, yielding

$$\frac{\partial v}{\partial x_1} = \frac{\partial v}{\partial r}\frac{\partial r}{\partial x_1} + \frac{\partial v}{\partial s}\frac{\partial s}{\partial x_1} \tag{8.25}$$

$$\frac{\partial v}{\partial x_2} = \frac{\partial v}{\partial r}\frac{\partial r}{\partial x_2} + \frac{\partial v}{\partial s}\frac{\partial s}{\partial x_2} \tag{8.26}$$

In matrix form we can write this as

$$\begin{bmatrix} \frac{\partial v}{\partial x_1} \\ \frac{\partial v}{\partial x_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial r}{\partial x_1} & \frac{\partial s}{\partial x_1} \\ \frac{\partial r}{\partial x_2} & \frac{\partial s}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial v}{\partial r} \\ \frac{\partial v}{\partial s} \end{bmatrix} = J^{-1} \begin{bmatrix} \frac{\partial v}{\partial r} \\ \frac{\partial v}{\partial s} \end{bmatrix} \tag{8.27}$$

where we have introduced the Jacobian matrix J, defined by

$$J = \begin{bmatrix} \frac{\partial x_1}{\partial r} & \frac{\partial x_2}{\partial r} \\ \frac{\partial x_1}{\partial s} & \frac{\partial x_2}{\partial s} \end{bmatrix} \tag{8.28}$$

Here, the explicit expressions for the entries of J are given by

$$J_{11} = \frac{\partial x_1}{\partial r} = \sum_i \frac{\partial S_i}{\partial r} x_1^{(i)} \tag{8.29}$$

$$J_{12} = \frac{\partial x_2}{\partial r} = \sum_i \frac{\partial S_i}{\partial r} x_2^{(i)} \tag{8.30}$$

$$J_{21} = \frac{\partial x_1}{\partial s} = \sum_i \frac{\partial S_i}{\partial s} x_1^{(i)} \tag{8.31}$$

$$J_{22} = \frac{\partial x_2}{\partial s} = \sum_i \frac{\partial S_i}{\partial s} x_2^{(i)} \tag{8.32}$$

To summarize, given the node coordinates, shape functions, and nodal values v_i of a finite element function, we can compute its partial derivative at a point (x_1, x_2) in K, or equivalently, (r, s) in $\bar{K}$, by solving the 2×2 linear system (**??**).

We remark that the invertability of J depends on the quality of K, which can be used to show that the isoparametric map is one to one.

We observe that for the linear Lagrange finite element the Jacobian matrix is given by

$$J = \begin{bmatrix} x_1^{(2)} - x_1^{(1)} & x_2^{(2)} - x_2^{(1)} \\ x_2^{(3)} - x_2^{(1)} & x_2^{(3)} - x_2^{(1)} \end{bmatrix} \tag{8.33}$$

where $(x_1^{(i)}, x_2^{(i)})$, $i = 1, 2, 3$, are the vertices of K. Further, the determinant of J is given by

$$\det(J) = 2|K| \tag{8.34}$$

This is to be expected since we might recall from calculus that the determinant of a mapping is the area scale between the image and range of the mapping (i.e., two domains K and $\bar{K}$). Now, the area of $\bar{K}$ is $1/2$. Hence, the factor 2 in front of $|K|$. Needless to say $\det(J)$ is constant for this element.

A routine for computing the Jacobian J at (r, s) given the n node coordinates $(x_1^{(i)}, x_2^{(i)})$ is given below.

```
function [S,dSdx,dSdy,detJ] = Isopmap(x,y,r,s,shapefcn)
[S,dSdr,dSds]=shapefcn(r,s);
j11=dot(dSdr,x); j12=dot(dNdr,y);
j21=dot(dSds,x); j22=dot(dNds,y);
detJ=j11*j22-j12*j21;
dSdx=( j22*dSdr-j12*dSds)/detJ;
dSdy=(-j21*dSdr+j11*dSds)/detJ;
```

Here, `shapefun` is assumed to be a function handle, which can be either of the subroutines `P1shapes` and `P2shapes`, depending on if we want to evaluate linear or quadratic shape functions.

8.2.1 Quadrature

The entries of the stiffness matrix and load vector involves integrals over the physical elements K. However, since we want to compute on the reference element $\bar{K}$ we have to study how the isoparametric map $(x_1,x_2) \mapsto (r,s)$ affects integrals. To do so, we recall the following change of variables formula

$$\int_K f(x_1,x_2)\,dx = \int_{\bar{K}} f(r,s)\det(J(r,s))\,drds \tag{8.35}$$

which allows us to integrate over $\bar{K}$ instead of K.

Now, approximating the integral over $\bar{K}$ by a quadrature formula we have

$$\int_{\hat{K}} f(r,s)\det(J(r,s))\,drds \approx \sum_{q=1}^{n_q} w_q f(r_q,s_q)\det(J(r_q,s_q)) \tag{8.36}$$

where N_q is the number of quadrature points, w_q the quadrature weights, and (r_q,s_q) the quadrature points.

The construction of efficient quadrature rules on triangles is difficult and still to some extent unexplored territory. All the same, a routine which tabulates Gauss quadrature weights and points on $\bar{K}$ up to precision four (i.e., polynomials of maximal degree four can be integrated exactly) is given below. The weights are scaled so that they sum to one. As a consequence the determinant $\det(J)$ needs to be divided by two to integrate correctly.

```
function [rspts,qwgts] = Gausspoints(precision)
switch precision
 case 1
  qwgts=[1];
  rspts=[1/3 1/3];
 case 2
  qwgts=[1/3 1/3 1/3];
  rspts=[1/6 1/6;
 2/3 1/6;
 1/6 2/3];
 case 3
  qwgts=[-27/48 25/48 25/48 25/48];
  rspts=[1/3 1/3;
 0.2 0.2;
 0.6 0.2;
 0.2 0.6];
 case 4
  qwgts=[0.223381589678011
 0.223381589678011
 0.223381589678011
 0.109951743655322
```

```
  0.109951743655322
  0.109951743655322];
   rspts=[0.445948490915965 0.445948490915965;
  0.445948490915965 0.108103018168070;
  0.108103018168070 0.445948490915965;
  0.091576213509771 0.091576213509771;
  0.091576213509771 0.816847572980459;
  0.816847572980459 0.091576213509771];
  otherwise
   error('Quadrature precision too high')
 end
```

As a small example of use we integrate the mass matrix $M^K = (S_i, S_j)_K$ on a triangle K with vertices at $(0,0)$, $(3,0)$ and $(-2,4)$, using linear Lagrange shape functions.

```
[rspts,qwgts]=Gausspoints(2) % quadrature rule
x=[0 3 -2]; % node x-coordinates
y=[0 0  4]; %      y-
MK=zeros(3,3); % allocate element mass matrix
for q=1:length(qwgts) % quadrature loop
  r=rspts(q,1); % r coordinate
  s=rspts(q,2); % s
  [S,dSdx,dSdy,detJ]=Isopmap(x,y,r,s,@P1shape); % map
  wxarea=qwgts(q)*detJ/2; % weight times det(J)
  MK=MK+(S*S')*wxarea; % compute and add integrand to MK
end
```

8.2.2 Renumbering the Mesh for Quadratic Nodes

As we have seen triangular Lagrange finite elements have $n = (o+1)(o+2)/2$ nodes per element. To correctly assemble the stiffness matrix and load vector it is therefore necessary to modify the mesh to include all nodes. In this section we show how this can be done efficiently for the special case $o = 2$. Recall that quadratic Lagrange elements have nodes at the vertices and the mid-points of the edges. As the vertex nodes are already numbered by the mesh generator `initmesh` the problem boils down to numbering the edge nodes. To do so, we first record the node to edge incidence by using a sparse matrix A. More precisely, if there is a edge between vertex i and j then we set $A(i, j) = -1$. Using the standard point and triangle matrices `p` and `t` this can efficiently be done with the following code snippet.

```
np=size(t,2); % number of vertices
nt=size(t,2); % number of triangles
i=t(1,:); % i=1st vertex within all elements
j=t(2,:); % j=2nd
```

```
k=t(3,:); % k=3rd
A=sparse(j,k,-1,np,np);   % 1st edge is between (j,k)
A=A+sparse(i,k,-1,np,np); % 2nd                 (i,k)
A=A+sparse(i,j,-1,np,np); % 3rd                 (i,j)
```

Since the edge between vertex i and j trivially also lies between vertex j and i we should have $A(i,j) = A(j,i) = -1$. To ensure this we add the transpose A^T to A and look for negative matrix entries, that is,

```
A=-((A+A.')<0);
```

We can look at the stored matrix entries (i.e., created edges) by typing

```
A=triu(A); % extract upper triangle of A
[r,c,v]=find(A); % rows, columns, and values(=-1)
```

Now, to number the edges we simply tale the matrix values, which are all -1, and renumber them consecutively, staring from 1. Then, we reassemble the upper triangle part of A. Finally, we expand A to symmetric form by again adding A^T to A.

```
v=[1:length(v)]; % renumber values (ie. edges)
A=sparse(rows,cols,entries,np,np); % reassemble A
A=A+A'; % expand A to a symmetric matrix
```

The edge numbers for the three edges of each element can now be read form A.

```
edges=zeros(nt,3);
for k=1:nt
  edges(k,:)=[A(t(2,k),t(3,k))
              A(t(1,k),t(3,k))
              A(t(1,k),t(2,k))]';
end
```

In the Appendix we list a routine called `Tri2Edge` containing the above code.

Using the edge numbering routine it is straight forward to insert the new nodes into the point and triangle matrices `p` and `t`.

```
function [p,t] = ChangeP1toP2Mesh(p,t)
np=size(p,2); % number of nodes
edges=Tri2Edge(p,t); % get element edge numbers
edges=edges+np; % change edges to new nodes
i=t(1,:); j=t(2,:); k=t(3,:);
e=edges(:,1);
p(1,e)=0.5*(p(1,j)+p(1,k)); % edge node coordinates
p(2,e)=0.5*(p(2,j)+p(2,k));
e=edges(:,2);
p(1,e)=0.5*(p(1,i)+p(1,k));
p(2,e)=0.5*(p(2,i)+p(2,k));
e=edges(:,3);
```

```
p(1,e)=0.5*(p(1,i)+p(1,j));
p(2,e)=0.5*(p(2,i)+p(2,j));
t(7,:)=t(4,:); % move subdomain info, resize t
t(4:6,:)=edges'; % insert edge nodes into t
```

For higher order Lagrange elements it is necessary to insert more nodes on the edges, but this is fairly simple once these have been properly numbered. Higher order elements also contains interior nodes, but these are trivial to number uniquely.

8.2.3 Assembly of the Isoparametric Quadratic Stiffness Matrix

We next show how to assemble the usual stiffness matrix on a mesh renumbered for isoparametric Lagrange finite elements of order 2.

```
function [A,M,F] = IsoP2StiffMat2D(p,t,force)
[rspts,qwgts]=Gausspoints(4); % quadrature rule
np=size(p,2); % number of nodes
nt=size(t,2); % number of elements
A=sparse(np,np); % allocate stiffness matrix
for i=1:nt % loop over elements
  nodes=t(1:6,i); % node numbers
  x=p(1,nodes); % node x-coordinates
  y=p(2,nodes); %      y-
  AK=zeros(6,6); % elements stiffness
  for q=1:length(qwgts) % quadrature loop
   r=rspts(q,1); % quadrature r-coordinate
   s=rspts(q,2); %            s-
   [S,dSdx,dSdy,detJ]=Isopmap(x,y,r,s,@P2shapes);
   wxarea=qwgts(q)*detJ/2; % weight times area
   AK=AK+(dSdx*dSdx'+dSdy*dSdy')*wxarea; % element stiffness
  end
  A(nodes,nodes)=A(nodes,nodes)+AK;
end
```

To call this routine one can type for example

```
[p,e,t] = initmesh('squareg');
[p,t] = ChangeP1toP2Mesh(p,t);
A = IsoP2StiffMat2D(p,t);
```

8.3 Some More Exotic Finite Elements

Finite elements are often invented for a particular purpose. They might be designed for a specific application area, or constructed to mimic a particular function space.

As we recall Lagrange elements approximate H^1 functions, while the Argyris element approximate H^2 functions. In this section we shall briefly look at a few exotic elements, which are tailor made to mimic a certain Hilbert space, or are somewhat peculiar.

8.3.1 The Crouzeix-Raviart Element

The Crouzeix-Raviart element is a finite element defined on triangles or tetrahedrons. It a linear element which is only continuous at the mid-points of the triangle edges or tetrahedron faces. Figure 8.5 shows a mesh of the unit square and the Crouzeix-Raviart interpolant of $1+2\sin(3x_1)$. Note that the interpolant is discontinuous except at the mid-point of the triangle edges.

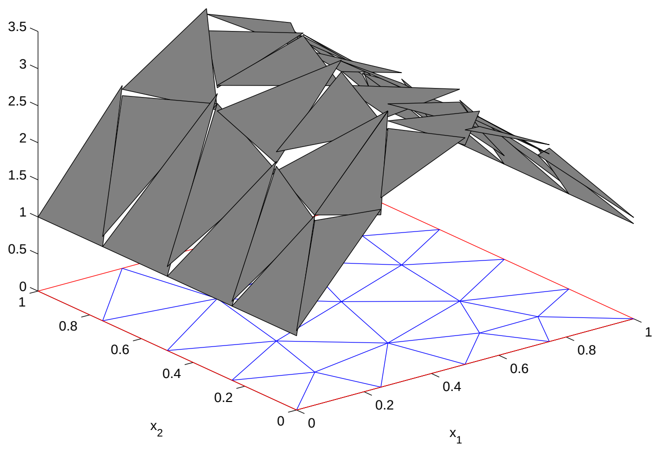


Fig. 8.5 Crouzeix-Raviart interpolant of $1+2\sin(3x_1)$ on a mesh of the unit square.

On a standard straight sided triangle K, the polynomial space for the Crouzeix-Raviart element is $P^1(K)$, and the defining functionals are given by

$$L_i(v) = (v,1)_{E_i}, \quad i = 1,2,3 \tag{8.37}$$

where E_i is triangle edge i. In other words the degrees of freedom is the mean value of v over E_i.Now, since the mean of a linear function over E_i is the value of v at the mid-point m_i of E_i, we can alternatively define the functionals by

$$L_i(v) = v(m_i), \quad i = 1,2,3 \tag{8.38}$$

The explicit expressions for the shape functions are given by

$$S_1^{CR} = -\varphi_1 + \varphi_2 + \varphi_3, \quad S_2^{CR} = \varphi_1 - \varphi_2 + \varphi_3, \quad S_3^{CR} = \varphi_1 + \varphi_2 - \varphi_3 \tag{8.39}$$

where φ_i are the usual hat functions on K.

Because the Crouzeix-Raviart functions are continuous only at each edge midpoint they are generally discontinuous along the edges. Thus, this finite element space is not a subspace of H^1, which is a little strange since the Crouzeix-Raviart element is used to approximate precisely H^1. Finite element spaces that are not a subspace of the continuous space on which the variational equation is posed is called non-conforming.

The Crouzeix-Raviart element finds application in fluid mechanics.

8.3.2 The Lowest Order Raviart-Thomas Element

Not all finite elements are scalar. There are also vector valued elements. As the name suggests vector valued elements are used to approximate vector valued equations. One such element is the Raviart-Thomas element, which is used to approximate the Hilbert space $H(\text{div}) = \{v \in [L^2(\Omega)]^2 : \nabla \cdot v\}$, that is, the space of all vectors $v \in \mathbb{R}^2$ with bounded divergence $\nabla \cdot v$. A simple application of Green's formula shows that all such functions must have continuous normal components, which is the basic design feature of the Raviart-Thomas element. Typical applications include finite element methods for acoustics and elasticity.

Actually there is a whole family of Raviart Thomas elements, but we shall only study the simplest of them called the RT_0 element. On a general triangle K the polynomial space for RT_0 is $P = [P^0(K)]^2 + [x_1, x_2]^T P^0(K)$, that is, all vectors v of the form

$$v = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} + b \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{8.40}$$

for some coefficients a_1, a_2, and b. Further, the defining functionals are given by

$$L_i(v) = (n_i, v)_{E_i}, \quad i = 1,2,3 \tag{8.41}$$

where n_i is a unit normal on edge E_i of K.

Closed form formulas for the RT_0 shape functions can be derived and is given by

$$S_i^{RT_0} = \frac{1}{2|K|} \begin{bmatrix} x_1 - x_1^{(i)} \\ x_2 - x_2^{(i)} \end{bmatrix}, \quad i = 1,2,3 \tag{8.42}$$

where $(x_1^{(i)}, x_2^{(i)})$ are the coordinates of the vertex opposite edge E_i, see Figure 8.6.

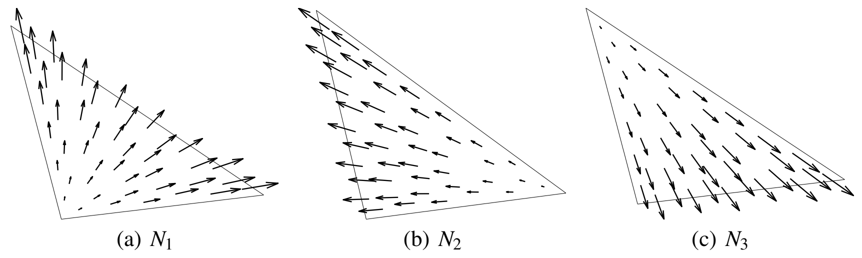


Fig. 8.6 RT_0 shape functions on a triangle.

If the normal is chosen consistently on each edge E_i in the mesh, then by construction is the RT_0 shape functions are normal continuous across any edge of two adjacent elements. This ensures that the RT_0 functions belong to $H(\text{div})$.

The isoparametric map can not be used for RT_0 elements since the divergence is not preserved by this mapping.

8.3.3 The Lowest Order Nedelec Element

The Nedelec, or edge, elements is another example of a family of vector valued finite elements. The Nedelec elements are used to approximate the space $H(\text{curl}) = \{v \in [L^2(\Omega)]^2 : \nabla \times v \in L^2(\Omega)\}$, that is the space of vectors $v \in \mathbb{R}^2$ with bounded curl. This space is the natural one for certain electromagnetic problems.

On a triangle K the polynomial space for the lowest oder Nedelec element is $P = [P^0(K)]^2 + [x_2, x_1]^T P^0(K)$ and the defining functionals are

$$L_i(v) = (t_i, v)_{E_i}, \quad i = 1, 2, 3 \tag{8.43}$$

where t is a unit tangent vector on edge E_i.

The explicit formulas for the shape functions are given by

$$S_1 = \varphi_2 \nabla \varphi_3 - \varphi_3 \nabla \varphi_2 \tag{8.44}$$
$$S_2 = \varphi_1 \nabla \varphi_3 - \varphi_3 \nabla \varphi_1 \tag{8.45}$$
$$S_3 = \varphi_1 \nabla \varphi_2 - \varphi_2 \nabla \varphi_1 \tag{8.46}$$

where φ_i are the usual hat functions, see Figure 8.7.

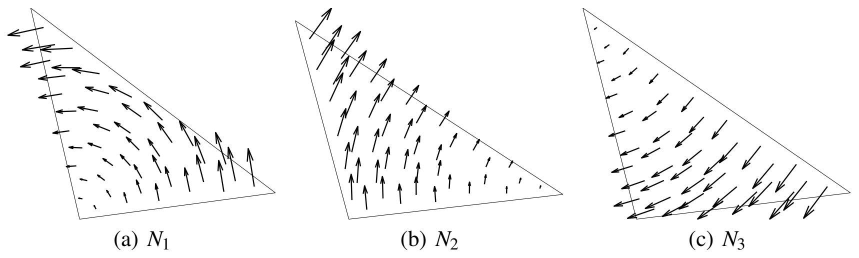


Fig. 8.7 Lowest order Nedelec shape functions on a triangle.

The Nedelec shape functions are tangent continuous across element edges provided that the tangent is chosen consistently on each edge in the mesh.

The isoparametric map can not be used for Nedelec elements since the curl is not preserved by this mapping.

8.4 Problems

Exercise 8.1. Work out the formulas for the cubic Lagrange shape functions on the reference traingle $\bar{K}$.

Exercise 8.2. Show that the bilinear element is not unisolvent if the four nodes are placed at $(-1,0)$, $(0,-1)$, $(1,0)$, and $(0,1)$ on the reference square $\bar{Q}$.

Exercise 8.3. Write a routine `IsoP2MassMat` for assembling the mass matrix M.

Exercise 8.4. Calculate the Crouzeix-Raviart interpolant of $f = 2x_1x_2 + 4$ on the reference triangle $\bar{K}$.

Exercise 8.5. How does the isoparametric map look in three dimensions?

Chapter 9
Non-linear Problems

Abstract Many real-world problems are modeled by non-linear mathematical models. Plasticity of highly stressed materials, drying paint, and turbulent flow of atmospheric gases are just some examples of such non-linear phenomenons. In fact, most of the physical, biological, and chemical processes going on around us everyday are described by more or less non-linear laws of nature. Thus, non-linear equations are of special interest, but unfortunately, they are intrinsically hard to solve. In this chapter we study the standard methods for tackling non-linear partial differential equations discretized by finite element methods, namely, Newton's method and its simplified variant Piccard, or, fixed-point iteration.

9.1 Piccard Iteration

Piccard, or fixed-point, iteration is perhaps the most primitive technique for solving non-linear equations. It is applicatble to equations of the form

$$x = g(x) \tag{9.1}$$

where we for simplicity assume that g is a scalar non-linear function of a single variable x. The basic idea is to take a first rough guess at the solution x^0, and then to compute successively until convergence

$$x^k = g(x^{k-1}) \tag{9.2}$$

This leads to the following algorithm:

Algorithm 22 Piccard Iteration for a Scalar Non-linear Equation

```
1: Choose a staring guess $x^0$, and a desired accuracy $\varepsilon$.
2: for $k = 1,2,3,\dots$ do
3:     Compute the next solution guess from $x^{k+1} = g(x^k)$.
4:     if $|\delta^k| < \varepsilon$ then
5:         Stop.
6:     end if
7: end for
```

This algorithm will converge if the operator g is a contraction mapping, that is, if there exist a constant $L < 1$ such that $\|g(x) - g(y)\| \leq L\|x - y\|$ for all x and y. To see this, let $\bar{x}$ be the exact solution to (9.1) (i.e., a so-called fixed point). Then, by subtracting $\bar{x} = g(\bar{x})$ from (9.1) we have $\|x_{k+1} - \bar{x}\| = \|g(x^k) - g(\bar{x})\| \leq L\|x^k - \bar{x}\| \leq L^k\|x^0 - \bar{x}\|$, from which we see that convergence is indeed guaranteed if $L < 1$.

Piccard iteration is simple to implement, but its rate of convergence is often slow.

9.2 Newton's Method

Besides Piccard iteration there is also Newton's method for solving non-linear equations. Newton's method is more complicated than the Piccard iteration technique, but it usually converges much faster. To explain Newton's method let us again considering the non-linear equation $g(x) = 0$.

The first step is to assume that the solution $\bar{x}$ can be written as the sum

$$\bar{x} = x_0 + \delta \tag{9.3}$$

where x_0 is some known guess of $\bar{x}$ and δ a correction. The unspoken hope is that x_0 is close to $\bar{x}$ so that δ is small. Next, from the Taylor expansion of $g(x)$ around $\bar{x}$, we have

$$g(\bar{x}) = g(x_0 + \delta) = g(x_0) + g'(x_0)\delta + \mathscr{O}(\delta^2) \tag{9.4}$$

Neglecting second order terms, and using that $g(\bar{x}) = 0$, we further have

$$0 \approx g(x_0) + g'(x_0)\delta \tag{9.5}$$

The pivotal point here is that this is a linear relation with respect to δ, and even if it is not really an equation, we use it to define an approximate correction $\delta^0 \approx \delta$. Thus, by evaluating

$$\delta^0 = -g(x^0)/g'(x^0) \tag{9.6}$$

and adding δ^0 to x^0 we ought to get a better approximation of $\bar{x}$ than x^0, at least if x^0 is close to $\bar{x}$. This line of reasoning leads to the following algorithm, which is precisely Newton's method:

Algorithm 23 Newton's Method for a Scalar Non-linear Equation

```
1: Choose a staring guess $x^0$, and a desired accuracy $\varepsilon$.
2: for $k = 1,2,3,\dots$ do
3:     Compute the correction $\delta^k = -g(x^k)/g'(x^k)$.
4:     Update the solution guess $x^{k+1} = x^k + \delta^k$.
5:     if $|\delta^k| < \varepsilon$ then
6:         Stop.
7:     end if
8: end for
```

Newton's method is popular because it usually converges rapidly. One can show that

$$\|x_{k+1} - \bar{x}\| \leq C\|x_k - \bar{x}\|^2 \tag{9.7}$$

when x_k is sufficiently close to $\bar{x}$. From this we see that the asymptotic rate of convergence is quadratic, which is very fast for any numerical method.

The primary drawback of Newton's method is that it requires information about the derivative $g'(x)$, which can be costly to compute.

9.3 The Non-linear Poisson Equation

Having derived Newton's method for a scalar equation we shall now do the same for a non-linear partial differential equation. We do this by first linearizing the continuous problem and then apply finite element discretization. As model problem we use the non-linear Poisson equation

$$-\nabla \cdot (a(u)\nabla u) = f, \quad \text{in } \Omega \tag{9.8a}$$

$$u = 0, \quad \text{on } \partial\Omega \tag{9.8b}$$

where a and f are given coefficients. The non-linearity is due to the coefficient $a = a(u)$, which depends on the unknown solution u. In order to fulfill the Lax-Milgram lemma we assume that $a(u)$ is a positive function on Ω. Typically, $a(u)$ is, or can be approximated, by a polynomial in u.

9.3.1 The Newton-Galerkin Method

As usual, multiplying $-\nabla \cdot (a(u)\nabla u) = f$ by a test function which is zero on the boundary $\partial\Omega$, and integrating by parts we obtain the weak form of (9.1): find $u \in H_0^1$ such that

$$(a(u)\nabla u, \nabla v) = (f, v), \quad \forall v \in H_0^1 \tag{9.9}$$

Newton's method is in the context of non-linear variational equations known as the Newton-Galerkin method, and to derive it for the weak form above we first write u as the sum

$$u = u^0 + \delta \tag{9.10}$$

where u^0 is a some known approximation of u, and δ is a correction. This gives us

$$(a(u^0 + \delta)\nabla(u^0 + \delta), \nabla v) = (f, v), \quad \forall v \in H_0^1 \tag{9.11}$$

Making a Taylor expansion of $a(u) = a(u^0 + \delta)$ around u^0 we get

$$a(u^0 + \delta) = a(u^0) + a'_u(u^0)\delta + \mathscr{O}(\delta^2) \tag{9.12}$$

Substituting this into (9.11) we have

$$((a(u^0) + a'_u(u^0)\delta + \mathscr{O}(\delta^2))\nabla(u^0 + \delta), \nabla v) = (f, v), \quad \forall v \in H_0^1 \tag{9.13}$$

Neglecting in particular the term $(a'_u(u^0)\delta\nabla\delta, \nabla v)$ and all other terms that are quadratic in δ, we end up with an equation for an approximate correction $\delta^0 \approx \delta$: find $\delta^0 \in H_0^1$ such that

$$(a(u^0)\nabla\delta^0 + a'_u(u^0)\delta^0\nabla u^0, \nabla v) = (f, v) - (a(u^0)\nabla u^0, \nabla v), \quad \forall v \in H_0^1 \tag{9.14}$$

Once we have found δ^0 the Newton-Galerkin method is then to set $u^1 = u^0 + \delta^0$ and iterate, starting with the new solution guess u^1.

9.3.2 Finite Element Approximation

Let $\mathscr{K} = \{K\}$ be a mesh of Ω, and let $V_{h,0} \subset H_0^1$ be the usual space of continuous piecewise linears on $\mathscr{K}$. Replacing H_0^1 with $V_{h,0}$ in the weak form we obtain the finite element approximation of (9.7): find $\delta_h^0 \in V_{h,0}$ such that

$$(a(u^0)\nabla\delta_h^0 + a'_u(u_h^0)\delta_h^0\nabla u_h^0, \nabla v) = (f, v) - (a(u^0)\nabla u^0, \nabla v), \quad \forall v \in V_{h,0} \tag{9.15}$$

Here, we have tacitly assumed that $u^0 = u_h^0$ is a function in the finite element space $V_{h,0}$.

The finite element method (9.15) is equivalent to

$$(a(u^0)\nabla\delta_h + a'_u(u_h^0)\delta_h\nabla u_h^0, \nabla\varphi_i) = (f,\varphi_i) - (a(u^0)\nabla u^0, \nabla\varphi_i), \quad i = 1,\dots,n_i \tag{9.16}$$

where $\{\varphi_i\}_1^{n_i}$ is the usual set of hat functions which forms a basis for $V_{h,0}$. Further, writing δ_h as the sum

$$\delta_h = \sum_{j=1}^{n_i} d_j\varphi_j \tag{9.17}$$

and inserting into (9.15) we get

$$\sum_{j=1}^{n_i} d_j(a(u_h^0)\nabla\varphi_j + a'_u(u_h^0)\varphi_j\nabla u_h^0, \nabla\varphi_i) = (f,\varphi_i) - (a(u_h^0)\nabla u^0, \nabla\varphi_i), \quad i = 1,\dots,n_i \tag{9.18}$$

which is a system of n_i linear equations for the n_i unknown coefficients d_j. Indeed, in matrix form we write this

$$Jd = r \tag{9.19}$$

where J is the $n_i \times n_i$ Jacobian matrix with entries

$$J_{ij} = (a(u_h^0)\nabla\varphi_j, \nabla\varphi_i) + (a'_u(u_h^0)\varphi_j\nabla u_h^0, \nabla\varphi_i), \quad i,j = 1,\dots,n_i \tag{9.20}$$

and r is the $n_i \times 1$ residual vector with entries

$$r_i = (f,\varphi_i) - (a(u_h^0)\nabla u_h^0, \nabla\varphi_i), \quad i = 1,\dots,n_i \tag{9.21}$$

We can now formulate a discrete Newton-Galerkin method.

Algorithm 24 Newton-Galerkin Method for the Non-linear Poisson Equation

```
1: Choose a starting guess $u_h^0 \in V_{h,0}$, and a desired tolerance $\varepsilon$.
2: for $k = 1, 2, 3, \dots$ do
3:     Assemble the Jacobian matrix $J^k$ and the residual vector $r^k$ with entries
           $$J_{ij}^k = (a(u_h^k)\nabla\varphi_j, \nabla\varphi_i) + (a_u'(u_h^k)\varphi_j\nabla u_h^k, \nabla\varphi_i) \quad (9.22)$$
           $$r_i^k = (f, \varphi_i) - (a(u_h^0)\nabla u_h^0, \nabla\varphi_i) \quad (9.23)$$
4:     Solve the linear system
           $$J^k d^k = r^k \quad (9.24)$$
5:     Set $u_h^{k+1} = u_h^k + \delta_h^k$.
6:     if $\|\delta_h^k\| < \varepsilon$ then
7:         Stop.
8:     end if
9: end for
```

Here, we terminate the iteration process when the correction δ_h^k is small, which indicates that the iteration error $u_h^{k+1} - u_h^k$ is small, but we could equally well stop iterating when the residual r^k is small, which would indicate that the equation is well satisfied by u_h^k. Both these termination criteria are natural and it does not matter which one is used.

In practice, the assembly of the Jacobian matrix is simplified by using mass lumping, that is, replaced by a diagonal matrix containing the origonal matrix row sums. Since $\sum_j \varphi_j = 1$ we have the approximation

$$(a_u'(u_h^k)\varphi_j\nabla u_h^k, \nabla\varphi_i) \approx \delta_{ij}(a_u'(u_h^k)\nabla u_h^k, \nabla\varphi_i) \tag{9.25}$$

where δ_{ij} is 1 if $i = j$ and 0 otherwise. As a consequence, if A and b are the usual stiffness matrix and load vector with entries $A_{ij}^{(a)} = (a\nabla\varphi_j, \nabla\varphi_i)$, $i, j = 1, \dots, n_i$, and $b_i = (f, \nabla\varphi_i)$, $i = 1, \dots, n_i$, then

$$J^k \approx \operatorname{diag}(A^{(a_u')}u^k) + A^{(a)} \tag{9.26}$$

and

$$r^k = b - A^{(a)}u^k \tag{9.27}$$

where u^k is the $n_i \times 1$ vector of nodal values for u_h^k.

9.3.3 Piccard Iteration as a Simplified Newton Method

The computation of the Jacobian is numerically costly and we would like to simplify this as much as possible. We have already done so by using mass lumping to obtain the approximate Jacobian $J^k \approx \text{diag}(A^{(a'_u)}u^k) + A^{(a)}$. However, a more brutal approximation is to omit the diagonal matrix all together. This gives the simplified Newton iteration

$$\begin{aligned} u^{k+1} &= u^k + d^k && (9.28)\\ &= u^k + {J^k}^{-1} r^k && (9.29)\\ &= u^k + {A^{(a)}}^{-1}(b - A^{(a)}u^k) && (9.30)\\ &= {A^{(a)}}^{-1} b && (9.31)\end{aligned}$$

We recognize this as Piccard iterations on $A^{(a)}u = b$. Recall that this non-linear system of equations is precisely what one gets when applying finite elements to the original non-linear Poisson equation $-\nabla \cdot (a(u)\nabla u) = f$. Thus, Piccard iteration can be seen as a simplified Newton method in which the Jacobian J is approximated by the stiffness matrix $A^{(a)}$. Of course, this method will work only for very mildly non-linearities.

9.3.4 Computer Implementation

Below we present a MATLAB code for assembling the Jacobian matrix (9.26) and the residual vector (9.27). The computation of the derivative a'_u is done using numeric differentiation.

```
function [J,r] = JacRes(p,e,t,u,Afcn,Ffcn)
i=t(1,:); j=t(2,:); k=t(3,:); % triangle vertices
xc=(p(1,i)+p(1,j)+p(1,k))/3; % triangle centroids
yc=(p(2,i)+p(2,j)+p(2,k))/3;
% Evaluate u, a, a', and f.
tiny=1.e-8;
uu=(u(i)+u(j)+u(k))/3;
aa=Afcn(uu); % a(u)
da=Afcn(uu+tiny); % a(u+tiny)
da=(da-aa)/tiny; % da(u)/du
ff=Ffcn(xc,yc);
% Assemble Jacobian and residual.
[Aa ,unused,b]=assema(p,t,aa',0,ff);
[Ada,unused]  =assema(p,t,da',0,0);
J=diag(Ada*u)+Aa; % Jacobian
r=b-Aa*u; % residual
```

```
% Enforce zero Dirichlet BC.
fixed=unique([e(1,:) e(2,:)]); %  boundary nodes
for i=1:length(fixed)
  n=bdry(i); % a boundary node
  J(n,:)=0; % zero out row n of the Jacobian, J
  J(n,n)=1; % set diagonal entry J(n,n) to 1
  r(n)=0;   % set residual entry r(n) to 0
end
```

Input to this routine is the usual point, edge, and triangle matrices `p`, `e`, and `t`, and a vector `u` containing the nodal values of the current approximation u^k. The coefficients a and f are assumed to be defined by two separate subroutines `Afcn` and `Ffcn` defined elsewhere and passed via function handles. Output is the assembled Jacobian matrix J^k and the residual vector r^k. The actual assembly of the necessary matrices are done with `assema`.

As a numerical experiment let us compute the finite element solution to the non-linear Poisson equation (9.1) on the unit square $\Omega = [0,1]^2$ with $a(u) = 0.125 + u^2$, and $f = 1$. The necessary code is listed below.

```
function NewtonPoissonSolver()
g=Rectg(0,0,1,1);
[p,e,t]=initmesh(g,'hmax',0.05); % create mesh
u=zeros(size(p,2),1); % initial guess
for k=1:5 % non-linear loop
  [J,r]=JacRes(p,e,t,u,@Afcn,@Ffcn); % assemble J and r
  d=J\r; % solve for correction
  u=u+d; % update solution
  sprintf('|d|=%f, |r|=%f', norm(d), norm(r))
end
pdesurf(p,t,u)

function z = Afcn(u)
z=0.125+u.^2;

function z = Ffcn(x,y)
z=1;
```

In Figure 9.1 we show the computed solution u_h. Due to the non-linearity u_h is flatter on the top and has steeper gradients near the boundary than in the linear case $a = 0.125$.

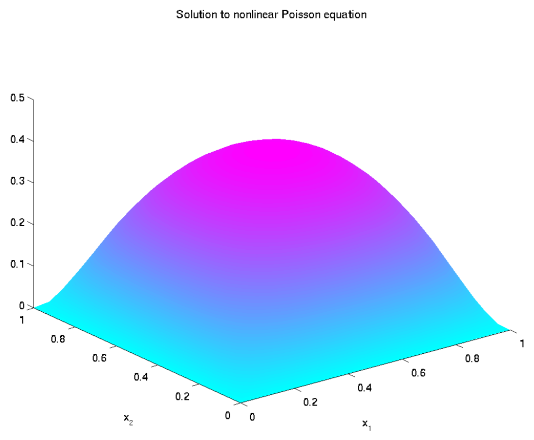


Fig. 9.1 Computed solution to the non-linear Poisson equation $-\nabla \cdot ((0.125 + u^2)\nabla u) = 1$ on the unit square $\Omega = [0,1]^2$, with zero boundary conditions.

At each Newton step k we monitor the 2-norm of the vectors d^k and the r^k holding the nodal values of the correction and the residual. Table 9.1 shows these numbers. Clearly the method converges, but not with a quadratic rate of convergence. The reason for this is the cheat with mass lumping when assembling the Jacobian.

k	$\|d^k\|$	$\|r^k\|$
1	8.767519	0.038330
2	1.061985	0.038821
3	0.886889	0.009474
4	0.285174	0.003118
5	0.048836	0.000787

Table 9.1 Norm of correction and residual in each Newton step.

9.4 The Bistable Equation

Instead of deriving a Newton method by first linearizing the continuous problem and then discretizing with finite elements, there is of course also the possibility of doing these things in reverse order. That is, applying Newton's method after finite element discretization. For completeness let us do this on the following equation called the Bistable equation.

$$\dot{u} - \varepsilon \Delta u = u - u^3, \quad \text{in } \Omega \times I \tag{9.32a}$$
$$n \cdot \nabla u = 0, \quad \text{in } \partial\Omega \times I \tag{9.32b}$$
$$u = u_0, \quad \text{in} \in \Omega, \text{ for } t = 0 \tag{9.32c}$$

Here, $\varepsilon > 0$ is a small number, $I = (0, T]$ is the time interval, and u_0 a given initial condition. Obviously, this is a non-linear equation due to the the cubic term u^3.

9.4.1 Weak Form

The weak form of (9.32) reads: find u such that for every fixed time t, $u \in H^1$ and

$$(\dot{u}, v) + \varepsilon(\nabla u, \nabla v) = (f(u), v), \quad \forall v \in H^1 \tag{9.33}$$

where $f(u) = u - u^3$.

9.4.2 Space Discretization

As always for transient problems we make the space discrete ansatz

$$u_h = \sum_{j=1}^{n_p} \xi_j(t)\varphi_j \tag{9.34}$$

where φ_j, $j = 1, 2, \dots, n_p$, are the usual hat basis functions of V_h and n_p the number of mesh nodes.

Substituting u_h into (9.33) and choosing $v = \varphi_i$, $i = 1, 2, \dots, n_p$, we get a system of n_p

$$\sum_{j=1}^{n_p} \dot{\xi}_j(\varphi_j, \varphi_i) + \varepsilon \sum_{j=1}^{n_p} \xi_j(\nabla\varphi_j, \nabla\varphi_i) = (f(\xi)), \varphi_i) \tag{9.35}$$

In matrix notation we write this

$$M\dot{\xi} + A\xi = b(\xi) \tag{9.36}$$

where M is the mass matrix, A is the stiffness matrix, and b a non-linear load vector, with entries

$$M_{ij} = (\varphi_j, \varphi_i) \tag{9.37}$$
$$A_{ij} = \varepsilon(\nabla\varphi_j, \nabla\varphi_j), \tag{9.38}$$
$$b_i(\xi) = (f(U(\xi)), \varphi_i) \tag{9.39}$$

9.4.3 Time Discretization

Applying backward Euler on the ODE system (9.36) we get the following time stepping scheme

$$M\frac{\xi_l-\xi_{l-1}}{k_l}+A\xi_l=b(\xi_l) \tag{9.40}$$

or equivalently

$$(M+k_lA)\xi_l=M\xi_{l-1}+k_lb(\xi_l) \tag{9.41}$$

We must now solve this non-linear system of equations using either Piccard iteration or Newton's method.

9.4.4 Piccard Iteration

Applying Piccard iteration to (9.41) yields

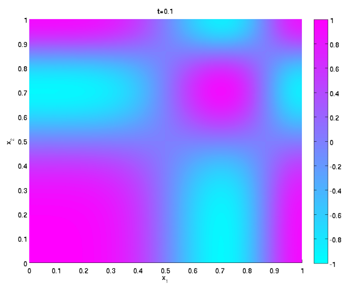
$$\xi_l^k=(M+k_lA)^{-1}(M\xi_{l-1}+k_lb(\xi_l^{k-1})) \tag{9.42}$$

This iteration scheme has the structure of a double for loop over the indices l and k. The outer loop evolves time and the loop index l counts the discrete time steps. For each time step l we have to solve the non-linear problem (9.41), and the loop index k keeps track of the Piccard iterates ξ_l^k and ξ_l^{k+1} needed for doing so. The natural choice for ξ_l^0 is the solution ξ^{l-1} from the previous time step. Once a new solution has been computed in the inner loop ξ^{l-1} is overwritten by ξ_l^{k+1} and the outer time loop is incremented one step. The double for loop is clearly seen in the code below, which solves the Bistable equation (9.32) on the unit square $\Omega=[0,1]^2$ with the parameter $\varepsilon=0.01$ and the initial condition $u_0=\cos(2\pi x_1^2)\cos(2\pi x_2^2)$.

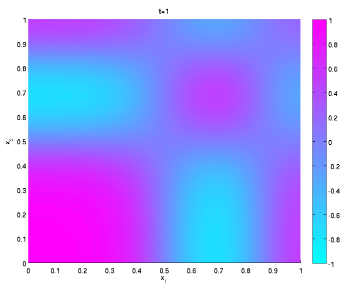
```
function PiccardBiStableSolver()
g=Rectg(0,0,1,1);
[p,e,t]=initmesh(g,'hmax',0.025);
x=p(1,:)'; y=p(2,:)';
xi_old=cos(2*pi*x.^2).*cos(2*pi*y.^2); % IC
xi_new=xi_old;
dt=0.1; % time step
epsilon=0.01;
[A,M]=assema(p,t,1,1,0);
for l=1:100 % time loop
  for k=1:3 % non-linear loop
    xi_tmp=xi_new;
    b=M*(xi_tmp-xi_tmp.^3);
```

```
      xi_new=(M+dt*epsilon*A)\(M*xi_old+dt*b);
      fixpterror=norm(xi_tmp-xi_new)
    end
    xi_old=xi_new;
    pdesurf(p,t,xi_new)
    axis([0,1,0,1,-1,1]), caxis([-1,1]), pause(.1);
  end
```

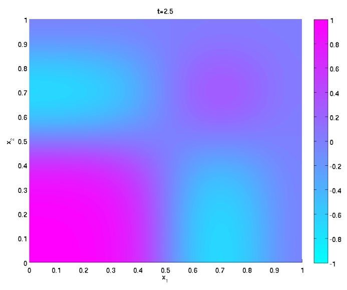
In Figure 9.2 we show a few snapshots of the finite element solution u_h at various times. The Bistable equation is a little peculiar because it has three steady states, $u = \pm 1$ and $u = 0$. The first two of these are stable, while the third is unstable. As a consequence there is always a struggle between regions where the solution is 1 and regions where it is -1. In the end, however, one of these will win and the solution will always end up being constnat and either 1 or -1. Which of these states it will be is somewhat random and depends on the parameter ε, and in the discrete setting also the mesh size h, and the time step k_l. Indeed, fron the figure we see that the final solution at $t = 25$ is constant -1.



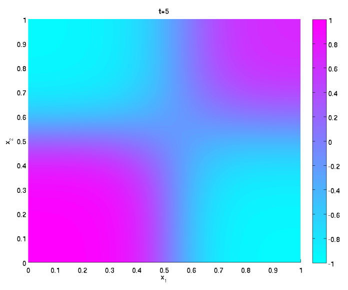
(a) $t = 0.1$



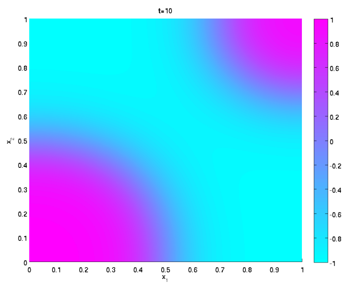
(b) $t = 1$



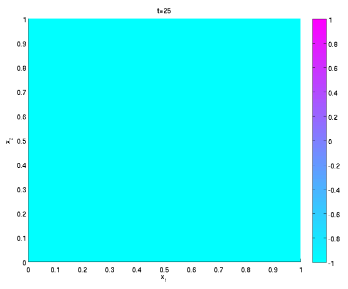
(c) $t = 2.5$



(d) $t = 5$



(e) $t = 20$



(f) $t = 25$

Fig. 9.2 Snapshots of the computed solution to the Bistable equation.

9.4.5 Newton's Method

We end this chapter by also deriving Newton's method for the time stepping scheme (9.41). In order to do so we recast this non-linear system of equations into the equivalent form

$$\rho(\xi_l) = 0 \tag{9.43}$$

where ρ is defined by

$$\rho(\xi_l) = (M + k_l A)\xi_l - M\xi_{l-1} - k_l b(\xi_l) \tag{9.44}$$

As we know by now Newton's method consists of taking a first rough guess ξ_l^0 at the solution ξ_l, and then repeatedly solving the linearized equations

$$J(\xi_l^{k+1} - \xi_l^k) = -\rho(\xi_l^k) \tag{9.45}$$

where the Jacobian matrix J has the entries $J_{ij} = \partial\rho_i/\partial(\xi_l)_j$, $i, j = 1, 2, \dots, n_p$.

To compute J, we note that the first term in the left hand side of (9.44) is easy to differentiate with respect to ξ_l. We have

$$\frac{\partial((M + k_l A)\xi_l)_i}{\partial(\xi_l)_j} = M_{ij} + k_l A_{ij} \tag{9.46}$$

Further, the second term, $M\xi_{l-1}$, does not depend on ξ_l, and so its derivative is therefore identically zero. The third term, though, is a bit complicated to differentiate. To do so we use the chain rule as follows

$$\frac{\partial b(\xi_l)_i}{\partial \xi_j} = \frac{\partial}{\partial \xi_j}\int_\Omega f(\xi_l)\varphi_i\,dx = \int_\Omega \frac{\partial f}{\partial u_h}\frac{\partial u_h}{\partial \xi_j}\varphi_i\,dx = \int_\Omega \frac{\partial f}{\partial u}\varphi_j\varphi_i\,dx \tag{9.47}$$

since $\partial f/\partial u_h = \partial f/\partial u$. This is just a mass matrix $M^{(f')}$ with the coefficient $\partial f/\partial u$. Thus, the Jacobian J is given by

$$J = (M + k_l A) - k_l M^{(f')} \tag{9.48}$$

The MATLAB implementation of Newton's method described above takes the following form.

```
function NewtonBiStableSolver()
g=Rectg(0,0,1,1);
[p,e,t]=initmesh(g,'hmax',0.02);
x=p(1,:); y=p(2,:);
xi_old=(cos(2*pi*x.^2).*cos(2*pi*y.^2))';
xi_new=xi_old;
dt=0.1; % time step
epsilon=0.01;
[A,M]=assema(p,t,1,1,0);
for l=1:100 % time loop
  for k=1:3 % non-linear loop
    ii=t(1,:); jj=t(2,:); kk=t(3,:);
    xi_tmp=xi_new; % copy temporary solution to new
    xi_tmp_mid=(xi_tmp(ii)+xi_tmp(jj)+xi_tmp(kk))/3;
    f =(xi_tmp_mid-xi_tmp_mid.^3); % evaluate f
    df=1-3*xi_tmp_mid.^2; % evaluate derivative df of f
```

```
      [crap,Mdf,b]=assema(p,t,0,df',f');
      J=(M+dt*epsilon*A)-dt*Mdf; % Jacobian
      rho=(M+dt*epsilon*A)*xi_new ...
        -M*xi_old-dt*b; % residual
      xi_new=xi_tmp-J\rho; % Newton update
      error=norm(xi_tmp-xi_new)
    end
    xi_old=xi_new; % copy old solution to new
    pdesurf(p,t,xi_new)
    axis([0 1 0 1 -1 1]), caxis([-1,1]), pause(.25)
  end
```

9.5 Problems

Exercise 9.1. Derive Newton's method for the non-linear problems $-\Delta u = u + u^3$, $-\Delta u + \sin(u) = f$, and $-\nabla \cdot ((1+u^2)\nabla u) = f$ with homogeneous Dirichlet boundary conditions $u = 0$.

Exercise 9.2. Use `NewtonPoissonSolver` to solve the non-linear Poisson equation (9.8) on the unit square with $a = 0.1 + u^2$ and $f = 1$. Study the influence of the non-linear term u^2 by comparing with the linear case $a = 0.1$ and $f = 1$. Study in particular the shape of the computed solutions.

Exercise 9.3. Verify numerically that the assumption $a > 0$ is necessary for existence of a solution by trying to solve non-linear Poisson equation (9.8) with $a = \varepsilon + u^2$ for $\varepsilon = 1$, 0.1, 0.075, 0.05, and 0.01. You should find that the method breaks down already at $\varepsilon = 0.05$. This can be temporarily remedied by using a modified update formula of type $u^{k+1} = u^k + \alpha d^k$, where $0 < \alpha \leq 1$ is a (small) number, typically $\alpha = \varepsilon$. This is the damped Newton method. The introduction of α affects the rate of convergence and it thus takes more iterations to achieve a desired level of accuracy. Of course, even damping can not prevent the method from breaking down as ε becomes really small.

Exercise 9.4. Derive Newton's method for $-\Delta u = f(u)$ with $f(u)$ is a differentiable function of u. Assume $u = 0$ on the boundary.

Exercise 9.5. Modify `NewtonPoissonSolver` and solve the equation $-\Delta u = e^{-u}$ with $u = 0$ on the boundary using Newton's method. Use `assema` for assembly of the involved matrices and vectors.

Chapter 10
Transport Problems

Abstract In this chapter we study the important transport equation, which occurs in almost all applications in continuum mechanics. In particular, this equation models convective heat transport, that is, a situation where heat is transported by some external physical process, such as a air blown by a fan or a moving fluid, for instance. We analysis this equation in the abstract framework outlined and prove existence and uniqueness of the solution using the Lax-Milgram Lemma. To handle transport involving high convection and low diffusion we introduce the Galerkin Least Squares (GLS) method and discuss its basic features.

10.1 The Transport Equation

The transport equation takes the form

$$-\varepsilon\Delta u + b\cdot\nabla u = f, \quad \text{in } \Omega \tag{10.1a}$$

$$u = 0, \quad \text{on } \partial\Omega \tag{10.1b}$$

where ε is a small parameter, b a given vector field and f is a given function.

Loosely speaking each of the two operators $-\varepsilon\Delta$, and $b\cdot\nabla$ play a specific role for the solution u, and can each be given a simple interpretation. The first smears u proportionally to ε, while the second transports u in the direction of the vector b. Therefore we say that these operators model the physical processes of diffusion, and convection, respectively. In fact, the transport equation is sometimes called the Convection-Diffusion equation.

For this problem to be well-posed we must require that $\nabla\cdot b = 0$.

For simplicity we shall perform the analysis using homogeneous Dirichlet conditions. However, other types of boundary conditions are of course possible. Indeed, for the numerical experiments we shall use both Neumann and Robin conditions.

10.1.1 Weak Form

The weak form of the transport equation (10.1) reads: find $u \in V_0 = H^1_0(\Omega)$ such that

$$a(u,v) = l(v), \quad \forall v \in V_0 \tag{10.2}$$

where the bilinear and linear forms $a(\cdot,\cdot)$ and $l(\cdot)$ are given by

$$\begin{aligned} a(u,v) &= \varepsilon(\nabla u, \nabla v) + (b \cdot \nabla u, v) && (10.3)\\ l(v) &= (f,v) && (10.4)\end{aligned}$$

10.1.2 Existence and Uniqueness of the Solution

The bilinear form $a(\cdot,\cdot)$ is bounded and coercive on V_0. To see this note that

$$\begin{aligned} a(u,v) &\le \varepsilon\|\nabla u\|\|\nabla v\| + \max b\|\nabla u\|\|v\| && (10.5)\\ &\le C(\|\nabla u\|\|\nabla v\| + \|\nabla u\|\|v\|) && (10.6)\\ &\le C\|\nabla u\|\|\nabla v\| && (10.7)\end{aligned}$$

due to the Poincare inequality $\|v\| \le C\|\nabla v\|$. This proves that $a(\cdot,\cdot)$ is bounded on V. To prove that $a(\cdot,\cdot)$ also is coercive we first notice that from the chain rule we have

$$(\nabla \cdot (bv^2), 1) = (\nabla \cdot b, v^2) + 2(b \cdot \nabla v, v) = 2(b \cdot \nabla v, v) \tag{10.8}$$

since by assumption $\nabla \cdot b = 0$. Here, the first integral is zero, which follows from Green's formula

$$(\nabla \cdot (bv^2), 1) = (b \cdot n, v^2)_{\partial\Omega} = 0 \tag{10.9}$$

and the fact that $v = 0$ on $\partial\Omega$. Using this result, we then have

$$\begin{aligned} a(v,v) &= \varepsilon(\nabla v, \nabla v) + (b \cdot \nabla v, v) + (cv, v) && (10.10)\\ &= \varepsilon(\nabla v, \nabla v) + (b \cdot \nabla v, v) && (10.11)\\ &\ge \varepsilon\|\nabla v\|^2 && (10.12)\end{aligned}$$

which shows that $a(\cdot,\cdot)$ is coercive on V. Also $l(\cdot)$ is bounded since we trivially have $l(v) \le \|f\|\|v\| \le C\|\nabla v\|$. Thus, invoking the Lax-Milgram Lemma we conclude that there exist a unique solution u to (10.2).

10.1.3 Standard Finite Element Approximation

To formulate a numerical method let $\mathscr{K} = \{K\}$ be a mesh of Ω and let $V_{h,0} \subset V_0$ be the space of continuous piecewise linears on $\mathscr{K}$ that vanish on the boundary. The standard finite element approximation of (10.2) then reads: find $u_h \in V_{h,0}$ such that

$$a(u_h, v) = l(v), \quad \forall v \in V_{h,0} \tag{10.13}$$

10.1.4 Computer Implementation

Let $\{\varphi_i\}_{i=1}^{n_i}$ be the usual hat function basis for V_h with n_i the number of interior nodes. Expanding the finite element ansatz $u_h = \sum_{i=1}^{n_i} \xi_j$ and choosing $v = \varphi_i$, $i = 1, 2, \dots, n_i$ in the finite element method (10.13) we obtain the following linear system

$$(A + C)\xi = b \tag{10.14}$$

where the matrix and vector entries are given by

$$A_{ij} = \varepsilon(\nabla\varphi_j, \varphi_i) \tag{10.15}$$

$$C_{ij} = (b \cdot \nabla\varphi_j, \varphi_i) \tag{10.16}$$

$$b_i = (f, \varphi_i) \tag{10.17}$$

with $i, j = 1, 2, \dots, n_i$.

The diffusion (i.e., stiffness) matrix A, and load vector b can be assembled using the built-in `assema` routine for instance. However, we have no routine to assemble the convection matrix C. To write such a routine we observe that the element convection matrix is approximately given by

$$C_{ij}^K = (b \cdot \nabla\varphi_j, \varphi_i)_K = b(x_c) \cdot [b_j, c_j]^T (\varphi_i, 1)_K = b(x_c) \cdot [b_j, c_j]^T |K|/3, \quad i, j = 1, 2, 3 \tag{10.18}$$

where $\nabla\varphi_j = [b_j, c_j]^T$ is the gradient of hat function φ_j, and x_c the centroid of K. This immediately translates into a assembly routine for C.

```
function C = ConvMat2D(p,t,bx,by)
np=size(p,2);
nt=size(t,2);
C=sparse(np,np);
for i=1:nt
  loc2glb=t(1:3,i);
  x=p(1,loc2glb);
  y=p(2,loc2glb);
  [area,b,c]=Gradients(x,y);
```

```
    bxmid=mean(bx(loc2glb));
    bymid=mean(by(loc2glb));
    CK=ones(3,1)*(bxmid*b+bymid*c)'*area/3;
    C(loc2glb,loc2glb)=C(loc2glb,loc2glb)+CK;
  end
```

Input us the usual point and triangle matrix p and t and the components bx and by of the convection field $b = [b_1, b_2]$. The components are and given as two $n_p \times 1$ vectors of nodal values with n_p the total number of nodes. Output is the assembled global convection matrix C.

A main routine for solving the transport equation $-\varepsilon\Delta u + [1,1]^T \cdot \nabla u = 1$ on the square $\Omega = [-1,1]^2$ with homogeneous Dirichlet boundary conditions is listed below.

```
  function TransportSolver()
  epsilon=0.1; % diffusion parameter
  [p,e,t]=initmesh('squareg','hmax',0.05); % create mesh
  np=size(p,2); % number of nodes
  [A,crap,b]=assema(p,t,1,0,1); % diffusion and load
  C=ConvMat2D(p,t,ones(np,1),ones(np,1)); % convection
  fixed=unique([e(1,:) e(2,:)]); % boundary nodes
  free=setdiff([1:np],fixed); % interior nodes
  b=b(free); % modify load
  A=A(free,free); C=C(free,free); % modify stiffness
  U=zeros(np,1); % solution vector
  U(free)=(epsilon*A+C)\b; % solve for free node values
  pdesurf(p,t,U) % plot
```

Running the code with $\varepsilon = 0.1$ we get the results of Figure 10.1. Note how the finite element solution u_h is offset in the direction of $b = [1,1]^T$. This is more clearly seen in Figure 10.2, which shows isocontours. The compression of the isocontours seen in the upper right corner, where u_h must bend downwards to satisfy the Dirichlet boundary condition, is called a layer.

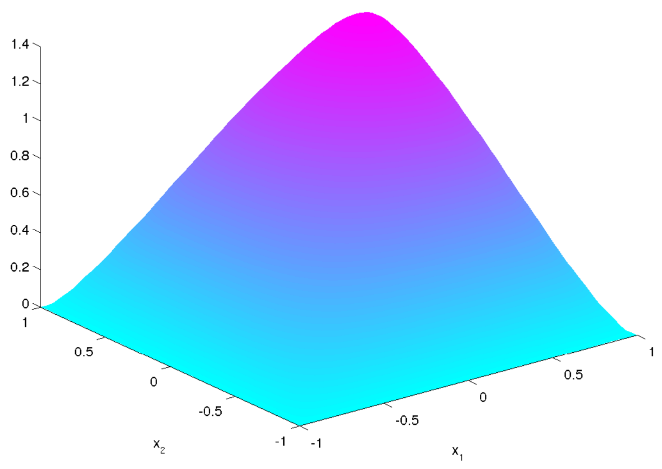


Fig. 10.1 Surface plot of u_h.

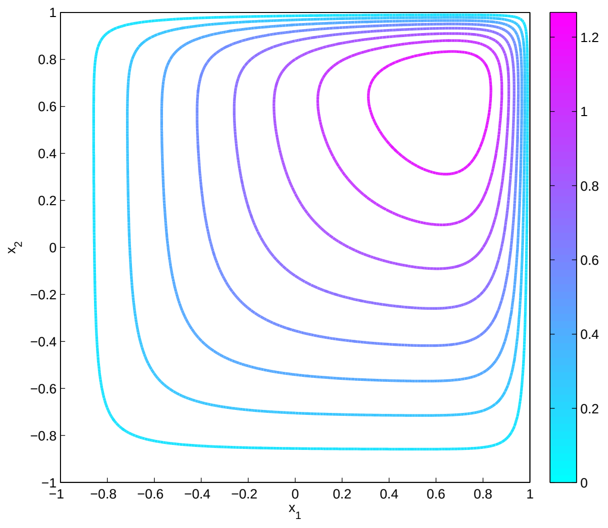


Fig. 10.2 Isocountours of u_h.

10.1.5 The Need for Stabilization

Using the Poincaré inequality $\|u\| \leq C\|\nabla u\|$ it is a simple task to derive the stability estimate

$$\sqrt{\varepsilon}\|\nabla u\| \leq C\|f\| \tag{10.19}$$

which also holds also for the finite element approximation u_h. From this we see that as ε decreases we loose control of the gradients of u. In other words small perturbations of f can lead to large local gradients. Indeed, this is a common feature of the solution which often have thin regions called layers where it changes rapidly. As we have already seen layers typically arise near a boundary where u must adhear to a Dirichlet boundary condition. However, there layers may also occur in the interior of the domain.

Standard finite element methods have great difficulties handling layers. In fact layers may trigger oscillations throughout the whole computational domain that renders the finite element approximation useless. Too see this it suffice to consider the transport in one dimension, say,

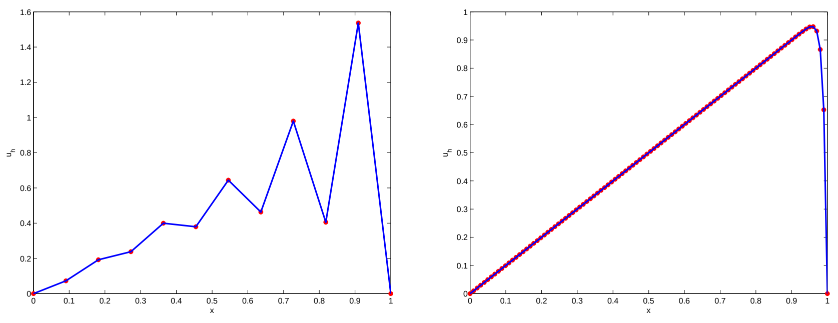
$$-\varepsilon u_{xx} + u_x = 1, \quad 0 < x < 1, \quad u(0) = u(1) = 0 \tag{10.20}$$

For small ε the exact solution to this equation looks like $u = x$ except near $x = 0$ where it drastically changes from 1 to 0 in order to satisfy the boundary condition $u(1) = 0$. This change takes place over a small distance of length proportional to ε and is therefore a layer.

Application of standard finite elements to 10.20 using a continuous piecewise linear approximation for u_h on a uniform mesh with $n+1$ nodes and mesh size h leads to the linear system

$$-\varepsilon \frac{u_{i+1} - 2u_i - u_{i-1}}{h^2} + \frac{u_{i+1} - u_{i-1}}{2h} = 1, \quad i = 1, 2, \dots, n-1 \tag{10.21}$$

where ξ_i are the nodal values of u_h with $\xi_0 = \xi_n = 0$. From this we see that if ε is small then information is only shared between every other node through the convective term. This opens up for the possibility of oscillations since node $i+1$ and $i-1$ talk with each other, but not with node i. Furthermore, in a layer we know that there is naturally a large variations between the node values ξ_i. Now, suppose that node $i-1$ has value $\xi_{i-1} = -1$ and node i has value $\xi_i = 1$. Then, due to the finite element method (10.21), and neglecting the unit load which has a small influence on a fine mesh, we will get $\xi_{i+1} = -1$, $\xi_{i+2} = 1$, $\xi_{i+3} = -1$ and so on. That is, a highly oscillatory finite element solution u_h. Figure 10.3 shows the finite element solution u_h for $\varepsilon = 0.01$ on two meshes with $n = 10$ and $n = 100$ elements, respectively.



(a) $n = 10$ (b) $n = 100$

Fig. 10.3 Illustration of oscillations due to under resolution of the mesh (a). Increasing the number of elements resolves the issue and yields a good finite element solution (b). Red asterisks denote node values.

We emphasize that oscillations might occur only if the diffusion parameter ε is smaller than the mesh size h. In this case the diffusion acts on a length scale below the mesh resolution. That is, the the solution can not be accurately represented on the mesh. Loosely speaking, this triggers the onset of oscillations.

The oscillatory behavior is present also in higher dimension.

10.1.6 Least-Squares Stabilization

The forming of layers and the inability of the standard finite element method to deal with these calls for modification of the numerical method. Since the oscillations are due to the small diffusion parameter ε a simple way of stabilizing is to add more diffusion. In doing so the general idea is to add as little as possible not do sacrifice accuracy, but as much as needed to obtain stability. A natural choice is to add h to ε. Then the stabilization will automatically decrease when using a finer mesh. This is known as isotropic stabilization. However, due to this $\mathscr{O}(h)$ perturbation of the equation such a method can only be first order accurate. It turns out that a better way to stabilize is to use a least squares stabilization which we shall describe in some detail next.

To explain the least squares stabilization technique let us consider the abstract equation

$$Lu = f \tag{10.22}$$

where L is a differential operator, u the sought solution, and f a given function. We do not worry about boundary conditions for the moment.

As we know by now, the weak form, or standard Galerkin method, is given by multiplying the equation by a test function v from a suitable space V and integrate.

This leads to the weak form: find $u \in V$ such that

$$(Lu, v) = (f, v), \quad \forall v \in V \tag{10.23}$$

We can interpret this as a demand for residual orthogonality $(r, v) = 0$ with r the residual $r = f - Lu$. A potential problem with this equation is if the product (Lu, v) does not define a norm on V, in which case it is not associated with a minimization principle. This can happen if (Lu, v) is not coercive or symmetric on V. In this case the numerical method resulting from finite element discretization might be unstable. As we have seen this is so for the transport equation.

Using instead the idea of orthogonal minimization we seek a solution $u \in V$, which is the minimizer of the problem

$$J(u) = \min_{v \in V} J(v) \tag{10.24}$$

where

$$J(v) = \|Lu - f\|_V^2 \tag{10.25}$$

The first order optimality condition for this optimization problem takes the form: find $u \in V$ such that

$$(Lu, Lv) = (f, Lv), \quad \forall v \in V \tag{10.26}$$

From linear algebra we recognize this as the normal equations of the Least Squares method.

An advantage of the Least Squares method is that the functional $J(\cdot)$, which is artificial in the sense that it is invented by us, does not need to have a physical interpretation, yet its minimizer almost always exist and is stable. Indeed, the bilinear form (Lu, Lv) is always symmetric and coercive.

Least squares methods are known to be very robust, only requiring a minimum of regularity of the underlying equation. However, they are often not very accurate.

The Galerkin Least Squares (GLS) method is obtained by combining the standard Galerkin and the Least Squares method. In effect this amounts to replacing the test function v by $v + \delta Lv$, where δ is a parameter to be chosen. In doing so, we obtain the variational equation: find $u \in V$ such that

$$(Lu, v + \delta Lv) = (f, v + \delta Lv), \quad \forall v \in V \tag{10.27}$$

or

$$(Lu, v) + \delta(Lu, Lv) = (f, v) + \delta(f, Lv), \quad \forall v \in V \tag{10.28}$$

Needless to say one hopes to combine the accuracy of the Galerkin method with the robustness of the Least Squares method.

10.1.7 GLS for the Transport Equation

The transport equation (10.1) can be written $Lu = f$ with $L = -\varepsilon\Delta + b\cdot\nabla$. As a consequence, the Galerkin Least Squares finite element approximation takes the form: find $u_h \in V_{h,0}$ such that

$$a(u_h, v) = l_h(v), \quad \forall v \in V_{h,0} \tag{10.29}$$

where the bilinear and linear form $a_h(\cdot,\cdot)$ and $l_h(\cdot)$ are defined by

$$a_h(u,v) = \varepsilon(\nabla u, \nabla v) + (b\cdot\nabla u, v) + \delta \sum_{K\in\mathscr{K}} (-\Delta u + b\cdot\nabla u, -\Delta v + b\cdot\nabla v) \tag{10.30}$$

$$l_h(v) = (f,v) + \delta\sum_K (f, -\Delta v + b\cdot\nabla v) \tag{10.31}$$

Note that we have written the stabilizing terms (Lu, Lv) and (f, Lv) as a sum over the elements K. This is due to the fact that the term $-\varepsilon\Delta v$ does not lie in $L^2(\Omega)$ since the second order derivatives of the test function v are unbounded across element boundaries. However, it does lie in $L^2(K)$ for all K. To obtain a well defined integrals we have therefore applied all GLS terms only within each element. This is a general feature of GLS methods.

We next observe that the exact solution u satisfies the GLS method, that is,

$$a_h(u,v) = l_h(v), \quad \forall v \in V_{h,0} \tag{10.32}$$

Subtracting (10.32) from (10.29) we immediately obtain the Galerkin orthogonality

$$a_h(u - u_h, v) = 0, \quad \forall v \in V_{h,0} \tag{10.33}$$

Because of this we say that the GLS method is consistent.

To measure the size of the error $e = u - u_h$ we define the following norm on V_0

$$|||v|||^2 = \varepsilon\|\nabla v\|^2 + \delta\|b\cdot\nabla v\|^2 \tag{10.34}$$

The reason for introducing this norm is twofold. First, we want to have a norm that is related to $a_h(\cdot,\cdot)$ to be able to use the consistency of this bilinear form. Second, if we can prove an error estimate in this norm, then we have regained at least partial control over the solution gradient through the term $\delta\|b\cdot\nabla u_h\|$. In other words if we have a bound on this term then it cannot be too small because of the stabilization parameter δ in front. Due to the fact that $b\cdot\nabla u$ is the the derivative along the streamline b this GLS method is sometimes called the streamline-diffusion method.

An important question to ask is if the GLS method we have formulated is void or if it leads to a well defined solution approximation u_h. This is indeed the case since $a_h(\cdot,\cdot)$ is coercive on $V_{h,0}$, that is,

$$a_h(v,v) \geq |||v|||^2, \quad \forall v \in V_{h,0} \tag{10.35}$$

To see this note that for any $v \in V_{h,0}$ we have

$$a_h(v,v) = \varepsilon\|\nabla v\|^2 + (b\cdot v, v) + \delta \sum_K \| - \varepsilon\Delta v + b\cdot\nabla v\|_K^2 \tag{10.36}$$

$$= \varepsilon\|\nabla v\|^2 + \|b\cdot\nabla v\|^2 \tag{10.37}$$

$$= |||v|||^2 \tag{10.38}$$

Note, however, that the coercivity depends on the stabilization parameter δ, which consequently must not be too small for problems with small ε. On the other hand if ε is big then we can practically let δ vanish. A good choice of δ turns out to be

$$\delta = \begin{cases} Ch^2, & \text{if } \varepsilon > h \\ Ch/\|b\|_\infty, & \text{if } \varepsilon < h \end{cases} \tag{10.39}$$

As we shall see this follows from the error analysis.

For the GLS method we have the a priori estimate

$$|||e||| \leq Ch^{3/2}\|D^2u\| \tag{10.40}$$

The proof of this is a bit technical and relies on writing the error $e = u - u_h = (u - \pi u) + (\pi u - u_h)$, where $\pi u \in V_{h,0}$ is the interpolant to u. The unspoken hope is that $u - \pi u$ is easy to handle with interpolation estimates and that $\pi u - u_h$ is also easy to handle since it is discrete function in $V_{h,0}$. Thus, if we can bound these terms individually the Triangle inequality then gives us $|||e||| \leq |||u - \pi u||| + |||\pi u - u_h|||$. The interested reader might ask why we cannot apply Cea's lemma as usual to derive an priori estimate. The answer is that we want to estimate the error in the $|||\cdot|||$ norm, which is related to $a_h(\cdot,\cdot)$ and not to $a(\cdot,\cdot)$.

Now, using the coercivity of $a_h(\cdot,\cdot)$ on $V_{h,0}$ we have

$$|||u_h - \pi u|||^2 \leq a_h(u_h - \pi u, u_h - \pi u) \tag{10.41}$$

$$= a_h(u_h - \pi u, u_h - \pi u) + a_h(u - u_h, u_h - \pi u) \tag{10.42}$$

$$= a_h(u - \pi u, u_h - \pi u) \tag{10.43}$$

where we have used the Galerkin orthogonality (10.33). Let us estimate each of the three terms in $a_h(u - \pi u, u_h - \pi u)$ separately using the trivial estimates $\sqrt{\varepsilon}\|\nabla v\| \leq |||v|||$ and $\sqrt{\delta}\|b\cdot\nabla v\| \leq |||v|||$. First, we have

$$\varepsilon(\nabla(u - \pi u), \nabla(u_h - \pi u)) \leq \sqrt{\varepsilon}\|\nabla(u - \pi u)\|\sqrt{\varepsilon}\|\nabla(u_h - \pi u)\| \tag{10.44}$$

$$\leq C\sqrt{\varepsilon}h\|D^2u\||||u_h - \pi u||| \tag{10.45}$$

Then, using integration by parts and again that $\nabla\cdot b = 0$ we have

$$(b \cdot \nabla(u - \pi u), u_h - \pi u) = -(u - \pi u, \nabla \cdot (b(u_h - \pi u))) \tag{10.46}$$
$$\leq \|u - \pi u\| \|\nabla \cdot b(u_h - \pi u)\| \tag{10.47}$$
$$\leq Ch^2 \|D^2 u\| |||u_h - \pi u|||/\sqrt{\delta} \tag{10.48}$$

Finally, we have

$$\delta \sum_K (L(u - \pi u), L(u_h - \pi u))_K \leq \sqrt{\delta} \left(\sum_K \|L(u - \pi u)\|_K^2 \right)^{1/2} |||u_h - \pi u||| \tag{10.49}$$

where

$$\|L(u - \pi u)\|_K^2 = \| - \varepsilon \Delta (u - \pi u) + b \cdot \nabla (u - \pi u)\|_K^2 \leq C(\varepsilon^2 + h^2) \|D^2 u\|_K^2 \tag{10.50}$$

since $\Delta(\pi u) = 0$, $\|\Delta u\|^2 \leq C\|D^2 u\|$, and $\|b \cdot \nabla(u - \pi u)\| \leq \|b\|_\infty \|\nabla(u - \pi u)\| \leq Ch\|D^2 u\|$.

Now, by definition transport with high convective effects means that $\varepsilon = Ch$ and consequently that $\delta = Ch$. Thus, all of the three terms (10.44), (10.46), and (10.49), above are of order $h^{3/2}$. Hence, we have

$$|||u_h - \pi u|||^2 \leq Ch^{3/2} \|D^2 u\| \tag{10.51}$$

It remains to estimate $|||u - \pi u|||$. However, repeating the above estimates it is easily seen that this term is also of order $h^{3/2}$. Thus, we infer the a priori estimate (10.40).

The assembly of the GLS stabilization term $(b \cdot \nabla u, b \cdot \nabla v)$ for piecewise linears is easy. A routine is listed below.

```
function Sd = SDMat2D(p,t,bx,by)
np=size(p,2);
nt=size(t,2);
Sd=sparse(np,np);
for i=1:nt
  loc2glb=t(1:3,i);
  x=p(1,loc2glb);
  y=p(2,loc2glb);
  [area,b,c]=Gradients(x,y);
  bxmid=mean(bx(loc2glb));
  bymid=mean(by(loc2glb));
  SdK=(bxmid*b+bymid*c)*(bxmid*b+bymid*c)'*area;
  Sd(loc2glb,loc2glb)=Sd(loc2glb,loc2glb)+SdK;
end
```

Input is the same as for the routine `ConvMat2D`.

10.1.8 Heat Transfer in a Fluid Flow

We now study a real-world application with more general boundary conditions, namely, heat transfer in a fluid flow. This kind of physical problem is of interest when designing heat exchangers or electronic devises for instance. To this end we consider a heated object submerged into a channel with a flowing fluid. See Figure 10.4. The channel is rectangular and fluid is flowing from left to right round a heated circle.

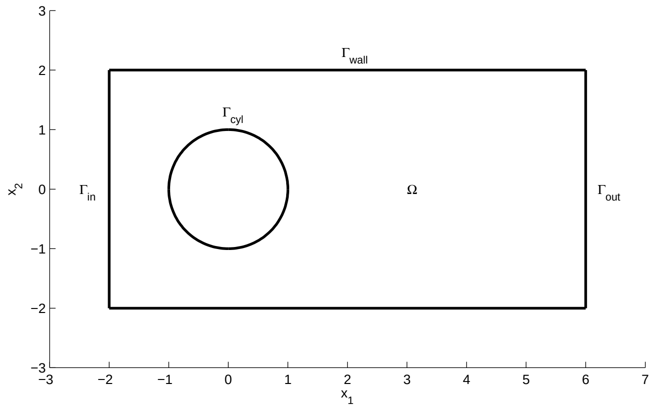


Fig. 10.4 Geometry of the channel domain and boundaries.

The fluid flow is given by the velocity field

$$b_1 = U_\infty \left(1 - \frac{x^2 - y^2}{(x^2+y^2)^2}\right) \tag{10.52}$$

$$b_2 = -2U_\infty \frac{xy}{(x^2+y^2)^2} \tag{10.53}$$

where $U_\infty = 1$ is the free stream velocity of the fluid. Figure 10.5 shows a glyph plot of b.

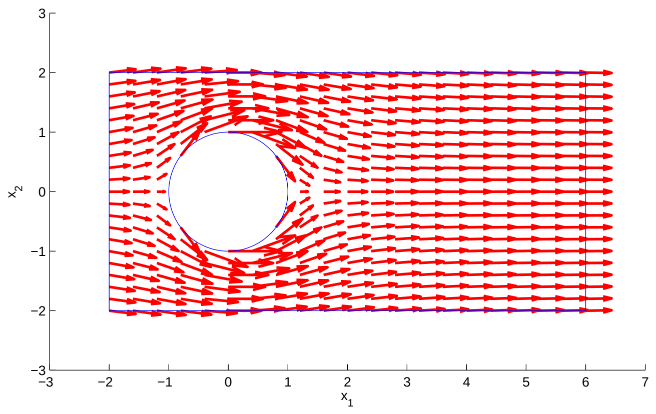


Fig. 10.5 Glyphs showing the fluid velocity field b.

For later use let us write a routine to evaluate the vector field b.

```
function [bx,by] = FlowField(x,y)
a=1; % cylinder radius
Uinf=1; % free stream velocity
r2=x.^2+y.^2; % radius vector squared
bx=Uinf*(1-a^2*(x.^2-y.^2)./r2.^2); % x-component of b
by=-2*a^2*Uinf*x.*y./r2.^2;         % y-
```

We assume that the cylinder is kept at constant temperature 1. Further, the walls of the channel are insulated so that no heat can flow across them. In other words, the normal heat flux $n \cdot q$ is zero on the walls, where q is given by the generalized Fourier's law

$$q = -\varepsilon \nabla u + bu \tag{10.54}$$

At the outflow diffusive effects are usually negligible which means that $\varepsilon n \cdot \nabla u = 0$. Finally, at the inflow the fluid has zero temperature.

All in all, we have the following transport equation and boundary conditions for the fluid temperature u.

$$-\varepsilon \Delta u + b \cdot \nabla u = 0, \quad \text{in } \Omega \tag{10.55a}$$
$$u = 0, \quad \text{on } \Gamma_{\text{in}} \tag{10.55b}$$
$$u = 1, \quad \text{on } \Gamma_{\text{cyl}} \tag{10.55c}$$
$$-\varepsilon n \cdot \nabla u = 0, \quad \text{on } \Gamma_{\text{out}} \tag{10.55d}$$
$$n \cdot (-\varepsilon \nabla u + bu) = 0, \quad \text{on } \Gamma_{\text{wall}} \tag{10.55e}$$

In order to simplify the computer implementation we first approximate the Dirichlet conditions (10.55b) and (10.55c) using the Robin conditions $-\varepsilon n \cdot \nabla u =$

$10^6 u$ on Γ_{in} and $-\varepsilon n \cdot \nabla u = 10^6(u-1)$ on Γ_{cyl}, respectively. Multiplying $-\varepsilon \Delta u + b \cdot \nabla u = 0$ by a test function v and integrating both the diffusive and convective term by parts, we then have

$$\begin{aligned} 0 &= \varepsilon(\nabla u, \nabla v) - \varepsilon(n \cdot \nabla u, v)_{\partial\Omega} - (u, b \cdot \nabla v) + (n \cdot bu, v)_{\partial\Omega} && (10.56) \\ &= \varepsilon(\nabla u, \nabla v) + 10^6 (u,v)_{\Gamma_{\text{in}}} + 10^6 (u-1, v)_{\Gamma_{\text{cyl}}} - (u, b\cdot\nabla v) + (n \cdot bu, v)_{\Gamma_{out}} && (10.57) \end{aligned}$$

As a consequence the weak form reads: find $u_h \in V = H^1$ such that

$$\begin{aligned} &\varepsilon(\nabla u, \nabla v) + 10^6 (u,v)_{\Gamma_{\text{in}}} + 10^6 (u,v)_{\Gamma_{\text{cyl}}} \\ &-(u, b\cdot\nabla v) + (n\cdot bu, v)_{\Gamma_{out}} = 10^6 (1,v)_{\Gamma_{\text{cyl}}}, \quad \forall v \in V \end{aligned} \qquad (10.58)$$

To approximate V let $V_h \subset V$ be the usual space of all continuous piecewise linears. Adding now the least squares term $\delta(b \cdot \nabla u, b \cdot \nabla v)$ to the weak form we obtain the finite element approximation: find $u_h \in V_h$ such that

$$\begin{aligned} &\varepsilon(\nabla u, \nabla v) + 10^6 (u,v)_{\Gamma_{\text{in}}} + 10^6 (u,v)_{\Gamma_{\text{cyl}}} \\ &-(u, b\cdot\nabla v) + (n\cdot bu, v)_{\Gamma_{out}} + \delta(b\cdot\nabla u, b\cdot\nabla v) = 10^6 (1,v)_{\Gamma_{\text{cyl}}}, \quad \forall v \in V_h \end{aligned} \qquad (10.59)$$

We observe that the left hand side boundary terms can be written $(\kappa u, v)_{\partial\Omega}$ with κ chosen as

$$\kappa = \begin{cases} 10^6, & \text{on } \Gamma_{\text{in}} \cup \Gamma_{\text{cyl}} \\ b \cdot n, & \text{on } \Gamma_{\text{out}} \\ 0, & \text{elsewhere} \end{cases} \qquad (10.60)$$

or, written as a MATLAB routine,

```
function k = Kappa(x,y)
k=0;
if x<-1.99 % inflow
  k=1e6;
end
if sqrt(x^2+y^2)<1.01 % cylinder
  k=1e6;
end
if x>5.99 % outflow
  [bx,by]=FlowField(x,y);
  nx=1; ny=0; % normal components
  k=bx*nx+by*ny; % kappa = dot(b,n)
end
```

We can now compute the left hand side boundary terms with a call to `RobinMat2D` with a function handle to `Kappa` as argument. Similarly, the right hand side boundary integral can be computed with a call to `RobinVec2D` with function handles to the following two routines as arguments.

```
function g = g_D(x,y)
g=0;
if sqrt(x^2+y^2)<1.01, g=1; end

function g = g_N(x,y)
g=0;
```

Finally, we notice that the convection matrix stemming from the term $-(u, b\cdot\nabla v)$ is just the negative transpose of the matrix assembled by the routine `ConvMat2D`.

Putting all pieces together we obtain the following main routine.

```
function HeatFlowSolver()
channel=RectCirc(); % channel geometry
epsilon=0.01; % diffusion parameter
h=0.1; % meshsize
[p,e,t]=initmesh(channel,'hmax',h); % create mesh
A=assema(p,t,1,0,0); % stiffness matrix
x=p(1,:); y=p(2,:); % node coordinates
[bx,by]=FlowField(x,y); % evaluate vector filed b
C =ConvMat2D(p,t,bx,by); % convection matrix
Sd=SDMat2D(p,t,bx,by); % GLS stabilization matrix
R =RobinMat2D(p,e,@Kappa); % RHS boundary terms
g =RobinVec2D(p,e,@Kappa,@g_D,@g_N); % LHS boundary term
delta=h; % stabilization parameter
U=(epsilon*A-C'+R+delta*Sd)\g; % solve linear system
pdecont(p,t,U), axis equal % plot solution
```

Note that the mesh size $h = 0.1$ while the diffusion parameter $\varepsilon = 0.01$, which can lead to potential problems with oscillations. However, by choosing the stabilization parameter δ proportional to h we get additional diffusion along the streamlines of b that prevents the solution u_h from oscillating. Running this code we get the result of Figure 10.6.

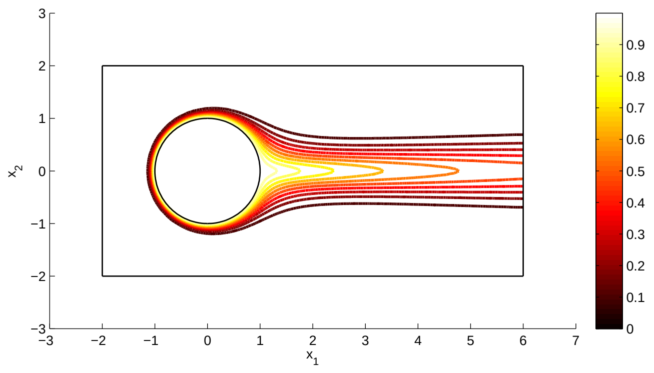


Fig. 10.6 Isocontours of the temperature u_h in the fluid.

From this figure it is clearly seen how the temperature behind the cylinder is transported downstream by the fluid flow, whereas a boundary layer is formed in front of the cylinder. As expected the temperature is decreasing downstream due to the artificial diffusion and no oscillations are visible. A nice detail visible is that the outflow appears transparent in the sense that the temperature isocontours seem unaffected by the domain boundary.

10.2 Problems

Exercise 10.1. Compute the least squares solution to the linear system $Ax = b$ with

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 2 \\ 6 \end{bmatrix}$$

Which norm is minimized by this solution?

Exercise 10.2. Show the stability estimate (10.19). *Hint:* look at the derivation of the coercvity for the bilinear form $a(\cdot,\cdot)$.

Exercise 10.3. Verify that a standard one-dimensional finite element method for the transport equation yields the linear system (10.21).

Exercise 10.4. Derive a GLS method for the problem

$$-\varepsilon\Delta u + b\cdot\nabla u + cu = f, \quad x \in \Omega, \qquad u = 0, \quad x \in \partial\Omega$$

Also, using a standard continuous piecewise linear approximation of the solution what does the linear system resulting from finite element discretization look like?

Exercise 10.5. Fill in the details of the a priori error estimate (10.40) for the GLS method.

Exercise 10.6. Use `TransportSolver` to verify that standard Galerkin is unstable also in higher dimensions. Choose the diffusion parameter $\varepsilon = 0.01$ and the mesh size $h = 0.05$, for example.

Chapter 11
Solid Mechanics

Abstract Arguably one of the most important areas of application for finite element methods is solid mechanics. Today, finite elements are used together with computed aided design (CAD) tools to optimize and speed up the design and manufacturing process of practically all mechanical structures, ranging from bridges to airplanes. In this chapter we derive the equations of linear elasticity and formulate finite element approximations of them. We do this in the abstract framework of elliptic partial differential equations and prove existence and uniqueness of the solution using the Lax-Milgram Lemma. A priori and a posteriori error estimates are also proved. Particular effort is laid on explaining the somewhat intricate implementation of the finite element method. We study several applications, including thermal stress and modal analysis.

11.1 Governing Equations

11.1.1 Cauchy's Equilibrium Equation

We shall now derive the partial differential equation governing linear elasticity. Because elastic deformation is a three dimensional phenomenon we must work in three dimensions and not two as we have done up till now.

The basic idea behind elastostatics is that the total force acting on any material volume must vanish.

There are two kinds of forces which can act on a volume $\Omega \in \mathbb{R}^3$. First there are forces penetrating the whole volume. These are described by a force density f. The most common example is gravity with $f = -\rho g$. Then there are contact forces which acts on the surface $\partial\Omega$. Even if they only act on the surface, contact forces are described by vector fields imagined to exist throughout the whole volume. A simple example is the pressure p within a fluid that acts along the normal to any real or imagined fluid surface and with a force proportional to the surface area.

The fundamental concept for describing contact forces is the stress tensor. The stress tensor is a 3×3 matrix σ, defined such that component σ_{ij}, $i,j = 1,2,3$, is the force per unit area acting in direction x_i on a surface with unit normal in direction x_j. Thus, the force on a small surface ds with unit normal n is given by $\sigma \cdot n\,ds$. The total force F on a volume Ω with surface $\partial\Omega$ is consequently the sum of volume and surface contributions

$$F = \int_{\Omega} f\,dx + \int_{\partial\Omega} \sigma \cdot n\,ds \tag{11.1}$$

Using the divergence theorem we may convert the surface integral over the surface into a volume integral.

$$F = \int_{\Omega} (f + \nabla\cdot\sigma)\,dx \tag{11.2}$$

where we have introduced the notation

$$(\nabla\cdot\sigma)_i = \sum_{j=1}^{3} \frac{\partial\sigma_{ij}}{\partial x_j}, \quad i = 1,2,3 \tag{11.3}$$

In equilibrium the total force F must vanish for any volume Ω and we thus infer

$$f + \nabla\cdot\sigma = 0 \tag{11.4}$$

This is Cauchy's equation of equilibrium. It says that the net force vanishes on every material particle throughout the volume. It can be thought of as a specialization of Newton's second law, which says that the net force on any material particle equals equals mass times acceleration.

Cauchy's equilibrium equation (11.4) consists of three differential equations. In component form they are given by

$$f_1 + \frac{\sigma_{11}}{\partial x_1} + \frac{\sigma_{12}}{\partial x_2} + \frac{\sigma_{13}}{\partial x_3} = 0 \tag{11.5a}$$

$$f_2 + \frac{\sigma_{21}}{\partial x_1} + \frac{\sigma_{22}}{\partial x_2} + \frac{\sigma_{23}}{\partial x_3} = 0 \tag{11.5b}$$

$$f_3 + \frac{\sigma_{31}}{\partial x_1} + \frac{\sigma_{32}}{\partial x_2} + \frac{\sigma_{33}}{\partial x_3} = 0 \tag{11.5c}$$

In order to obtain a closed set of equations it is necessary to supplement (11.5) with additional so-called constitutive equations, expressing the local relations between the stresses and the local state of matter. Different kinds of matter (i.e., gases, liquids, or solids) only differ by their constitutive equations. We shall see below how this is done for isotropic linear elastic solids and later also for incompressible viscous fluids.

The stress tensor σ has by default nine independent components. However, from conservation of angular momentum it follows that σ is be symmetric, that is,

$$\sigma = \sigma^T \tag{11.6}$$

which reduces the number of independent components to six.

11.1.2 Constitutive Equations and Hooke's Law

Any deformation of a material body may be described by specifying how each material particle within the body is displaced from its initial position. The displacement of a material particle is naturally defined as the vector $u = x - x_0$, where x is the current and x_0 the initial position of the particle.

The general displacement of a body includes translations and rotations that really should not be classified as deformations, since a true deformation is characterized by geometric changes within the body such as stretching, for example. The relevant quantity for describing deformation is the strain tensor, which under the assumption of small displacement gradients, is defined by

$$\varepsilon = \frac{1}{2}(\nabla u + \nabla u^T) \tag{11.7}$$

or, in component form,

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right), \quad i,j = 1,2,3 \tag{11.8}$$

A displacement u is defined as a deformation such that the strain tensor ε does not vanish everywhere. This is in contrast to translations and rotations, so-called rigid body modes, for which $\varepsilon = 0$.

Geometrically, the diagonal component ε_{ii} is the relative change in length along the x_i-axis, whereas the off-diagonal component ε_{ij} is proportional to the change in angle between the initially orthogonal coordinate axes x_i and x_j.

Pure translations and rotations should not create stresses, implying that the local stresses can only depend on the local strains. When the strains are small, it is reasonable to assume that the relation between the stress tensor and the strain tensor is approximately linear. This assumption is called Hooke's law and is a constitutive equation, meaning that it is not a law of nature, but rather deduced from empirical measurements and experiments.

In the most general case it takes a relation of the form $\sigma_{ij} = \sum_{kl} C_{ijkl}\varepsilon_{kl}$, where C_{ijkl} is a fourth order tensor with up to 36 independent components, or, elastic moduli, to characterize the most complex linear elastic material. In contrast isotropic materials (i.e., materials characterized by properties which are independent of spatial direction) only require two elastic moduli for their description.

Assuming that there are no stresses before any deformation of a body occupied by a linear elastic isotropic material, the stress and strain relation can for symmetry reasons only take the form

$$\sigma = 2\mu\varepsilon(u) + \lambda(\nabla\cdot u)I \tag{11.9}$$

where I is the 3×3 identity matrix. The elastic moduli μ and λ are called the Lamé parameters and are defined by

$$\mu = \frac{E}{2(1+\nu)} \qquad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \tag{11.10}$$

where E is Young's elastic modulus, and ν is Poisson's ratio.

Young's modulus is a material parameter that describes the stiffness of the material. Poisson's ratio is a measure of the tendency for a material to narrow its cross section when it is stretched. In principal E and ν may change throughout the material but for homogeneous materials they are constant.

Combining the equilibrium equation (11.4) with the constitutive equation (11.9) we get a system of two vector valued partial differential equations governing the displacement field u

$$-\nabla\cdot\sigma = f \tag{11.11a}$$

$$\sigma = 2\mu\varepsilon(u) + \lambda(\nabla\cdot u)I \tag{11.11b}$$

Alternatively, substituting (11.11a) into (11.11b) using the vector identity $\nabla\cdot(2\varepsilon) = \Delta u + \nabla(\nabla\cdot u)$ we eventually end up with a single vector valued partial differential equation for u called the Cauchy-Navier equation

$$f + \mu\Delta u + (\lambda+\mu)\nabla(\nabla\cdot u) = 0 \tag{11.12}$$

11.1.3 Boundary Conditions

To obtain a unique solution u, (11.12) must be supplemented by boundary conditions, which can be of the two standard types, Dirichlet and Neumann boundary conditions. Dirichlet boundary conditions are constraints on the displacements u and take the form $u = g_D$ where g_D is given function. Often $g_D = 0$ which corresponds to a situation where the material is clamped to the surroundings and unable to move at the boundary. Neumann boundary conditions are constraints on the normal stress and take the form $\sigma\cdot n = g_N$, where n is the outward unit normal on the boundary and g_N is a given traction load.

11.2 The Equations of Linear Elastostatics

Thus, the basic problem of linear elastostatics is to find the stress tensor σ and the displacement vector u such that

$$-\nabla\cdot\sigma = f, \qquad \text{in } \Omega \tag{11.13a}$$
$$\sigma = 2\mu\varepsilon(u) + \lambda(\nabla\cdot u)I, \qquad \text{in } \Omega \tag{11.13b}$$
$$u = 0, \qquad \text{on } \Gamma_D \tag{11.13c}$$
$$\sigma\cdot n = g_N, \qquad \text{on } \Gamma_N \tag{11.13d}$$

where Γ_D and Γ_N are two boundary segments associated with the Dirichlet and Neumann boundary conditions, respectively, and such that $\Gamma_D \cup \Gamma_N = \partial\Omega$ and $\Gamma_D \cap \Gamma_N = \emptyset$.

11.2.1 Variational Formulation

In order to derive a variational formulation of (11.13) let

$$V = \{v \in [H^1(\Omega)]^3 : v|_{\Gamma_D} = 0\} \tag{11.14}$$

Multiplying $-\nabla\cdot\sigma = f$ with a test function $v \in V$ and integrating by parts we have

$$(-\nabla\cdot\sigma, v) = \sum_{i,j=1}^{3} \left(-\frac{\partial\sigma_{ij}}{\partial x_j}, v_i\right) \tag{11.15}$$
$$= \sum_{i,j=1}^{3} -(\sigma_{ij}, n_j v_i)_{\partial\Omega} + \left(\sigma_{ij}, \frac{\partial v_i}{\partial x_j}\right) \tag{11.16}$$
$$= (f, v) \tag{11.17}$$

Introducing the contraction operator : defined by

$$A : B = \sum_{i,j=1}^{3} A_{ij}B_{ij} \tag{11.18}$$

for any two 3×3 matrices A and B we can write (11.15) as

$$-(\sigma\cdot n, v)_{\partial\Omega} + (\sigma : \nabla v) = (f, v) \tag{11.19}$$

where the entries of the 3×3 gradient matrix ∇v are given by $[\nabla v]_{ij} = \partial v_i/\partial v_j$.

From the boundary condition $\sigma\cdot n = g_N$ on Γ_N, and since $v = 0$ on Γ_D we further have

$$(\sigma\cdot n, v)_{\partial\Omega} = (\sigma\cdot n, v)_{\Gamma_D} + (\sigma\cdot n, v)_{\Gamma_N} = 0 + (g_N, v)_{\Gamma_N} \tag{11.20}$$

Thus, we end up with the variational equation

$$(\sigma : \nabla v) = (f, v) + (g_N, v)_{\Gamma_N}, \quad \forall v \in V \tag{11.21}$$

Next we note that if A is symmetric and B anti-symmetric with zero diagonal then $A : B = 0$. Further, recalling that any matrix A can be decomposed into its symmetric and anti-symmetric part by writing $A = (A + A^T)/2 + (A - A^T)/2$, we have

$$\sigma : \nabla v = \sigma : \tfrac{1}{2}(\nabla v + \nabla v^T) + \sigma : \tfrac{1}{2}(\nabla v - \nabla v^T) = \sigma : \varepsilon(v) + 0 \tag{11.22}$$

since σ is symmetric. This allows us to replace ∇v with $\varepsilon(v)$ in (11.21). We then get

$$(\sigma(u) : \varepsilon(v)) = (f, v) + (g_N, v)_{\Gamma_N}, \quad \forall v \in V \tag{11.23}$$

or, if we insert $\sigma = 2\mu\varepsilon(u) + \lambda(\nabla \cdot u)I$ and use that $I : \varepsilon(v) = \nabla \cdot v$

$$\int_\Omega 2\mu\varepsilon(u) : \varepsilon(v) + \lambda(\nabla \cdot u)(\nabla \cdot v)\,dx = (f, v) + (g_N, v)_{\Gamma_N}, \quad \forall v \in V \tag{11.24}$$

In abstract form the variational formulation of (11.13) thus reads: Find $u \in V$ such that

$$a(u, v) = l(v) \quad \forall v \in V \tag{11.25}$$

where the bilinear from $a(\cdot,\cdot)$ and the linear form $l(\cdot)$ are defined by

$$a(u, v) = \int_\Omega 2\mu\varepsilon(u) : \varepsilon(v) + \lambda(\nabla \cdot u)(\nabla \cdot v)\,dx \tag{11.26}$$

$$l(v) = \int_\Omega f \cdot v\,dx + \int_{\Gamma_N} g_N \cdot v\,ds \tag{11.27}$$

11.2.2 Existence and Uniqueness of Solutions

One of the first question we must ask us is if the variational equation (11.25) is well posed and has a unique solution u? As we know this follows form the Lax-Milgram lemma if we can establish coercivity and continuity of the bilinear form $a(\cdot,\cdot)$ on V, and also continuity of the linear form $l(\cdot)$ on V. To this end we equip V with the standard $H^1(\Omega)$ norm $\|\cdot\|_V = \|\cdot\|_{H^1(\Omega)}$. Furthermore, to measure the size of the various tensors and vectors involved we also introduce the following norms on V

$$\|A\|_V^2 = \sum_{i,j=1}^{3} \|a_{ij}\|_V^2, \qquad \|b\|_V^2 = \sum_{i=1}^{3} \|b_i\|_V^2 \tag{11.28}$$

for A a 3×3 tensor, and b a 3×1 vector.

We begin by showing the continuity of $a(\cdot,\cdot)$. Using the Cauchy-Schwartz inequality we have

$$a(u,v) = \int_\Omega 2\mu\varepsilon(u) : \varepsilon(v) + \lambda(\nabla \cdot u)(\nabla \cdot v)\, dx \tag{11.29}$$
$$\leq 2\mu\|\varepsilon(u)\|\|\varepsilon(v)\| + \lambda\|\nabla \cdot u\|\|\nabla \cdot v\| \tag{11.30}$$
$$\leq C\|\nabla u\|\|\nabla v\| \tag{11.31}$$
$$\leq C\|u\|_V\|v\|_V \tag{11.32}$$

The continuity of the linear form $l(\cdot)$ follows from the trace inequality

$$\|v\|_{L^2(\Gamma)} \leq C(\|\nabla v\| + \|v\|) \leq C\|v\|_V \tag{11.33}$$

where Γ is any part of the boundary $\partial\Omega$. Using this inequality we have

$$l(v) \leq (f,v) + (g,v)_{\Gamma_N} \tag{11.34}$$
$$\leq \|f\|\|v\| + \|g\|_{L^2(\Gamma_N)}\|v\|_{L^2(\Gamma_N)} \tag{11.35}$$
$$\leq \|f\|\|v\|_V + \|g\|_{L^2(\Gamma_N)}\|v\|_V \tag{11.36}$$
$$\leq C\|v\|_V \tag{11.37}$$

which proves continuity of $l(\cdot)$.

To prove coercivity of the bilinear form $a(\cdot,\cdot)$ we need the following result.

Theorem 11.1 (Korn's Inequality). *There exist a positive constant C such that*

$$C\|\nabla v\|^2 \leq \|\varepsilon(v)\|^2 = \int_\Omega \sum_{i,j=1}^{3} \varepsilon_{ij}(v)\varepsilon_{ij}(v)\, dx \tag{11.38}$$

Proof. For simplicity let us assume that $u = 0$ on the whole boundary $\partial\Omega$. Straight forward calculation reveals that

$$\int_\Omega \sum_{i,j=1}^{3} \varepsilon_{ij}(v)\varepsilon_{ij}(v)\, dx = \int_\Omega \sum_{i,j=1}^{3} \frac{1}{2}\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right)\frac{1}{2}\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right) dx \tag{11.39}$$
$$= \frac{1}{4}\int_\Omega \sum_{i,j=1}^{3} \left(\frac{\partial v_i}{\partial x_j}\right)^2 + 2\frac{\partial v_i}{\partial x_j}\frac{\partial v_j}{\partial x_j} + \left(\frac{\partial v_j}{\partial x_i}\right)^2 dx \tag{11.40}$$
$$= \frac{1}{2}\|\nabla v\|^2 + \frac{1}{2}\sum_{i,j=1}^{3}\int_\Omega \frac{\partial v_i}{\partial x_j}\frac{\partial v_j}{\partial x_j}\, dx \tag{11.41}$$

The theorem now follows if we can show that the second term in the last line is positive. Now, using partial integration twice and that $v = 0$ on $\partial\Omega$, we have

$$\sum_{i,j=1}^{3}\int_{\Omega}\frac{\partial v_i}{\partial x_j}\frac{\partial v_j}{\partial x_j}\,dx = -\sum_{i,j=1}^{3}\int_{\Omega} v_i\frac{\partial^2 v_j}{\partial x_j \partial x_j}\,dx + \int_{\partial\Omega} n_j v_i \frac{\partial v_j}{\partial x_i}\,ds \tag{11.42}$$

$$= \sum_{i,j=1}^{3}\int_{\Omega}\frac{\partial v_i}{\partial x_i}\frac{\partial v_j}{\partial x_j}\,dx - \int_{\partial\Omega} n_i v_i \frac{\partial v_j}{\partial x_j}\,ds \tag{11.43}$$

$$= \int_{\Omega}\left(\sum_{i=1}^{3}\frac{\partial v_i}{\partial x_i}\right)\left(\sum_{j=1}^{3}\frac{\partial v_j}{\partial x_j}\right)dx \tag{11.44}$$

$$= \int_{\Omega}(\nabla\cdot v)^2\,dx \geq 0 \tag{11.45}$$

We are done.

The coercivity of $a(\cdot,\cdot)$ now follows from the Poincaré inequality $\|v\|_V \leq C\|\nabla v\|$, since

$$a(u,u) = 2\mu\|\varepsilon(u)\|^2 + \lambda\|\nabla\cdot u\|^2 \geq 2\mu\|\varepsilon(u)\|^2 \geq C\|\nabla u\|_V^2 \geq C\|v\|_V^2 \tag{11.46}$$

In view of these results we thus conclude that the requirements for the Lax-Milgram lemma are satisfied, and hence that there exist a unique solution $u \in V$ to the abstract variational equation (11.25).

11.2.3 Finite Element Approximation

From the Lax-Milgram lemma we know that the variational equation (11.25) has a unique solution $u \in V$, which can be approximated using finite elements. To this end let $\mathscr{K} = \{K\}$ be a partition of Ω into tetrahedrons K.

We shall choose to approximate the displacement field using continuous piecewise linears for each displacement component. The appropriate discrete space for doing so is

$$V_h = \left\{v \in [S_h]^3 : v|_{\Gamma_D} = 0\right\} \tag{11.47}$$

where S_h is the space of continuous piecewise linears on $\mathscr{K}$.

The finite element approximation to (11.25) reads: Find $u_h \in V_h$, such that

$$a(u_h, v) = l(v), \quad \forall v \in V_h \tag{11.48}$$

11.3 A Priori Error Estimate

As always we need to assert the accuracy of the finite element solution u_h by estimating the error $e = u - u_h$.

We have the following a priori result.

Theorem 11.2. *The finite element solution u_h satisfies the following estimate*

$$\|\nabla e\| \leq Ch\|D^2u\| \tag{11.49}$$

where C is constant independent of u, u_h, and the meshsize h.

Proof. Starting from the coercivity result we have

$$C\|\nabla e\|^2 \leq a(e,e) = a(e,u-u_h) = a(e,u-\pi u+\pi u-u_h) = a(e,u-\pi u) \tag{11.50}$$

where we have added and subtracted an interpolant $\pi u \in V_h$ to u from the finite element space, and used that $a(e,\pi u) = 0$ by Galerkin orthogonality. Using also the continuity if $a(\cdot,\cdot)$ we have

$$C\|\nabla e\|^2 \leq a(e,u-\pi u) \leq C\|\nabla e\|\|\nabla(u-\pi u)\| \tag{11.51}$$

Now, from interpolation theory we have the estimate

$$\|\nabla(u-\pi u)\| \leq Ch\|D^2u\| \tag{11.52}$$

from which the theorem immediately follows.

11.4 Engineering Notation

In the engineering business it is customary to rewrite the bilinear form $a(u_h,v)$ as a product of a few matrices, since this allows a simple bookkeeping of the index i,j, and of the components $(u_h)_j$ and v_i. The starting point is to rearrange the independent components of the stress tensor into a vector, viz.

$$\sigma = \begin{bmatrix}\sigma_{11} & \sigma_{22} & \sigma_{33} & \sigma_{12} & \sigma_{23} & \sigma_{31}\end{bmatrix}^T \tag{11.53}$$

The strain tensor is written as a vector as well

$$\varepsilon = \begin{bmatrix}\varepsilon_{11} & \varepsilon_{22} & \varepsilon_{33} & 2\varepsilon_{12} & 2\varepsilon_{23} & 2\varepsilon_{31}\end{bmatrix}^T \tag{11.54}$$

Hooke's law (11.9) can now be expressed as

$$\sigma = D\varepsilon \tag{11.55}$$

with

$$D = \begin{bmatrix} \lambda+2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda+2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda+2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix} \tag{11.56}$$

for three-dimensional elasticity.

For two-dimensional applications one has to differ between plane strain

$$\varepsilon_{13} = \varepsilon_{23} = \varepsilon_{31} = 0, \qquad \sigma_{33} = \nu(\sigma_{11} + \sigma_{22}) \tag{11.57}$$

and plane stress

$$\sigma_{13} = \sigma_{23} = \sigma_{31} = 0, \qquad \varepsilon_{33} = -\frac{\nu}{E}(\sigma_{33} + \sigma_{22}) \tag{11.58}$$

Both cases can be handled by a constitutive law of the form $\sigma = D\varepsilon$, but now with

$$\sigma = \begin{bmatrix}\sigma_{11} \ \sigma_{22} \ \sigma_{12}\end{bmatrix}^T, \qquad \varepsilon = \begin{bmatrix}\varepsilon_{11} \ \varepsilon_{22} \ 2\varepsilon_{12}\end{bmatrix}^T \tag{11.59}$$

and

$$D = \begin{bmatrix} \lambda+2\mu & \lambda & 0 \\ \lambda & \lambda+2\mu & 0 \\ 0 & 0 & \mu \end{bmatrix} \tag{11.60}$$

for plane strain and

$$D = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)/2 \end{bmatrix} \tag{11.61}$$

for plane stress. We shall return to this later on.

The engineering notation adopted above to define the stress and strain tensors allows us to write

$$\varepsilon : \sigma = \varepsilon^T \sigma = \varepsilon^T D \varepsilon \tag{11.62}$$

which implies that

$$a(u,v) = \int_\Omega \varepsilon(v) : \sigma(u)\, dx = \int_\Omega \varepsilon^T(v)\sigma(u)\, dx = \int_\Omega \varepsilon^T(v) D \varepsilon(u)\, dx \tag{11.63}$$

It is convenient to write the finite element ansatz $u_h \in V_h$ in matrix form as

$$u_h = \begin{bmatrix} (u_h)_1 \\ (u_h)_2 \\ (u_h)_3 \end{bmatrix} = \begin{bmatrix} \varphi_1 & 0 & 0 & \varphi_2 & 0 & 0 & \dots & \varphi_{n_i} & 0 & 0 \\ 0 & \varphi_1 & 0 & 0 & \varphi_2 & 0 & \dots & 0 & \varphi_{n_i} & 0 \\ 0 & 0 & \varphi_1 & 0 & 0 & \varphi_2 & \dots & 0 & 0 & \varphi_{n_i} \end{bmatrix} \begin{bmatrix} d_{11} \\ d_{12} \\ d_{13} \\ d_{21} \\ d_{22} \\ d_{23} \\ \vdots \\ d_{N1} \\ d_{N2} \\ d_{N3} \end{bmatrix} = \varphi d \tag{11.64}$$

where φ_i, $i = 1, 2, \dots, n_i$ are the hat basis functions in three dimensions, and d is a vector containing the nodal displacements. Note that there are three displacements $d_{ij} = (u_h)_j(N_i)$ per node N_i, and thus that d is of length three times the number of (internal) nodes, n_i.

The strain field is linked to the displacements by (11.8). An alternative way of writing this is

$$\begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ 2\varepsilon_{12} \\ 2\varepsilon_{23} \\ 2\varepsilon_{31} \end{bmatrix} = \begin{bmatrix} \partial/\partial x_1 & 0 & 0 \\ 0 & \partial/\partial x_2 & 0 \\ 0 & 0 & \partial/\partial x_3 \\ \partial/\partial x_2 & \partial/\partial x_1 & 0 \\ 0 & \partial/\partial x_3 & \partial/\partial x_2 \\ \partial/\partial x_3 & 0 & \partial/\partial x_1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{11.65}$$

Introducing the strain matrix

$$B = \begin{bmatrix} \partial/\partial x_1 & 0 & 0 \\ 0 & \partial/\partial x_2 & 0 \\ 0 & 0 & \partial/\partial x_3 \\ \partial/\partial x_2 & \partial/\partial x_1 & 0 \\ 0 & \partial/\partial x_3 & \partial/\partial x_2 \\ \partial/\partial x_3 & 0 & \partial/\partial x_1 \end{bmatrix} \begin{bmatrix} \varphi_1 & 0 & 0 & \varphi_2 & 0 & 0 & \dots & \varphi_{n_i} & 0 & 0 \\ 0 & \varphi_1 & 0 & 0 & \varphi_2 & 0 & \dots & 0 & \varphi_{n_i} & 0 \\ 0 & 0 & \varphi_1 & 0 & 0 & \varphi_2 & \dots & 0 & 0 & \varphi_{n_i} \end{bmatrix} \tag{11.66}$$

we have the discrete strains and stresses

$$\varepsilon = Bd \tag{11.67}$$

$$\sigma = DBd \tag{11.68}$$

With these definitions the matrix formulation of the finite element method (11.48) becomes

$$\left(\int_\Omega B^T DB \, dx \right) d = \int_\Omega \varphi^T f \, dx + \int_{\Gamma_N} \varphi^T g \, ds \tag{11.69}$$

or simply

$$Kd = F \tag{11.70}$$

where K is the $3n_i \times 3n_i$ stiffness matrix

$$K = \int_\Omega B^T DB\,dx \tag{11.71}$$

and F is the $3n_i \times 1$ load vector

$$F = \int_\Omega \varphi^T f\,dx + \int_{\Gamma_N} \varphi^T g\,ds \tag{11.72}$$

11.4.1 Computer Implementation

Although deformation is a genuine three-dimensional phenomenon it is sometimes possible to reduce the analysis to two dimensions. For example, say that we have a very slender structure oriented along the x_3-axis with length much greater than cross-section area. Then the strains associated with length (i.e, ε_{13}, ε_{23}, and ε_{33}) are small compared to the cross-sectional strains, since they are constrained by nearby material. In this case it suffice to consider a reduced two-dimensional elastic problem within the cross-section to deduce the deformation of the structure. The conditions that $u_3 = 0$ and that there is no variation with respect to x_3 (i.e., $\partial/\partial x_3 = 0$) are precisely the assumptions of plain strain. The state of plane stress applies to structures which are large but thin, such as plates or shells, for instance.

Let us work through the details of writing a two-dimensional finite element solver. To this end let $\Omega \subset \mathbb{R}^2$ from now on denote a two-dimensional domain within the x_1x_2-plane, and let $\mathscr{K} = \{K\}$ be a triangle mesh of Ω.

As usual the stiffness matrix (11.71) and the load vector (11.72) can be assembled by summing integral contributions from each element. Consider therefore an element K with the three nodes N_i, $i = 1,2,3$. On K the element displacements u_h^K are given by

$$u_h^K = \begin{bmatrix} \varphi_1 & 0 & \varphi_2 & 0 & \varphi_3 & 0 \\ 0 & \varphi_1 & 0 & \varphi_2 & 0 & \varphi_3 \end{bmatrix} \begin{bmatrix} d_{11} \\ d_{12} \\ d_{21} \\ d_{22} \\ d_{31} \\ d_{32} \end{bmatrix} = \varphi^K d^K \tag{11.73}$$

where φ_i are hat functions. Recall that these are given by

$$\varphi_i = \frac{1}{2|K|}(a_i + b_i x_1 + c_i x_2) \tag{11.74}$$

where $|K|$ is the area of K, and where the coefficients a_i, b_i, and c_i are determined from the requirement $\varphi_i(N_j) = \delta_{ij}$.

The element strains are given by

$$\varepsilon^K = \begin{bmatrix} \partial/\partial x_1 & 0 \\ 0 & \partial/\partial x_2 \\ \partial/\partial x_2 & \partial/\partial x_1 \end{bmatrix} u_h^K = \frac{1}{2|K|} \begin{bmatrix} b_1 & 0 & b_2 & 0 & b_3 & 0 \\ 0 & c_1 & 0 & c_2 & 0 & c_3 \\ c_1 & b_1 & c_2 & b_2 & c_3 & b_3 \end{bmatrix} d^K = B^K d^K \tag{11.75}$$

We note that the strain matrix B^K is constant and hence that that the strains are constant on the element. Because the element strains are constant, so are the element stresses $\sigma^K = D\varepsilon^K$.

The element stiffness matrix is now given by

$$K^K = \int_K B^{K^T} D B^K \, dx \tag{11.76}$$

which simplifies to $K^K = B^{K^T} D B^K |K|$, since the integrand is constant. Here, let us assume a state of plane strain in which case the matrix D is given by (11.57).

Writing a code for computing K^K is easy.

```
function KK = ElasticStiffness(x,y,mu,lambda)
% triangle area and gradients (b,c) of hat functions
[area,b,c]=Gradients(x,y);
% elastic matrix
D=mu*[2 0 0; 0 2 0; 0 0 1]+lambda*[1 1 0; 1 1 0; 0 0 0];
% strain matrix
BK=[b(1)   0  b(2)   0  b(3)   0 ;
      0  c(1)   0  c(2)   0  c(3);
    c(1) b(1) c(2) b(2) c(3) b(3)];
% element stiffness matrix
KK=BK'*D*BK*area;
```

Input to this routine is the node coordinates `x` and `y`, and the Lamé parameters `lambda` and `mu`. Output is the 6×6 element stiffness matrix `KK`.

The element load vector is given by

$$F^K = \int_K \varphi^{K^T} f \, dx = \int_K \begin{bmatrix} \varphi_1 & 0 \\ 0 & \varphi_1 \\ \varphi_2 & 0 \\ 0 & \varphi_2 \\ \varphi_3 & 0 \\ 0 & \varphi_3 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} dx \tag{11.77}$$

To evaluate these integrals without pain we can use the old trick of replacing f with its linear interpolant πf, and then integrate the interpolant. Recall that πf is defined on K by

$$\pi f = \begin{bmatrix} \pi f_1 \\ \pi f_2 \end{bmatrix} = \begin{bmatrix} \varphi_1 & 0 & \varphi_2 & 0 & \varphi_3 & 0 \\ 0 & \varphi_1 & 0 & \varphi_2 & 0 & \varphi_3 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{12} \\ f_{22} \\ f_{13} \\ f_{23} \end{bmatrix} = \varphi^{K^T} f^K \tag{11.78}$$

where $f_{ij} = f_i(N_j)$ are the nodal force values. This now gives us

$$F^K = \int_K \varphi^{K^T} f \, dx \approx \int_K \begin{bmatrix} \varphi_1 & 0 \\ 0 & \varphi_1 \\ \varphi_2 & 0 \\ 0 & \varphi_2 \\ \varphi_3 & 0 \\ 0 & \varphi_3 \end{bmatrix} \begin{bmatrix} \varphi_1 & 0 & \varphi_2 & 0 & \varphi_3 & 0 \\ 0 & \varphi_1 & 0 & \varphi_2 & 0 & \varphi_3 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{12} \\ f_{22} \\ f_{13} \\ f_{23} \end{bmatrix} dx \tag{11.79}$$

$$= \int_K \begin{bmatrix} \varphi_1^2 & 0 & \varphi_2\varphi_1 & 0 & \varphi_3\varphi_1 & 0 \\ 0 & \varphi_1^2 & 0 & \varphi_2\varphi_1 & 0 & \varphi_3\varphi_1 \\ \varphi_1\varphi_2 & 0 & \varphi_2^2 & 0 & \varphi_3\varphi_2 & 0 \\ 0 & \varphi_1\varphi_2 & 0 & \varphi_2^2 & 0 & \varphi_3\varphi_2 \\ \varphi_1\varphi_3 & 0 & \varphi_2\varphi_3 & 0 & \varphi_3^2 & 0 \\ 0 & \varphi_1\varphi_3 & 0 & \varphi_2\varphi_3 & 0 & \varphi_3^2 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{12} \\ f_{22} \\ f_{13} \\ f_{23} \end{bmatrix} dx = M^K f^K \tag{11.80}$$

where M^K is the element mass matrix. Evaluating its integrals one finds that

$$M^K = \frac{1}{12} \begin{bmatrix} 2 & 0 & 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 & 0 & 1 \\ 1 & 0 & 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 & 0 & 1 \\ 1 & 0 & 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 1 & 0 & 2 \end{bmatrix} |K| \tag{11.81}$$

which immediately translates into MATLAB code.

```
function MK = ElasticMass(x,y)
area=polyarea(x,y);
MK=[2 0 1 0 1 0;
    0 2 0 1 0 1;
    1 0 2 0 1 0;
    0 1 0 2 0 1;
    1 0 1 0 2 0;
    0 1 0 1 0 2]*area/12;
```

Since the element load is approximately given by $F^K = M^K f^K$ on each element, it is straight forward to assemble the load vector F as the sum $F = \sum_K F^K$.

When performing the assembly of the global system of equations, one needs to recall that there are two unknowns, or, degrees of freedom, per node. This makes the insertion of element matrix contributions into the global system matrix a bit more trickier than usual. In order to add the local stiffness K_{ij}^K to its correct location in the global stiffness matrix K, we have to make a map between the node numbers and the numbering of the displacement degrees of freedom. To be honest we have already set up this mapping when ordering the nodal displacements in the vector d. Recall that all odd vector entries d_{2i-1} have to do with the x_1-displacement $(u_h)_1$, and that all even entries d_{2i} has to do with the x_2-displacement $(u_h)_2$. This is also true for the element displacement vector d^K. Thus the two displacement components in node number i is mapped onto vector entries d_{2i-1} and d_{2i}, $i = 1, 2, \dots, N$, and the map between a node N_i and its degrees of freedom is consequently $i \mapsto (2i-1, 2i)$. For example, if element K has the nodes 3, 5 and 6, then the degrees of freedom is 5, 6, 9, 10, 11, and 12. From this it follows that the local stiffness K_{15}^K should be added to row 5 column 11 in the global stiffness matrix K.

Using the subroutines `ElastStiffness` and `ElastMass` we can now write a routine for assembling the global stiffness matrix K and the global load vector F. For later use we also assemble the global mass matrix M.

```
function [K,M,F] = ElasticAssema(p,e,t,lambda,mu,force)
ndof=2*size(p,2); % total number of degrees of freedom
K=sparse(ndof,ndof); % allocate stiffness matrix
M=sparse(ndof,ndof); % allocate mass matrix
F=zeros(ndof,1); % allocate load vector
dofs=zeros(6,1); % allocate element degrees of freedom
for i=1:size(t,2) % assemly loop over elements
  nodes=t(1:3,i); % element nodes
  x=p(1,nodes); y=p(2,nodes); % node coordinates
  dofs(2:2:end)=2*nodes; % element degrees of freedom
  dofs(1:2:end)=2*nodes-1;
  f=force(x,y); % evaluate force at nodes
  KK=ElasticStiffness(x,y,lambda,mu); % element stiffness
  MK=ElasticMass(x,y); % element mass
  fK=[f(1,1) f(2,1) f(1,2) f(2,2) f(1,3) f(2,3)]';
  FK=MK*fK; % element load
  K(dofs,dofs)=K(dofs,dofs)+KK; % add to stiffness matrix
  M(dofs,dofs)=M(dofs,dofs)+MK; % add to mass matrix
  F(dofs)=F(dofs)+FK; % add to load vector
end
```

Input is the usual point, edge, and triangle matrices `p`, `e`, and `t`, the Lamé parameters `lambda`, and `mu`, and a function handle `force` to a subroutine specifying the body force f. For example,

```
function f = Force(x,y)
f=[35/13*y-35/13*y.^2+10/13*x-10/13*x.^2;
  -25/26*(-1+2*y).*(-1+2*x)];
```

Output is the global stiffness matrix `K`, the global mass matrix `M`, and the global load vector `F`.

The Lamé parameters λ and μ can conveniently be computed from the Young modulus E and the Poisson's ratio ν, which are the usual physical data available, with the following subroutine.

```
function [mu,lambda] = Enu2Lame(E,nu)
mu=E/(2*(1+nu));
lambda=E*nu/((1+nu)*(1-2*nu));
```

For the stiffness matrix to be invertible some boundary conditions must be enforced. Assuming a Dirichlet type boundary condition this can be done as usual with a list `fixed` containing the fixed degrees of freedom on the Dirichlet boundary Γ_D, and another list `values` with the corresponding nodal displacement values. For example, if we have the homogeneous Dirichlet condition $u = g_D = 0$ on the whole boundary, then the construction of `fixed` and `values` can be done with the code

```
bdry=unique([e(1,:) e(2,:)]); % boundary nodes
fixed=[2*bdry-1 2*bdry]; % boundary degrees of freedom
values=zeros(length(fixed),1); % zero boundary values
```

The elimination of the boundary degrees of freedom and subsequent solution of the linear system $Kd = F$ is then done with the lines

```
ndof=length(F);
free=setdiff([1:ndof],fixed);
F=F(free)-K(free,fixed)*values;
K=K(free,free);
d=zeros(ndof,1);
d(free)=K\F;
d(fixed)=values;
```

The main routine for our linear elastic finite element solver is given below.

```
function ElasticitySolver()
g=Rectg(0,0,1,1);
[p,e,t]=initmesh(g,'hmax',0.1);
E=1; nu=0.3;
[mu,lambda]=Enu2Lame(E,nu);
[K,M,F]=ElasticAssema(p,e,t,mu,lambda,@Force);
bdry=unique([e(1,:) e(2,:)]);
fixed=[2*bdry-1 2*bdry];
values=zeros(length(fixed),1);
ndof=length(F);
free=setdiff([1:ndof],fixed);
F=F(free)-K(free,fixed)*values;
K=K(free,free);
d=zeros(ndof,1);
d(free)=K\F;
d(fixed)=values;
U=d(1:2:end); V=d(2:2:end);
```

```
figure(1), pdesurf(p,t,U), title('(u_h)_1')
figure(2), pdesurf(p,t,V), title('(u_h)_2')
```

11.4.2 Verifying the Energy Norm Convergence

Let us verify that the finite element solver outlined above is implemented correctly. By taking the logarithm of the estimate $\sqrt{a(e,e)} \leq Ch$, which can be deduced from the proof of the a priori estimate (11.49), we find that the error $e = u - u_h$ satisfies

$$\log \sqrt{a(e,e)} \leq \log(Ch) = C + \log h \tag{11.82}$$

where C is a constant depending on D^2u. The quantity $\sqrt{a(\cdot,\cdot)}$ is called the energy norm and is sometimes denoted $\|\cdot\|_E$. From (11.82) it follows that if we make a plot of $\log h$ versus $\log \|e\|_E$ we should asymptotically get a straight line with slope 1. However, to be able to compute e we need to know the exact solution u, and we shall therefore manufacture a problem with known solution. Let $\Omega = [0,1] \times [0,1]$ be the unit square and let $u = [x_1(1-x_1)x_2(1-x_2), 0]$. This choice of u assures that $u = 0$ on the boundary $\partial\Omega$. Using u to first compute the strain tensor ε, and then the stress tensor σ, and finally $-\nabla \cdot \sigma$, we find that f equals

$$f = \begin{bmatrix} 35/13x_2 - 35/13x_2^2 + 10/13x_1 - 10/13x_1^2 \\ -25/26(-1+2x_2)(-1+2x_1) \end{bmatrix} \tag{11.83}$$

with $E = 1$ and $\nu = 0.3$. In the same way we also find that

$$a(u,u) = \int_\Omega \sigma(u) : \varepsilon(u)\,dx = 1/52 \tag{11.84}$$

To compute $a(e,e)$ we note that $a(e,e) = a(u,u) - a(u_h,u_h)$ by Galerkin orthogonality, and that $a(u_h,u_h)$ can be easily computed as $a(u_h,u_h) = d^T K d = F^T d$. Recording the meshsize h and the energy norm error $\|e\|_E$ for 10 different uniform meshes we get the results shown in Table 11.1. In Figure 11.1 we show a loglog plot of the data points. Looking at the plot we see that it is almost a straight line and by doing a linear least squares fit on the data we find that the slope of the line is 1.0104, which indeed is close to the predicted value of 1.

h	$\sqrt{F^T d}$	$\|e\|_E$
0.1250	0.1372	0.0201
0.1125	0.1374	0.0187
0.1000	0.1377	0.0162
0.0875	0.1379	0.0146
0.0750	0.1381	0.0125
0.0625	0.1383	0.0103
0.0500	0.1384	0.0083
0.0375	0.1385	0.0061
0.0250	0.1386	0.0040
0.0125	0.1387	0.0020

Table 11.1 Convergence of error in the energy norm.

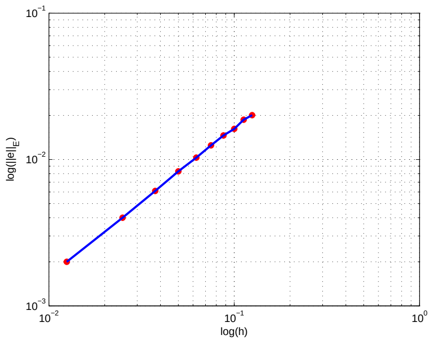


Fig. 11.1 Loglog plot of error in energy norm versus mesh size.

11.5 A Posteriori Estimate

In order to formulate adaptive finite elements we want to derive a posteriori estimates for the error $e = u - u_h$. In doing so let us for simplicity assume zero displacement boundary conditions along the whole boundary. Starting from Korn's inequality and the Galerkin orthogonality, $a(e, v) = 0$ for all $v \in V_h$, with v chosen as the interpolant $\pi e \in V_h$, we have

$$\begin{aligned}
C\|\nabla e\|^2 &\le a(e,e) && (11.85)\\
&= a(e,e-\pi e) && (11.86)\\
&= a(u,e-\pi e) - a(u_h,e-\pi e) && (11.87)\\
&= (f,e-\pi e) - a(u_h,e-\pi e) && (11.88)\\
&= \sum_{K\in\mathcal{K}} (f,e-\pi e)_K - (\sigma : \varepsilon(e-\pi e))_K && (11.89)\\
&= \sum_{K\in\mathcal{K}} (f,e-\pi e)_K + (\nabla\cdot\sigma, e-\pi e) - (\sigma\cdot n, e-\pi e)_{\partial K\setminus\partial\Omega} && (11.90)\\
&= \sum_{K\in\mathcal{K}} (f+\nabla\cdot\sigma, e-\pi e)_K + (\tfrac{1}{2}[\sigma\cdot n], e-\pi e)_{\partial K\setminus\partial\Omega} && (11.91)
\end{aligned}$$

where as usual $[\sigma\cdot n]$ denotes the jump in the normal stress over the element boundaries. Recall that if K^+ and K^- are two elements sharing edge E with unit normal n_E pointing from K^+ to K^-, then by definition $[\sigma\cdot n] = (\sigma|_{K^-} - \sigma|_{K^-})\cdot n_E$ on E.

Using the Cauchy-Schwartz inequality on each term of (11.91) we further have

$$\begin{aligned}
\|\nabla e\|^2 \le C \sum_{K\in\mathcal{K}} &\|f+\nabla\cdot\sigma\|_K \|e-\pi e\|_K && (11.92)\\
&+ h_K^{-1/2}\|\tfrac{1}{2}[\sigma\cdot n]\|_{\partial K\setminus\partial\Omega} h_K^{1/2}\|e-\pi e\|_{\partial K\setminus\partial\Omega}
\end{aligned}$$

since both u and u_h, and thus also e and πe, are zero on the boundary.

Recalling next the trace inequality $\|v\|_{\partial K} \le C(h_K^{-1/2}\|v\|_K + h_K^{1/2}\|\nabla v\|_K)$, the interpolation estimate $\|v-\pi v\|_K \le Ch_K\|\nabla v\|_K$, and the stability estimate $\|\nabla(\pi v)\| \le C\|\nabla v\|$, we obtain

$$h_K^{1/2}\|e-\pi e\|_{\partial K} \le C(\|e-\pi e\|_K + h_K\|\nabla(e-\pi e)\|_K) \le h_K\|\nabla e\|_K \qquad (11.93)$$

Using this result and the Cauchy-Schwartz inequality again we have

$$\begin{aligned}
\|\nabla e\|^2 &\le C \sum_{K\in\mathcal{K}} (h_K\|f+\nabla\cdot\sigma\|_K + h_K^{1/2}\|\tfrac{1}{2}[\sigma\cdot n]\|_{\partial K\setminus\partial\Omega})\|\nabla e\|_K && (11.94)\\
&\le C\left(\sum_{K\in\mathcal{K}} h_K^2\|f+\nabla\cdot\sigma\|_K^2 + h_K\|\tfrac{1}{2}[\sigma\cdot n]\|_{\partial K\setminus\partial\Omega}^2\right)^{1/2} \|\nabla e\| && (11.95)
\end{aligned}$$

Finally, dividing by $\|\nabla e\|$ we end up with the a posteriori estimate

$$\begin{aligned}
\|\nabla e\| &\le C\left(\sum_{K\in\mathcal{K}} h_K^2\|f+\nabla\cdot\sigma\|_K^2 + h_K\|\tfrac{1}{2}[\sigma\cdot n]\|_{\partial K\setminus\partial\Omega}^2\right)^{1/2} && (11.96)\\
&\le C\sum_{K\in\mathcal{K}} h_K\|f+\nabla\cdot\sigma\|_K + h_K^{1/2}\|\tfrac{1}{2}[\sigma\cdot n]\|_{\partial K\setminus\partial\Omega} && (11.97)
\end{aligned}$$

Thus, we have proved the following theorem.

Theorem 11.3. *The finite element solution u_h satisfies the a posteriori error estimate*

$$\|\nabla e\| \leq C \sum_{K\in\mathscr{K}} \eta_K \tag{11.98}$$

where the element indicator η_K is the sum of the cell residual $R_K = h_K\|f+\nabla\cdot\sigma\|_K$ and the edge residual $r_K = h_K^{1/2}\|\frac{1}{2}[\sigma\cdot n]\|_{\partial K\setminus\partial\Omega}$.

Next we show how to compute the cell and edge residuals.

The cell residual is easy to compute with one point quadrature. Note that it simplifies to $R_K = \|f\|_K$ for a piecewise linear u_h.

```
function RK = CellResiduals(p,t,force)
nt=size(t,2); % number of elements
RK=zeros(nt,1); % allocate element residuals
for i=1:nt % loop over elements
  nodes=t(1:3,i); % nodes
  x=p(1,nodes); y=p(2,nodes); % node coordinates
  [area,ds]=Triutils(x,y); % area and side lengths
  f=force(mean(x),mean(y)); % force at element centroid
  h=max(ds); % local mesh size is max side length
  RK(i)=h*sqrt(dot(f,f)*area); % cell residual h|f|_K
end
```

Here, we use the following utility routine to compute the area, edge lengths, and outward unit normals on an element. Edge 1 is opposite node 1, edge 2 opposite node 2, etc.

```
function [area,ds,nx,ny] = Triutils(x,y)
area=polyarea(x,y); % triangle area
dx=[x(3)-x(2); x(1)-x(3); x(2)-x(1)];
dy=[y(2)-y(3); y(3)-y(1); y(1)-y(2)];
ds=sqrt(dx.*dx+dy.*dy); % side lengths
nx=-dy./ds; % outward unit normal components
ny=-dx./ds;
```

The edge residual is a little more complicated to compute than the cell residual, since it requires information about the element neighbors. A routine called `Tri2Tri` for computing element neighbors is given in the Appendix.

```
function rK = EdgeResiduals(p,t,E,nu,U,V)
nt=size(t,2);
rK=zeros(nt,1); % allocate edge residuals
nbrs=Tri2Tri(p,t); % get element neighbours
[mu,lambda]=Enu2Lame(E,nu);
[ux,uy]=pdegrad(p,t,U); % gradient of U
[vx,vy]=pdegrad(p,t,V);
for i=1:nt
```

```
    nodes=t(1:3,i);
    x=p(1,nodes); y=p(2,nodes);
    r=0; % sum of edge residuals sqrt(h)|0.5[n.sigma]|_dK
    [area,ds,nx,ny]=Triutils(x,y);
    h=max(ds);
    for j=1:3 % loop over element edges
      n=nbrs(i,j); % element neighbour
      if n<0 % no neighbour
        continue; % don't compute on boundary
      end
      Sp=Stress(mu,lambda,ux,uy,vx,vy,i); % stress on element i
      Sm=Stress(mu,lambda,ux,uy,vx,vy,n); % stress on neighbour
      jump=0.5*(Sm-Sp)*[nx(j); ny(j)]; % stress jump
      r=r+dot(jump,jump)*ds(j);
    end
    rK(i)=sqrt(h)*sqrt(r);
  end
```

To compute the stress tensor on a given element we use the following subroutine.

```
  function sigma = Stress(mu,lambda,ux,uy,vx,vy,i);
  div=ux(i)+vy(i); % div U
  grad=[ux(i) uy(i); vx(i) vy(i)]; % grad U
  epsilon=(grad+grad')/2; % strain
  sigma=2*mu*epsilon+lambda*div*eye(2); % stress
```

11.5.1 Adaptive Mesh Refinement on a Rotated L-shaped Domain

We illustrate the use of the element indicator η_K by adaptively solving a problem with a manufactured solution. The domain Ω is a rotated L-shaped polygon with vertex points $(-1,-1)$, $(0,0)$, $(-1,1)$, $(0,2)$, $(2,0)$, and $(0,-2)$, see Figure 11.2. The solution u is known in polar coordinates (r,θ).

$$u_r(r,\theta) = \frac{1}{2\mu} r^{\alpha}((c_2-\alpha-1)\cos((\alpha-1)\theta)-(\alpha+1)\cos((\alpha+1)\theta)) \tag{11.99}$$

$$u_\theta(r,\theta) = \frac{1}{2\mu} r^{\alpha}((\alpha+1)\sin((\alpha+1)\theta)+(c_2+\alpha-1)\sin((\alpha-1)\theta)) \tag{11.100}$$

where the exponent α is the solution to the equation $\alpha\sin(2\omega)+\sin(2\omega\alpha)=0$ with $\omega=3\pi/4$, $c_1=-\cos((\alpha+1)\omega)/\cos((\alpha-1)\omega)$, and $c_2=2(\lambda+2\mu)/(\lambda+\mu)$. This displacement field satisfies (11.13) with $f=0$ and $\Gamma_D=\partial\Omega$. In the computations we use $E=1$ and $\nu=0.3$.

The most prominent feature of the solution u is that its gradient tends towards infinity at origo, which coincides with the reentrant corner of the L-shaped domain.

In order to capture this rapid growth of the gradient it is necessary to have a high density of nodes near origo. Moreover, from the a posteriori estimate we have the upper bound $\|\nabla e\| \leq C \sum_K \eta_K$, which implies that we get accurate values of the gradient when using the element indicators η_K to select elements for refinement. Indeed, starting with the coarse mesh with ten elements and making ten adaptive refinement loops we obtain the mesh shown in Figure 11.2. Clearly, the adaptive algorithm has identified and resolved the region around the reentrant corner. The computed displacement is shown in Figure 11.3.

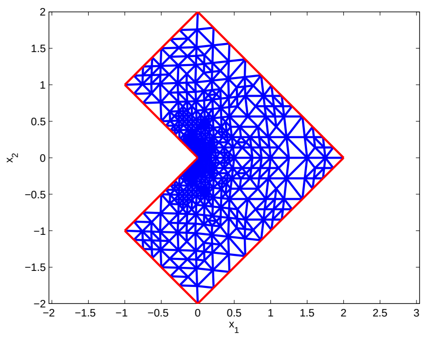


Fig. 11.2 Final adapted mesh.

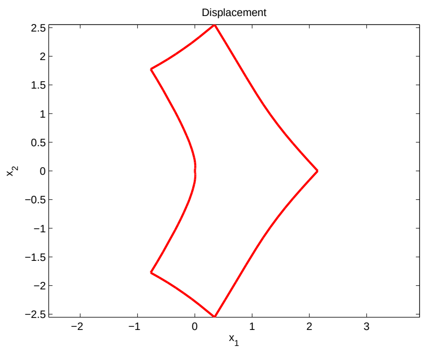


Fig. 11.3 Computed displacement of the rotated L-shaped domain.

11.6 The Equations of Linear Thermoelasticity

Heating or cooling of a material leads to isotropic expansion or contraction. The strains associated with this are called thermal strains and is to first order given by

$$\tau = \alpha(T - T_0)I \tag{11.101}$$

where α is the thermal expansion coefficient and T_0 a reference temperature. It is common to write the total strain ε as the sum of the thermal strains and the mechanical strains, where the stresses from the latter obeys Hooke's law. These assumptions give rise to a generalized Hooke's law relating stresses, temperature, and displacement

$$\sigma = 2\mu\varepsilon(u) + \lambda(\nabla \cdot u)I - \alpha(3\lambda + 2\mu)(T - T_0)I \tag{11.102}$$

Given the temperature T this modified stress strain relationship can is combined with the equations of motion $-\nabla \cdot \sigma = f$ to yield the equations of linear thermoelasticity for u. In variational form, assuming for simplicity $f = 0$, these equations takes the form: find $u \in V$ such that

$$\int_\Omega 2\mu\varepsilon(u) : \varepsilon(v) + \lambda(\nabla \cdot u)(\nabla \cdot v)\,dx = \int_\Omega \alpha(3\lambda + 2\mu)(T - T_0)(\nabla \cdot v)\,dx, \quad \forall v \in V \tag{11.103}$$

From this we see that the thermal strains yield a load proportional to the temperature raise $T - T_0$.

Usually the temperature T is not available in closed form, but has to be computed (e.g., by solving a heat transfer problem with finite elements).

11.7 The Equations of Linear Elastodynamics

So far we have only dealt with statics, but since it is easy to extend the analysis to dynamics let us do so. To this end recall that Newton's second law $F = ma$ says that the net force F acting on a particle equals the mass m of the particle times its acceleration a. Translated to the continuum setting these equations of motion takes the form

$$\rho\ddot{u} = f + \nabla \cdot \sigma \tag{11.104}$$

where ρ is the density of the material and is the second derivative of the displacement u with respect to time t. To see the analogy between Newton's second law and (11.104) we note that if we consider a small particle with volume dV inside a material body, then ρdV is precisely the mass of the particle, $\ddot{u}$ is its acceleration, and $(f + \nabla \cdot \sigma)dV$ is the net force acting on it. We recognize $f dV$ as externally applied forces, such as gravity for instance, and $\nabla \cdot \sigma dV = \sigma \cdot n dS$ as internal stresses acting on the surface dS of dV with n the outward unit normal on dS.

We can now write down the basic problem of linear elastodynamics: Find the time dependent symmetric stress tensor σ and the time dependent displacement vector u such that

$$\rho \ddot{u} - \nabla \cdot \sigma = f, \quad \text{in } \Omega \times I \tag{11.105a}$$

$$\sigma = 2\mu\varepsilon(u) + \lambda(\nabla \cdot u)I, \quad \text{in } \Omega \times I \tag{11.105b}$$

$$u = 0, \quad \text{on } \Gamma_D \times I \tag{11.105c}$$

$$\sigma \cdot n = 0, \quad \text{on } \Gamma_N \times I \tag{11.105d}$$

$$u = u_0, \quad \text{in } \Omega, \text{ for } t = 0 \tag{11.105e}$$

$$\dot{u} = v_0, \quad \text{in } \Omega, \text{ for } t = 0 \tag{11.105f}$$

where $I = (0, T]$ is the time interval, and u_0 and v_0 is a given initial displacement and velocity, respectively.

11.7.1 Modal Analysis

Noting that the equations of motion (11.104) resembles a wave equation it is natural to look for solutions in the form of plane waves, that is,

$$u = z \sin \sqrt{\lambda} t \tag{11.106}$$

where z is a function independent of time and λ a number. Needless to say both z and λ are unknown. Inserting this ansatz into $\rho \ddot{u} - \nabla \cdot \sigma(u) = f$, assuming $\rho = 1$ and $f = 0$, we obtain

$$-\nabla \cdot \sigma(z) = \lambda z \tag{11.107}$$

which we recognize as a continuous eigenvalue problem for the pair (z, λ).

The variational formulation of the eigenvalue problem reads: Find $(z, \lambda) \in V \times \mathbb{R}$ such that

$$a(z, v) = \lambda(z, v), \quad \forall v \in V \tag{11.108}$$

and the corresponding finite element approximation takes the form: Find $(Z, \Lambda) \in V_h \times \mathbb{R}$ such that

$$a(Z, v) = \Lambda(Z, v), \quad \forall v \in V_h \tag{11.109}$$

In matrix form we have

$$Kd = \Lambda M d \tag{11.110}$$

where K, is the stiffness matrix, M the mass matrix, and d a vector containing the nodal values of Z.

The computation of eigenmodes and eigenvalues is important in engineering and is routinely done to avoid vibrations that can cause mechanical structures to wear out unreasonably fast or fail. In doing so, the aim is to avoid getting resonance phenomenons if the structure is subjected to a harmonically varying external force.

11.7.1.1 Eigenvalues and Eigenmodes of a Steel Bracket

As a computational example we compute the ten lowest eigenvalues and eigenmodes of a freely vibrating steel bracket, see Figure 11.3. The relevant code for assembling the mass and stiffness matrix and calling the eigenvalue solver `eigs` is given below.

```
function ElastModalSolver()
E=1; nu=0.3;
[mu,lambda]=Enu2Lame(E,nu);
[K,M]=ElastAssema(p,e,t,mu,lambda,@force);
[D,lambda]=eigs(K,M,10,'SM');
```

The computed eigenvalues are listed in Table 11.2, and in Figure 11.5 we show the corresponding eigenmodes 1, 4, 5 and 8. Note that the three lowest eigenvalues are zero. This is explained by the fact that we have three rigid body modes, namely, two translations and one rotation, for problems with stress free boundary conditions in two dimensions. These eigenmodes are not proper displacements and causes no stress or strain on the structure. Hence, they belong to the kernel of the bilinear form $a(\cdot,\cdot)$, or equivalently, the null space of the stiffness matrix K.

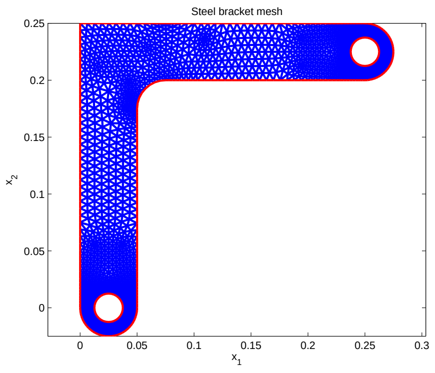
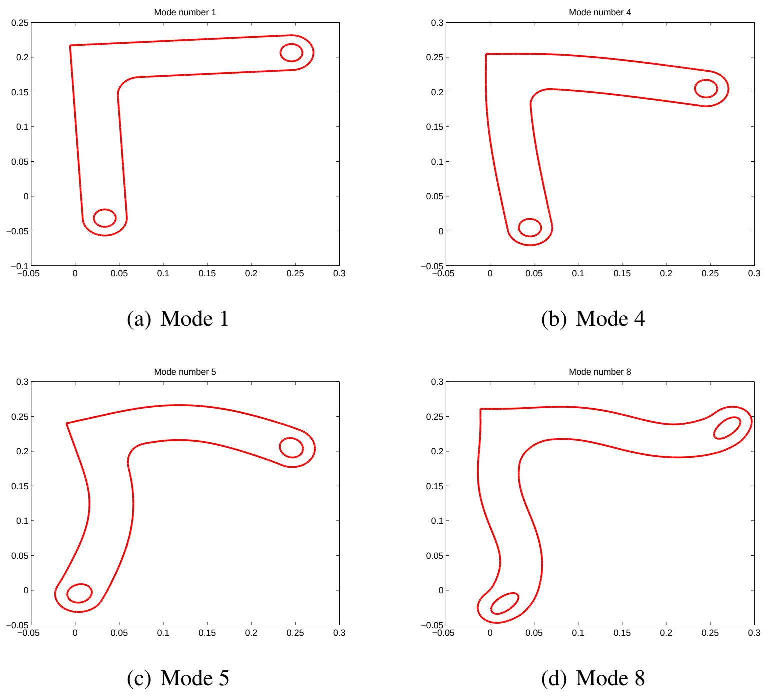


Fig. 11.4 Steel bracket and mesh.

i	Λ_i
1	0.00
2	-0.00
3	-0.00
4	1.67
5	15.07
6	35.20
7	93.94
8	154.25
9	119.29
10	185.92

Table 11.2 The ten lowest eigenvalues of the steel bracket.



(a) Mode 1 (b) Mode 4

(c) Mode 5 (d) Mode 8

Fig. 11.5 Eigenmodes 1, 4, 5, and 8 of the steel bracket.

11.8 Problems

Exercise 11.1. Given the stress field $\sigma_{11} = x_1x_2$, $\sigma_{12} = (1 - x_2^2)/2$, and $\sigma_{22} = 0$. Determine if this corresponds to a state of equilibrium under a zero body force.

Exercise 11.2. Show the vector identity $2\nabla \cdot \varepsilon(v) = \Delta v + \nabla(\nabla \cdot v)$ for $v = [v_1,\ v_2]$.

Exercise 11.3. Use the previous result to rewrite (11.13) as the single equation $\mu \Delta u + (\lambda + \mu)\nabla(\nabla \cdot u) + f = 0$.

Exercise 11.4. Show that the strain tensor $\varepsilon(u)$ is zero under the deformation

$$u = \begin{bmatrix} a \\ b \end{bmatrix} + \begin{bmatrix} 0 & -\theta \\ \theta & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

where a, b, and θ are constants. Can you give a physical interpretation of u, assuming that θ is small?

Exercise 11.5. Show that $\varepsilon(v) : I = \nabla \cdot v$.

Exercise 11.6. Show that the bilinear form can be written

$$a(u,v) = \int_\Omega 2\mu\varepsilon(u) : \varepsilon(v) + \lambda(\nabla \cdot u)(\nabla \cdot v)\, dx$$

Exercise 11.7. Verify that the conditions for the Lax-Milgram lemma are satisfied for the variational equation (11.25). For simplicity, you only have to consider the case of homogeneous Dirichlet boundary conditions $u = 0$ on the whole boundary $\partial\Omega$. *Hint:* Korn's inequality is useful.

Exercise 11.8. Calculate the element stiffness matrix K^e by hand for the triangle with corners at $(0,0)$, $(3,1)$, and $(2,2)$. Assume that

$$D = \begin{bmatrix} 4 & 1 & 0 \\ 1 & 4 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Verify that K^e has three zero eigenvalues. Can you explain why?

Exercise 11.9. Calculate by hand the element mass matrix M^e assuming a unit density on the reference triangle with vertices at origo, $(1,0)$, and $(0,1)$.

Exercise 11.10. A mesh of the square domain $\Omega = [-1,1]^2$ is obtained by typing `[p,e,t]=initmesh('squareg')`. Compute and plot the ten lowest eigenmodes on this domain. Assume elastic constants $\rho = 1$, $E = 1$, and $\nu = 0.3$. Test both clamped and stress free boundary conditions.

Chapter 12
Fluid Mechanics

Abstract In this chapter we study finite elements for incompressible fluids, that is, most liquids and gases. We start by reviewing the governing equations of mass and momentum balance and derive the Navier-Stokes equations. Restricting attention to laminar flow we then introduce the Stokes system and formulate a finite element methods for the velocity and pressure. We briefly discuss the inf-sup condition and the solution of saddle point linear systems. Finally, we introduce Chorin's projection method for simulating also time-dependent nearly turbulent fluid flow.

12.1 Governing Equations

12.1.1 Conservation of Mass

In classical physics mass can neither be destroyed nor created. This means that the mass of any small volume dx of matter (e.g., a fluid) can change over time only by flow in and out of the boundary ds. Letting u denote the flow velocity we immediately obtain the following a mass balance equation for a fluid occupying the domain $\Omega \subset \mathbb{R}^d$, with $d = 2$ or 3.

$$(\dot{\rho}, 1) + (\rho, u \cdot n)_{\partial\Omega} = 0 \tag{12.1}$$

Here, ρ is the density of the fluid. Since $dm = \rho dx$ is the mass of dx the first term represents the rate of change of mass within the domain. Further, during the small time span dt a total volume of matter of $dm = \rho u \cdot n ds$ will flow out of the surface ds, the second term represents the rate of mass loss through the domain boundary.

Using the divergence theorem on the surface integral we have

$$\dot{\rho} + \nabla \cdot (\rho u) = 0 \tag{12.2}$$

If the density ρ is constant, then this simplifies to

$$\nabla \cdot u = 0 \tag{12.3}$$

Physically this means that the volume of any small fluid particle dx does not change under deformation. Such fluids are said to be incompressible. Most everyday fluids (e.g., water) are incompressible to a very high degree.

12.1.2 Momentum Balance

Besides mass conservation a fluid also obeys conservation of momentum (i.e., Newton's second law). The momentum of a particle with mass m and velocity u is defined as the product $p = mu$. Newton's second law says that the rate of change of momentum equals the net force F acting on the particle, or $\dot{p} = F$.

Now, the momentum dp of a small volume of fluid dx is given by $dp = \rho u dx$.

Taking into account that momentum can be transported in and out of the surface $\partial\Omega$ of a domain Ω we have the following equation for momentum balance of a fluid.

$$(\dot{\rho u}, 1) + (\rho u, u \cdot n)_{\partial\Omega} = F \tag{12.4}$$

where F is the net force acting on the fluid. We can use our knowledge from mechanics to write $F = \nabla \cdot \sigma + f$ with σ the stress tensor of the fluid and f a given body load, such as gravity, for instance.

Using the divergence theorem on the surface integral we arrive at

$$(\dot{\rho u}, 1) + \nabla \cdot (\rho u u) = \nabla \cdot \sigma + f \tag{12.5}$$

Here, the right left side can be simplified by differentiating using the chain rule.

$$\begin{aligned} \dot{\rho u} + \nabla \cdot (\rho u u) &= u\dot{\rho} + \rho \dot{u} + u\nabla \cdot (\rho u) + \rho (u \cdot \nabla) u && (12.6) \\ &= \rho \dot{u} + \rho (u \cdot \nabla) u && (12.7) \end{aligned}$$

where we have used the conservation of mass equation to eliminate first and third term in the right hand side of (12.6). Thus, we end up with the momentum balance equation

$$\rho \dot{u} + \rho (u \cdot \nabla) u = \nabla \cdot \sigma + f \tag{12.8}$$

12.1.3 Incompressible Newtonian Fluids

The stresses acting on a fluid particle are of two types, namely:

- Internal stresses due to the fluid pressure.
- Viscous stresses.

Internal stresses always arise when a fluid is brought into motion, since the pressure p is changed from that existing when the fluid is at rest. The corresponding stress tensor takes the form

$$\sigma = -pI \tag{12.9}$$

with I the $d \times d$ identity tensor.

Viscosity is a measure of the resistance of a fluid to being deformed by stresses. It may be thought of as a friction caused by neighboring layers of fluid rubbing against each other. In reality it is fluid molecules with different velocities that bump into each other. Viscosity is commonly perceived as the thickness of the fluid. Thus, water is thin, having a lower viscosity, while oil is thick having a higher viscosity. All real fluids have some resistance to stress, but a fluid which has no resistance is called either inviscid or ideal.

Viscous stresses oppose deformation of neighboring fluid particles. Since a constant velocity field does not give rise to any relative movement between the fluid particles it is reasonable to assume that the stress tensor σ is related only to the velocity gradients ∇u. Clearly, the simplest assumption is that this relation is linear. Recalling that σ is symmetric gives us

$$\sigma = -pI + \mu(\nabla u + \nabla u^T) \tag{12.10}$$

where the coefficient of proportionality μ is the viscosity of the fluid. Fluids obeying this constitutive law are called Newtonian.

Now, inserting the constitutive law into the equation for balance of momentum (12.8) using that $\nabla \cdot \sigma = \mu(\Delta u + \nabla(\nabla \cdot u)) - \nabla p$, and assuming that the fluid is incompressible so that $\nabla \cdot u = 0$, we obtain a set of partial differential equations for the velocity u and pressure p,

$$\dot{u} + (u \cdot \nabla)u = \nu \Delta u - \frac{\nabla p}{\rho} + f \tag{12.11a}$$

$$\nabla \cdot u = 0 \tag{12.11b}$$

where $\nu = \mu/\rho$. These are the Navier-Stokes equations.

12.1.4 Boundary- and Initial Conditions

In order to yield a unique velocity-pressure pair (u, p) the Navier-Stokes equations must be supplemented by appropriate boundary conditions. The most common of these have names and include:

- Slip, $u \cdot n = 0$.
- No-slip, $u = g$.
- Stress free, $\sigma \cdot n = 0$.

- Do-nothing, $n \cdot \nabla u - pn = 0$.

Slip and no-slip boundary conditions apply at a solid wall with normal n. Slip boundary conditions says that the fluid flow is parallel to the boundary (i.e., orthogonal to n).

No-slip conditions prescribe that the velocity u agrees with a known vector g on the boundary. This might model fluid flow near a moving wall. Often $g = 0$ meaning that the wall has a rough surface, which prevents the fluid nearest the wall to move. Stress free and do-nothing boundary conditions are generally used on outflow boundaries. Stress free boundary conditions model free flow into a large reservoir, while do-nothing boundary conditions are used to truncate very long channel like domains.

Due to the time derivative on the velocity it is also necessary to specify initial conditions of the type $u(\cdot, t_0) = u_0$ with u_0 a given velocity at the initial time t_0.

12.2 The Stokes System

12.2.1 The Stationary Stokes System

Many applications concerns laminar fluid flow meaning that the flow is calm with essentially parallel streamlines. In such cases it is possible to omit the non-linear term $(u \cdot \nabla)u$, which governs inertial effects, from the Navier-Stokes equations. Furthermore, assuming that the flow is independent of time t we recover the stationary Stokes equations.

$$-\Delta u + \nabla p = f, \quad \text{in } \Omega \tag{12.12a}$$

$$\nabla \cdot u = 0, \quad \text{in } \Omega \tag{12.12b}$$

$$u = g, \quad \text{on } \partial\Omega \tag{12.12c}$$

where f and g are given functions. For simplicity, we assume a unit viscosity ν.

The Stokes equations equations are much easier to analyze than the Navier-stokes equations (e.g., they are linear), but they still provide a realistic model of fluid flow. This justifies our study of them.

Since only the gradient of the pressure enters the equations p is only determined up to an arbitrary constant. We say that the hydrostatic pressure level is undetermined. To fix this constant it is customary to require the pressure to have a zero mean value, that is,

$$(p, 1) = 0 \tag{12.13}$$

This is a characteristic feature of enclosed flows.

We also require the boundary condition g to satisfy $(g, n)_{\partial\Omega} = 0$.

Other types of boundary conditions than Dirichlet conditions are of course possible. For example,

$$-(n \cdot \nabla u, v) + (pn, v) = 0 \tag{12.14}$$

which is a kind of Neumann condition typically used on outflow boundaries. A nice feature with this boundary condition is that it automatically fixes the hydrostatic pressure level.

12.2.2 Variational Formulation

In order to make a variational formulation of the Stokes equations we need to introduce two function spaces V and Q for the velocity u and pressure p, respectively. Let

$$V_g = \{v \in [H^1(\Omega)]^d : v|_{\partial\Omega} = g\} \tag{12.15}$$

$$Q = \{q \in L^2(\Omega) : (q, 1) = 0\} \tag{12.16}$$

We see that the pressure space Q is the subset of L^2 functions, which have zero mean.

Now, multiplying the momentum equation $f = -\Delta u + \nabla p$ by a test function $v \in V_0$ and integrating by parts we have

$$(f, v) = (-\Delta u, v) + (\nabla p, v) \tag{12.17}$$

$$= (-n \cdot \nabla u, v)_{\partial\Omega} + (\nabla u : \nabla v) + (pn, v)_{\partial\Omega} - (p, \nabla \cdot v) \tag{12.18}$$

which, since $v = 0$ on $\partial\Omega$, simplifies to

$$(f, v) = (\nabla u : \nabla v) - (p, \nabla \cdot v) \tag{12.19}$$

Similarly, multiplying the incompressibility constraint $\nabla \cdot u = 0$ by a test function $q \in Q$ we trivially have

$$(\nabla \cdot u, q) = 0 \tag{12.20}$$

One might ask why Q is the appropriate test space for the incompressibility constraint $\nabla \cdot u = 0$. After all, the functions in Q are somewhat peculiar since they all have a zero mean value. The reason is that it suffice to test against these functions, since the variational equation is zero anyway for q constant. To see this let c be any constant and recall that by assumption $(g, n)_{\partial\Omega} = 0$. Using integration by parts we have

$$(\nabla \cdot u, c) = (u, nc)_{\partial\Omega} + (u, \nabla c) = c(g, n)_{\partial\Omega} = 0 \tag{12.21}$$

Thus, the variational formulation of (12.12) reads: find $u \in V_g$ and $p \in Q$ such that

$$a(u,v) + b(v,p) = (f,v), \quad \forall v \in V_0 \tag{12.22a}$$
$$b(u,q) = 0, \qquad \forall q \in Q \tag{12.22b}$$

where we have introduced the bilinear forms

$$a(u,v) = (\nabla u : \nabla v) \tag{12.23}$$
$$b(u,q) = -(\nabla \cdot u, q) \tag{12.24}$$

The sign of the incompressibility constraint (12.23) can be chosen arbitrarily, since the right hand side is zero anyway. Usually, $b(u,q) = 0$ is preferred over the perhaps more correct $-b(u,q) = 0$, since it gives a symmetric variational form. From a theoretical point of view it usually does not matter what the sign is, however, it can have a large impact on the numerics. Recall that symmetric matrices are often to be preferred when it comes to solving linear systems.

If the boundary data g is sufficiently smooth it can be extended form $\partial\Omega$ to Ω. We can then write $u = g + u_0$, where $u_0 \in V_0$ is a new unknown solution which is zero on the boundary. This allows us to work solely with the space V_0 with homogeneous boundary data. In this case the variational formulation takes the form: find $u_0 \in V_0$ such that

$$a(u_0,v) + b(v,p) = (f,v) - a(g,v), \quad \forall v \in V_0 \tag{12.25a}$$
$$b(u_0,q) = 0, \qquad \forall q \in Q \tag{12.25b}$$

From this we see that the effect of the inhomogeneous boundary condition can be accounted for by defining a new body force $\tilde{f}$ by $(\tilde{f},v) = (f,v) - a(g,v)$ for all $v \in V_0$. Thus, from now on we consider only the case $g = 0$ and set $V_g = V_0 = V$.

12.2.3 The Inf-Sup Condition

The existence and uniqueness of a solution (u,p) to the variational equation (12.22) depends on four conditions, namely, the boundedness and coercivity $a(\cdot,\cdot)$ on V, the boundedness of $b(\cdot,\cdot)$ on $V \times Q$, and the following result called the inf-sup condition.

Theorem 12.1. *There exist a constant $\beta > 0$ such that*

$$\beta \leq \inf_{q \in Q} \sup_{v \in V} \frac{b(v,q)}{\|q\|_Q \|v\|_V} \tag{12.26}$$

The inf-sup condition may be though of as a abstract condition of the angle between the spaces V and Q.

Establishing the inf-sup condition is difficult and outside the scope of this book.

It is easy to show that u exist and is unique. To this end let $Z = \{v \in V : \nabla \cdot v = 0\}$ be the subspace of V containing all divergence free vectors and notice that $b(v, p) = 0$ for all $v \in Z$. As a consequence the variational equation (12.22) reduces to: find $u \in Z$ such that

$$a(u, v) = (f, v), \quad \forall v \in Z \tag{12.27}$$

Now, since Z is a Hilbert space on which $a(\cdot,\cdot)$ is bounded and coercive we can simply invoke the Lax-Milgram Lemma to show existence and uniqueness of $u \in Z \subset V$.

Once it is proven that u exist it is possible to prove also existence of p using the inf-sup condition. However, this is a bit technical and we omit the proof.

12.2.4 Finite Element Approximation

In order to formulate a numerical method let $\mathscr{K}$ be a mesh of Ω. Further, to approximate the velocity and pressure let V_h and Q_h be two spaces of piecewise polynomials on $\mathscr{K}$ that approximates V and Q in some sense to be made precise.

The finite element approximation of (12.22) takes the form: find $u_h \in V_h$ and $p_h \in Q_h$ such that

$$a(u_h, v) + b(v, p_h) = (f, v), \quad \forall v \in V_h \tag{12.28a}$$

$$b(u_h, q) = 0, \quad \forall q \in Q_h \tag{12.28b}$$

Let $\{\varphi_i\}_1^n$ be a set of vector valued basis functions for V_h, and let $\{\chi_i\}_1^m$ be a set of scalar basis functions for Q_h. The finite element method (12.28) results in a linear system which can be written in block form as

$$\begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \xi \\ \varpi \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix} \tag{12.29}$$

where A is the $n \times n$ stiffness matrix, and B is the $n \times m$ divergence matrix with entries

$$A_{ij} = a(\varphi_j, \varphi_i) \tag{12.30}$$

$$B_{ij} = b(\varphi_j, \chi_i) \tag{12.31}$$

and b is the $n \times 1$ load vector with entries $b_i = (f, \varphi_i)$. Further, ξ and ϖ are $n \times 1$ and $m \times 1$ vectors containing the unknown degrees of freedom of $u_h = \sum_{j=1}^n \xi_j \varphi_j$ and $p_h = \sum_{j=1}^m \varpi_j \chi_j$, respectively.

Equation systems of the form (12.29) are called saddle-point linear systems, and are notoriously difficult to solve due to the all zero $m \times m$ lower diagonal block.

Very often the components of u_h are approximated using a single finite element space S_h with $V_h = S_h \times S_h$. Moreover, if $\{\phi_i\}_{i=1}^{o}$ is a basis for S_h, then a basis for V_h can be trivially constructed viz

$$\{\varphi_i\}_{i=1}^{n} = \left\{ \begin{bmatrix} \phi_1 \\ 0 \end{bmatrix}, \begin{bmatrix} \phi_2 \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} \phi_o \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \phi_1 \end{bmatrix}, \begin{bmatrix} 0 \\ \phi_2 \end{bmatrix}, \dots, \begin{bmatrix} 0 \\ \phi_o \end{bmatrix} \right\} \tag{12.32}$$

with $n = 2o$. In this case the saddle-point linear system (12.29) can be written

$$\begin{bmatrix} A_{11} & 0 & B_1^T \\ 0 & A_{11} & B_2^T \\ B_1 & B_2 & 0 \end{bmatrix} \begin{bmatrix} \xi_1 \\ \xi_2 \\ \varpi \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ 0 \end{bmatrix} \tag{12.33}$$

where the matrix and vector entries are given by

$$(A_{11})_{ij} = (\nabla\phi_j, \nabla\phi_i), \quad i, j = 1, 2, \dots, o \tag{12.34}$$

$$(B_s)_{ij} = -(\partial_{x_k}\phi_j, \chi_i), \quad i = 1, 2\dots, m, \; j = 1, 2\dots, o \tag{12.35}$$

$$(b_s)_i = (f_k, \varphi_i), \quad i = 1, 2, \dots, o \tag{12.36}$$

with $s = 1, 2$.

12.2.5 The Discrete Inf-sup Condition

So far we have not said anything more specific about the finite element spaces V_h and Q_h. In fact we do not even know if the finite element solution (u_h, p_h) is well defined. To assert this we must make sure that the saddle-point linear system (12.29) can be inverted. It turns out that this is equivalent to establishing a discrete inf-sup condition on V_h and Q_h. In other words, there must exist a constant $\gamma > 0$ such that

$$\gamma \leq \min_{q \in \mathbb{R}^m, q \neq 1} \max_{v \in \mathbb{R}^n, v \neq 0} \frac{|(Bv, q)|}{(q, Mq)^{1/2}(v, Av)^{1/2}} \tag{12.37}$$

where A is the stiffness matrix, B the divergence matrix, and M the $m \times m$ pressure mass matrix with entries $M_{ij} = (\chi_j, \chi_i)$. We emphasize that it is not trivial task to show this because even if we know that the inf-sup condition is satisfied on the continuous spaces V and Q, it need not hold on the discrete spaces V_h and Q_h, not even if the inclusions $V_h \subset V$ and $Q_h \subset Q$ hold. All the same, let us for the moment assume that the discrete inf-sup condition do hold and do block elimination on the $(n+m) \times (n+m)$ saddle-point linear system (12.29). From the first row we have $\xi = A^{-1}(b - B^t\varpi)$. Plugging this into the second row $B\xi = 0$, and rearranging we get the $m \times m$ linear system

$$BA^{-1}B^T\varpi = BA^{-1}b \tag{12.38}$$

for the pressure degrees of freedom ϖ. We now claim that the matrix $BA^{-1}B^T$, which is called the Schur complement, is invertible. To see this note that the discrete inf-sup condition (12.37) implies

$$0<\gamma\leq \min_{q\neq 1}\max_{v\neq 0}\frac{|(q,Bv)|}{(v,Av)^{1/2}(q,Mq)^{1/2}} \tag{12.39}$$

$$=\min_{q\neq 1}\frac{1}{(q,Mq)^{1/2}}\max_{w=A^{1/2}v,v\neq 0}\frac{|(q,BA^{-1/2}w)|}{(w,w)^{1/2}} \tag{12.40}$$

$$=\min_{q\neq 1}\frac{1}{(q,Mq)^{1/2}}\max_{w\neq 0}\frac{|(A^{-1/2}B^Tq,w)|}{(w,w)^{1/2}} \tag{12.41}$$

Here, the maximum is attained for $w=\pm A^{-1/2}B^Tq$. Using this we have

$$0<\gamma\leq \min_{q\neq 1}\frac{(A^{-1/2}B^Tq,A^{-1/2}B^Tq)^{1/2}}{(q,Mq)^{1/2}}=\min_{q\neq 1}\frac{(BA^{-1}B^Tq,q)^{1/2}}{(q,Mq)^{1/2}} \tag{12.42}$$

This shows that $BA^{-1}B^T$ is coercive and invertible.

Once ϖ has been found we can solve the $n\times n$ linear system

$$A\xi=b-B^T\varpi \tag{12.43}$$

for the velocity degrees of freedom ξ. This is of course possible since the stiffness matrix A is symmetric and positive definite.

To summarize we conclude that it is important to choose the finite element spaces so that the discrete inf-sup condition is satisfied. In the next section we present a three finite elements that has this property.

12.2.6 Three Inf-Sup Stable Finite Elements

12.2.6.1 The Taylor-Hood Element

The Taylor-Hood finite element is the standard finite element for simulating incompressible fluid flow, since it gives a good approximation of both velocity and pressure, and since it is not too numerically costly to use. The element consists of a continuous piecewise quadratic approximation of each velocity component and a continuous piecewise linear approximation of the pressure. That is, the velocity space is $V_h=\{v\in[C^0(\Omega)]^d : v|_K\in[P^2(K)]^d\}$, and the pressure space $Q_h=\{v\in C^0(\Omega) : v|_K\in P^1(K)\}$. Figure 12.1 shows the position of the velocity and pressure nodes on an element K.

Fig. 12.1 Velocity • and pressure ◯ nodes for the Taylor-Hood element.

12.2.6.2 The MINI Element

The MINI element is the simplest inf-sup stable element. It consists of a standard continuous piecewise linear approximation for both each velocity component and the pressure. However, on each element the velocity space is enriched by cubic bubble functions of the form

$$\varphi_{\text{bubble}} = \varphi_1 \varphi_2 \varphi_3 \tag{12.44}$$

where φ_i, $i = 1,2,3$, are the usual hat functions. More precisely, the velocity space is given by $V = \{v \in [C^0(\Omega)]^d : v|_K \in [P^1(K)]^d \bigoplus [B(K)]^d\}$, where $B(K) = \text{span}\{\varphi_{\text{bubble}}\}$ is the space of bubble functions on element K. Needless to say, the bubble function has earned its name from the fact that it looks like a bubble. By construction the bubble function φ_{bubble} vanishes on the boundary ∂K, which is important since it allows all bubble functions to be eliminated from the saddle-point linear system before attempting to invert it. The MINI element has become popular because it is easy to implement. Unfortunately, it is also known for giving a poor approximation of the pressure. The velocity and pressure nodes on K are shown in Figure 12.2.

12.2.6.3 The Non-conforming $P^1 - P^0$ Element

The non-conforming $P^1 - P^0$ element is constructed by approximating the velocity by Crouzeix-Raviart functions and the pressure by piecewise constants. This element has the desirable property of being able to yield a finite element solution that is exactly divergence free. As we shall see shortly it is also fairly easy to implement. The node locations are shown in Figure 12.3.

Fig. 12.2 Velocity • and pressure ◯ nodes for the MINI element.

Fig. 12.3 Velocity • and pressure ◯ nodes for the Crouzeix-Raviart element.

12.2.7 Computer Implementation

12.2.7.1 The Lid-Driven Cavity

To get some hands on experience with numerics for fluid flow let us implement a Stokes solver and simulate a classical benchmark problem called the lid-driven cavity problem. The problem setup is very simple. A square cavity $\Omega = [-1,1]^2$ is filled with a viscous incompressible fluid. No-slip boundary conditions apply on all four sides of the cavity. On the bottom and walls $u = 0$, while $u_1 = 1$ and $u_2 = 0$ on the lid. This creates a swirling flow inside the cavity. There is no body load. The task is to compute the velocity field and pressure distribution.

Let us write our solver based on the non-conforming $P^1 - P^0$ element. For this purpose, we must compute the matrices A_{11}, and B_s, $s = 1,2$, defined by (12.34) and (12.35), respectively. Moreover, the functions ϕ_i and χ_i occurring in the matrix entries of these matrices should be the basis functions for the Crouzeix-Raviart space and the space of piecewise constants, respectively. Recall that the former space has dimension n_e the number of edges, and the latter has dimension n_t the number of elements. Indeed, since the nodes of the Crouzeix-Raviart element is associated with the edges, we must number these before the assembly. Reusing our routine `Tri2Edge` the setup of the mesh and numbering of the edges is done with the following lines of code.

```
function CRStokesSolver()
[p,e,t]=initmesh('squareg'); % mesh square [-1,1]^2
t2e=Tri2Edge(p,t); % triangle-to-edge adjacency
nt=size(t,2); % number of triangles
ne=max(t2e(:)); % number of edges
```

Next we allocate the matrices A_{11}, B_1 and B_2, and a vector holding element areas.

```
A=sparse(ne,ne);
Bx=sparse(nt,ne);
By=sparse(nt,ne);
areas=zeros(nt,1);
```

The assembly of these matrices is done as usual. We start by looping over the elements. For each element we fetch the vertex numbers and the vertex coordinates.

```
for i=1:nt
  vertex=t(1:3,i);
  x=p(1,vertex);
  y=p(2,vertex);
```

Now, on each element K the three non-zero Crouzeix-Raviart basis functions $\phi_i = S_i^{CR}$, are given by

$$S_i^{CR} = -\hat{\varphi}_i + \hat{\varphi}_j + \hat{\varphi}_k \tag{12.45}$$

where $\hat{\varphi}_i$ are the usual hat functions and with cyclic permutation of i, j, k over $\{1,2,3\}$. Since the gradient of a hat function is the constant vector $\nabla\hat{\varphi}_i = [b_i, c_i]^T$ we readily find that

$$\nabla S_i^{CR} = [-b_i + b_j + b_k, -c_i + c_j + c_k]^T \tag{12.46}$$

Here, we can use the subroutine `Gradients` to compute the constants b_i and c_i.

```
  [area,b,c]=Gradients(x,y);
  Sx=[-b(1)+b(2)+b(3); b(1)-b(2)+b(3); b(1)+b(2)-b(3)];
  Sy=[-c(1)+c(2)+c(3); c(1)-c(2)+c(3); c(1)+c(2)-c(3)];
```

The 3×3 element matrix $(A_{11}^K)_{ij} = (\nabla S_j^{CR}, \nabla S_i^{CR})_K$ is then given simply by

```
  AK = (Sx*Sx'+Sy*Sy')*area;
```

To assemble this we retrieve the edge numbers on this element and add A^K to the appropriate matrix entries.

```
    edges=t2e(i,:);
    A(edges,edges)=A(edges,edges)+AK;
```

Next we observe that the piecewise constant basis function χ_i is just the characteristic function of element K_i, that is, $\chi_i = 1$ on K_i and zero otherwise. Thus, on K_i we set $\chi_i = 1$ and so it is a piece of cake to compute and the assemble the 1×3 element matrices $(B_1^K)_{ij} = -(\partial_{x_1} S_j^{CR}, \chi_i)_K$ and$(B_2^K)_{ij} = -(\partial_{x_2} S_j^{CR}, \chi_i)_K$. We also store the element area.

```
    Bx(i,edges)=-Sx'*area;
    By(i,edges)=-Sy'*area;
    areas(i)=area;
end
```

We can now build the big saddle-point linear system (12.33).

```
nu=0.1; % viscosity parameter
LHS=[nu*A sparse(ne,ne) Bx';
     sparse(ne,ne) nu*A By';
     Bx By sparse(nt,nt)];
RHS=zeros(ndofs,1);
```

Should be attempt so solve this we would find that the matrix `LHS` is singular. This is of course due to the fact that we have neither enforced boundary conditions on the velocity nor a zero mean on the pressure.

In the discrete setting zero mean value on p_h means that

$$(p_h, 1) = \sum_{K=1}^{n_t} \varpi_K (\chi_i, 1)_K = a^T \varpi = 0 \tag{12.47}$$

where a is the vector `areas`. To enforce this constraint we augment the saddle-point linear system with this equation together with a Lagrangian multiplier μ to get

$$\begin{bmatrix} A_{11} & 0 & B_1 & 0 \\ 0 & A_{11} & B_2 & 0 \\ B_1^T & B_2^T & 0 & a \\ 0 & 0 & a^T & 0 \end{bmatrix} \begin{bmatrix} \xi_1 \\ \xi_2 \\ \varpi \\ \mu \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ 0 \\ 0 \end{bmatrix} \tag{12.48}$$

Here, the load vectors b_1 and b_2 are zero since there is no body load. The code to modify the linear system looks like

```
last=[zeros(2*ne,1); areas]; % last row and column
LHS=[LHS last; last' 0];
RHS=[RHS; 0];
```

The last thing we need to do is to enforce the no-slip boundary condition $u = g$. We do this as described by first writing $u_h = g_h + u_0$ with g_h a Crouzeix-Raviart interpolant of g, and then modifying the right hand side vector `RHS` accordingly. The setting up of g_h is a bit messy since we do not have the node coordinates available. All the same, the following piece of code computes these.

```
i=t(1,:); j=t(2,:); k=t(3,:); % triangle vertices
edgrow=t2e(:); % all edges in a long row
nstart=[j i i]; % start vertices of all edges
nstop =[k k j]; % stop
xmid=(p(1,start)+p(1,stop))/2; % x-coordinates of
                               %   edge mid-points
ymid=(p(2,start)+p(2,stop))/2; % y-
[edgrow,idx]=unique(edgrow); % remove duplicate edges
xmid=xmid(idx);
ymid=ymid(idx);
```

The node numbers and node values of g_h can now be found by looping over the edges.

```
fixed=[]; % fixed nodes
gvals=[]; % nodal values of g
for i=1:length(edgrow) % loop over edges
  r=edgrow(i); % node number
  x=xmid(i); % node x-coordinate
  y=ymid(i); %      y-
  if (x<-0.99 | x>0.99 | y<-0.99 | y>0.99) % boundary
    fixed=[fixed; r; r+ne];
    u=0; v=0; % bc values
    if (y>0.99), u=1; end % lid
    gvals=[gvals; u; v];
  end
end
```

The modification of the linear system for the boundary condition is as usual.

```
neq=2*ne+nt+1; % number of equations
free=setdiff([1:neq],fixed);
RHS=RHS(free)-LHS(free,fixed)*gvals;
LHS=LHS(free,free);
SOL=zeros(neq,1); % allocate solution
SOL(fixed)=gvals; % insert no-slip values
SOL(free)=RHS\LHS; % solve linear system
```

Finally, to plot the velocity and pressure we type

```
U=SOL(1:ne); V=SOL(1+ne:2*ne); P=SOL(2*ne+1:2*ne+nt);
figure(1), pdesurf(p,t,P')
figure(2), quiver(xmid,ymid,U',V')
```

Running the code we get the velocity and pressure of Figures 12.4 and 12.5. As expected the velocity glyphs shows a swirling fluid due to the moving lid. The pressure distribution shows a high pressure in the upper right corner of the cavity, where the fluid crashes into the right wall. Similarly, a low pressure is visible in the upper left corner, where the fluid is swept away from the left wall by the moving lid.

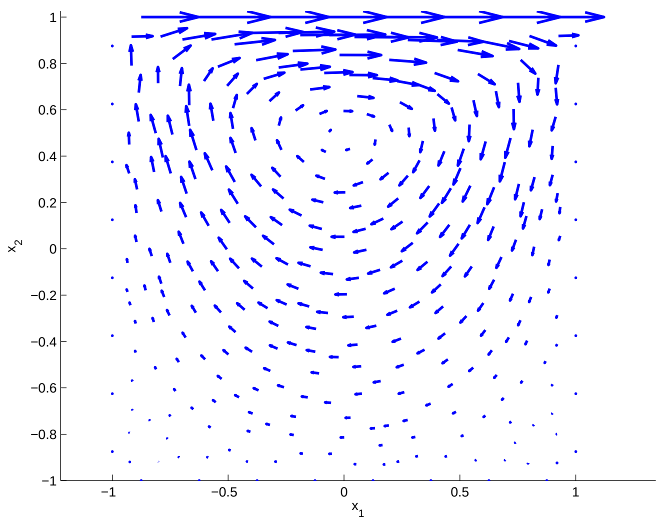


Fig. 12.4 Glyphs of velocity u_h in the cavity.

To check our implementation we can make a test and compute the null space of the matrix $B^T = [B_1\ B_2]^T$, which is the discrete gradient operator $-\nabla$. Recall that this operator determines the pressure p_h and should therefore have a null space consisting of the single vector 1, or a scaled copy of this vector. This is the discrete hydrostatic pressure mode, which we eliminated by adding the zero mean value constraint for p_h. The B^T matrix can be extracted from the saddle-point linear system. In doing so we must remember that the matrix `LHS` shrunk when we removed the boundary conditions. The null space is computed using the `null` command.

```
nfix=length(fixed);
n=2*ne-nfix; % number of free velocity nodes
Bt=LHS(1:n,n+1:n+nt); % extract B'
nsp=null(full(Bt)) % compute null space of B'
```

Indeed, the result of executing these lines is the vector `nsp`, which is a constant times the vector 1. This is a necessary condition for a finite element to be inf-sup stable. Of course it does not prove inf-sup stability, but it is one way of testing the code. Other ways to validate the code include computing the eigenvalues of the Schur complement, which should be positive, or checking that the Lagrange multiplier is close to the machine precision.

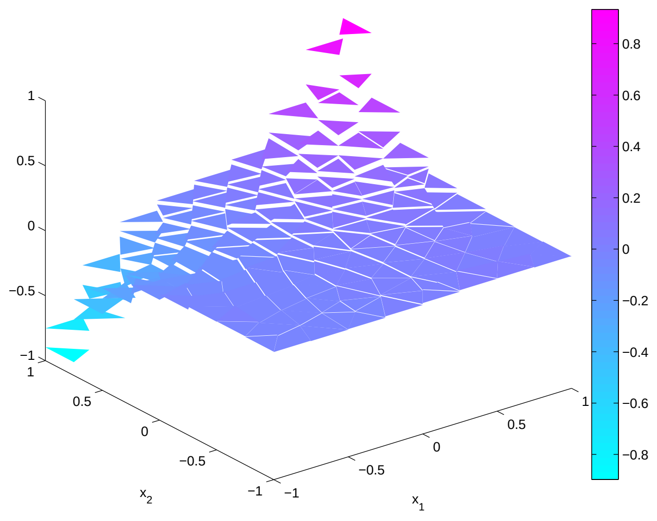


Fig. 12.5 Pressure distribution p_h in the cavity.

In this context we mention that the naive choice of equal order polynomial spaces for both the velocity and pressure does not satisfy the inf-sup condition. Thus, it is not possible to use piecewise linears for both u_h and p_h. Doing so yields a gradient matrix B^T with a too large null space. The basis vectors for this larger null space are called spurious pressure modes and pollute p_h. Typically, such artificial pressures are oscillating. The problem with equal order interpolation is that the the discrete inf-sup constant γ is proportional to the the mesh size h, and vanish as h tends to zero. As a consequence the numerical stability is lost under mesh refinement. A remedy for this is to use the GLS stabilization technique.

12.3 The Navier-Stokes Equations

Having studied some of the basic features and difficulties with simulating incompressible fluid flow we now turn to consider the full fledged Navier-Stokes equations, which in addition to the Stokes system are both non-linear and time-dependent. Indeed, the Navier-Stokes equations are so complex that their numerical study has grown into a discipline of its own called computational fluid dynamics, abbreviated CFD. This is a vast field involving continuum mechanics, thermodynamics, mathematics, and computer science. The applications are many and ranges from optimizing the mix of air and fuel in turbine engines to predicting the stresses in the walls of human blood vessels. However, the grand theoretical challenge for CFD is the understanding of turbulence. Turbulence is the highly chaotic flow pattern exhibited by a fast moving fluid with low viscosity. Think of the irregular plume of smoke rising from a cigarette, for example. Physically, turbulence is caused by a combination of dissipation of energy into heat at the microscopic level, with a large transport of momentum at the macroscopic level. The basic measure of the tendency for a fluid to develop a turbulent flow is the dimensionless Reynolds number, defined as $\mathrm{Re} = UL/\nu$, where ν is the viscosity and U and L is a representative velocity and length scale, respectively. A high Reynolds number implies a turbulent flow, while a low implies a steady state laminar flow. Because turbulence occurs on all length scales, down to the smallest so-called Kolmogorov scale $\mathrm{Re}^{-3/4}$, it is very difficult to simulate using finite elements on a perhaps coarse mesh. This is further complicated by the fact that turbulent flows are highly convective, which requires stabilization of the corresponding finite element methods with subsequent potential loss of accuracy. To remedy this substantial efforts have been made to model the effect of turbulence on the small scales by statistical means and derive additional terms supplementing the original equations. This has lead to turbulence models which hope to account for turbulence effects on average. In the simplest case this amounts to changing the viscosity ν to $\nu + \nu_T$, where ν_T is a variable eddy viscosity depending on the magnitude of the local velocity gradients. This is the frequently used Smagorinsky turbulence model. Obviously, there is much more to say on this matter, but we shall not attempt to do so here. Suffice it to say that many important fluid mechanic applications are somewhere in between laminar and turbulent.

For completeness we recall that the Navier-Stokes equations takes the form

$$\dot{u} + (u \cdot \nabla)u + \nabla p - \nu \Delta u = f, \quad \text{in } \Omega \times I \tag{12.49a}$$

$$\nabla \cdot u = 0, \quad \text{in } \Omega \times I \tag{12.49b}$$

$$u = g, \quad \text{in } \Gamma_D \times I \tag{12.49c}$$

$$\nu n \cdot \nabla u - pn = 0, \quad \text{in } \Gamma_N \times I \tag{12.49d}$$

$$u = u_0, \quad \text{in } \Omega, \text{ for } t = 0 \tag{12.49e}$$

where ν is viscosity, u and p the sought velocity and pressure, and f a given body force. We assume that boundary $\partial\Omega$ of the domain Ω is divided into two parts Γ_D and Γ_N associated with the no-slip and the do-nothing boundary conditions (12.49c) and (12.49d) with g is a given function describing the velocity on Γ_D. Typically, Ω is a channel and Γ_D denotes either the rigid walls of the channel, with $g = 0$, or the inflow region, with g the inflow velocity profile, while Γ_N denotes the outlet with the boundary condition $\nu n \cdot \nabla u - pn = 0$. The velocity at time $t = 0$ is given by the initial condition u_0 and $I = (0, T]$ is the time interval with final time T.

12.3.1 Chorin's Projection Method

There are many ways to derive a numerical method for the Navier-Stokes equations and it is not easy to know which one is the best. For example, should we use Newton's method or fixed-point iteration for the non-linearity, a GLS method or some kind of inf-sup stable element, implicit or explicit time stepping? Needless to say, each of these choices has its own pros and cons regarding accuracy and computational cost and a balance has to be struck as usual. Here, we shall favor computational speed and present a simple method called Chorin's projection method for discretizing the Navier-Stokes equations. The basic idea is as follows.

Discretizing the momentum equation (12.49a) in time using the forward Euler method we have the time stepping scheme

$$\frac{u_{n+1} - u_l}{k_l} + (u_l \cdot \nabla)u_l + \nabla p_l - \nu \Delta u_l = f_l \tag{12.50}$$

where k_l is the timestep and the subscript l indicates the iterate. Now, adding and subtracting a tentative velocity u_* in the discrete time derivative $k_l^{-1}(u_{l+1} - u_l)$ we further have

$$\frac{u_{l+1} - u_* + u_* - u_l}{k_l} + (u_l \cdot \nabla)u_l + \nabla p_l - \nu \Delta u_l = f_l \tag{12.51}$$

Obviously, this equation holds if

$$\frac{u_* - u_l}{k_l} = -(u_l \cdot \nabla)u_l + \nu \Delta u_l + f_l \tag{12.52}$$

and

$$\frac{u_{l+1} - u_*}{k_l} = -\nabla p_l \tag{12.53}$$

hold simultaneously.

The decomposition of (12.49a) into (12.52) and (12.53) is called operator splitting. The rationale is that we get a decoupling of the diffusion and convection of the velocity, and the pressure acting to enforce the incompressibility constraint. Thus, assuming we know u_l, we can compute u_* from (12.52) separately without having to worry about the pressure. However, to determine also the pressure we take the divergence of (12.53), yielding

$$\nabla \cdot \frac{u_{l+1} - u_*}{k_l} = -\nabla \cdot (\nabla p_l) \tag{12.54}$$

Now, since we desire $\nabla \cdot u_{l+1} = 0$ this reduces to

$$-\nabla \cdot \frac{u_*}{k_l} = -\Delta p_l \tag{12.55}$$

It follows that the pressure p_l can be determined from a Poisson type equation. In fact (12.55) is frequently referred to as the Pressure Poisson Equation (PPE). Thus, given u_* we can solve (12.55) to get a pressure p_l which makes the next velocity

u_{l+1} divergence free. Since p_l is manufactured from the tentative velocity u_*, it is not the actual pressure p, but at best a first order approximation in time.

The actual computation of u_{l+1} is done by reusing (12.53), but now in the form

$$u_{l+1} = u_* - k_l \nabla p_l \tag{12.56}$$

This line of reasoning leads us to the following algorithm:

Algorithm 25 Chorin's Projection Method

```
1: Given the initial condition $u_0 = 0$.
2: **for** $n = 1,2,3,\ldots$ **do**
3:     Compute the tentative velocity $u_*$ from
           $$\frac{u_* - u_l}{k_l} = -(u_l \cdot \nabla)u_l + \nu \Delta u_l + f_l \tag{12.57}$$
4:     Solve the pressure Poisson equation
           $$-\nabla \cdot u_* = -k_l \Delta p_l \tag{12.58}$$
5:     Update the velocity
           $$u_{l+1} = u_* - k_l \nabla p_l \tag{12.59}$$
6: **end for**
```

The boundary conditions for u_* and p_l are not clear and has been the source of some controversy. The simplest way of enforcing these is to put the Dirichlet, or no-slip, velocity boundary conditions (12.49d) on u_*, and a Neumann boundary condition $n \cdot \nabla p_l = 0$ on the pressure. The exception is at the outflow, where the do-nothing boundary condition (12.49e) is imposed term by term by assuming $n \cdot \nabla u_l = 0$ and $p_l = 0$. This generally means that u_{l+1} will not satisfy the velocity boundary conditions other than in a vague sense. The cause of controversy is the zero Neumann boundary condition for the pressure, which is unphysical and leads to a poor quality of both p_l and u_{l+1} near the boundary. This has raised questions of the validity of the projection method. Numerous methods have been suggested to remedy this with, at least, partial success.

12.3.1.1 The Discrete Chorin Projection Method

To obtain a fully discrete method we apply finite elements to Algorithm 12.3.1. Therefore, let V_h be the usual space of piecewise linears with the hat function basis $\{\varphi_i\}_{i=1}^{n_p}$ on a mesh $\mathscr{K}$ of Ω. A nice thing with operator splitting it that it allows us to use equal order polynomial spaces for both the velocity and pressure. This seemingly circumvents the cumbersome inf-sup condition. We say seemingly because spurious pressure modes may still occur if the time step k_l is much smaller than the mesh size h. However, as we shall see the ability to use the same space for both u and p allows for great simplicity when it comes to implementation. Thus, we set

$$u_{1,l} \approx \sum_{j=1}^{n_p} (\xi_{1,l})_j \varphi_j, \quad u_{2,l} \approx \sum_{j=1}^{n_p} (\xi_{2,l})_j \varphi_j, \quad p_l \approx \sum_{j=1}^{n_p} (\varpi_l)_j \varphi_j \tag{12.60}$$

with a similar representation for u_*.

Next we observe that (12.52) decouples into one equation for $u_{1,*}$ and one for $u_{2,*}$. After finite element discretization and in matrix notation these equations take the form

$$M\xi_{1,*} = M\xi_{1,l} - k_l(C_l + \nu A)\xi_{1,l} + b_1 \tag{12.61}$$

$$M\xi_{2,*} = M\xi_{2,l} - k_l(C_l + \nu A)\xi_{2,l} + b_2 \tag{12.62}$$

where M is the mass matrix, A the stiffness matrix, and $C_l = C(u_l)$ the convection matrix with convection field u_l. Note that C_l depends on the current velocity and must be reassembled at each timestep n. As usual the load vectors b_s, $s = 1,2$, contain contributions from any body force f.

As said before the PPE (12.55) is a standard Poisson equation, yielding the matrix form

$$A\varpi_l = -(B_1\xi_{1,*} + B_2\xi_{2,*})/k_l \tag{12.63}$$

where A again is the stiffness matrix, and B_s are convection matrices with corresponding convection fields $[1,0]$ for $s = 1$ and $[0,1]$ for $s = 2$. Of course this equation has to be adjusted for boundary conditions (i.e., $\varpi = 0$ on Γ_l) to yield a unique solution.

Finally, the discrete form of the update (12.56) is given by

$$M\xi_{1,l+1} = M\xi_{1,*} - k_l B_1 \varpi_l \tag{12.64}$$

$$M\xi_{2,l+1} = M\xi_{2,*} - k_l B_2 \varpi_l \tag{12.65}$$

The time step of the presented numerical method is limited by the use of the forward Euler scheme. For numerical stability it is necessary that the time step k_l is of magnitude h/u for convection dominated flow with $\nu < uh$, and h^2/ν for diffusion dominated flow with $\nu \geq uh$.

12.3.2 Computer Implementation

12.3.2.1 The DFG Benchmark

We now turn to the practical implementation of the Chorin projection method described above. As test problem we use the DFG benchmark, which is channel flow around a cylinder. The flow is assumed to be two-dimensional. The channel is rectangular with length 2.2 and height 0.41. At the point $(0.2, 0.2)$ is a circle with diameter 0.1. The fluid has viscosity $\nu = 0.001$ and unit density. On the upper and lower wall and on the cylinder a zero no-slip boundary condition is prescribed. A parabolic inflow profile with maximum velocity $U_{\max} = 0.3$ is prescribed on the left wall

$$u_1 = \frac{4U_{\max}y(0.41 - y)}{0.41^2}, \quad u_2 = 0 \tag{12.66}$$

The boundary conditions on the right wall is of do-nothing type, since this is the outflow. There are no body forces. Zero initial conditions are assumed.

The channel geometry is output from the routine `DFGg` listed in the Appendix. We start writing our solver by calling this routine, creating the mesh, and extracting the number of nodes and the node coordinates from the point matrix `p`.

```
function NSChorinSolver()
channel=DFGg();
[p,e,t]=initmesh(channel,'hmax',0.25);
np=size(p,2);
x=p(1,:);
y=p(2,:);
```

The zero boundary condition on the pressure is most easily enforced by adding large weights, say 10^6, to the diagonal entries of A corresponding to nodes on the outflow. This penalizes any deviation from zero of the pressure in these nodes. It is convenient to store the weights in a diagonal matrix R, which can be built with the following lines of code.

```
out=find(x>2.199); % nodes on outflow
wgts=zeros(np,1); % weights
wgts(out)=1.e+6;
R=spdiags(wgts,0,np,np); % diagonal penalty matrix
```

Moreover, the boundary conditions on the velocity can be be enforced little simpler than usual due to the explicit time stepping. In each time step we can simply zero out any current value of the no-slip nodes and replace with the correct boundary values. To do so we need two vectors `mask` and `g` to identify nodes with no-slip boundary conditions and to store the corresponding nodal value.

```
in =find(x<0.001); % nodes on inflow
bnd=unique([e(1,:) e(2,:)]); % all nodes on boundary
bnd=setdiff(bnd,out); % remove outflow nodes
mask=ones(np,1); % a mask to identify no-slip nodes
mask(bnd)=0; % set mask for no-slip nodes to zero
x=x(in); % x-coordinate of nodes on inflow
y=y(in); % y-
Umax=0.3; % maximum inflow velocity
g=zeros(np,1); % no-slip values
```

```
g(in)=4*Umax*y.*(0.41-y)/0.41^2; % inflow profile
```

The assembly of all matrices M, A, C_l, and B_s is easy to do by using the built-in routine `assema` for A and M, and our own `ConvMat2D` for B_s and C_l. To speed up the computation we lump the mass matrix M.

```
[A,crap,M]=assema(p,t,1,0,1);
Bx=ConvMat2D(p,t,ones(np,1),zeros(np,1));
By=ConvMat2D(p,t,zeros(np,1),ones(np,1));
```

Using these data structures the actual time loop with the projection scheme can be very compactly written.

```
dt=0.01;  % time step
nu=0.001; % viscosity
V=zeros(np,1); % x-velocity
U=zeros(np,1); % y-
for n=1:100
  % assemble convection matrix
  C=ConvMat2D(p,e,t,U,V);
  % compute tentative velocity
  U=U-dt*(nu*A+C)*U./M;
  V=V-dt*(nu*A+C)*V./M;
  % enforce no-slip BC
  U=U.*mask+g;
  V=V.*mask;
  % solve PPE
  P=(A+R)\-(Bx*U+By*V)/dt;
  % update velocity
  U=U-dt*(Bx*P)./M;
  V=V-dt*(By*P)./M;
  pdeplot(p,e,t,'flowdata',[U V]),axis equal,pause(.1)
end
```

The setup gives a Reynolds number of $\mathrm{Re} = 20$ with the characteristic velocity $U = \frac{2}{3}U_{\max} = 0.2$ the mean of the parabolic profile and $L = 0.1$ the cylinder diameter. This is a low Reynolds number and we expect to see a laminar flow. Running the code and simulating the flow during one second we obtain the results of Figures 12.6-12.8. Due to the low Reynolds number a steady state flow has evolved and from the glyphs plot we see that it is indeed laminar. As we might have anticipated the pressure isocontours shows a high pressure in front of the cylinder and a low pressure behind it. In this region we also see a small wake with recirculating flow forming. This is typical for incompressible fluid flow.

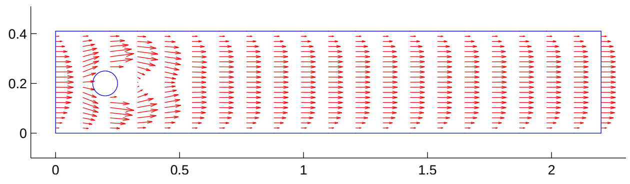


Fig. 12.6 Velocity glyphs for the DFG benchmark (Re=20).

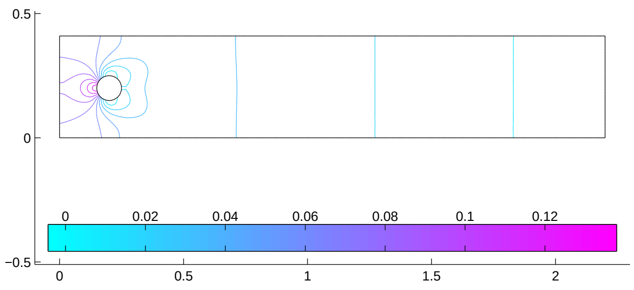


Fig. 12.7 Isocontours of the pressure (Re=20).

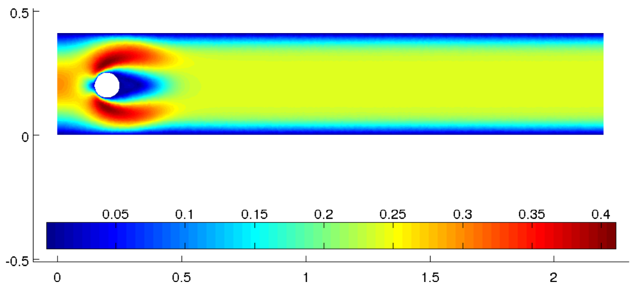


Fig. 12.8 Magnitude of the velocity (Re=20).

12.4 Problems

Exercise 12.1. Formulate a finite element approximation of the Stokes equations using the Taylor-Hood element. In particular, deduce the entries of the saddle-point linear system, resulting from finite element discretization.

Exercise 12.2. Formulate a GLS finite element approximation of the Stokes equations using piecewise linears for both the velocity components and the pressure. Can you guess the dependence between the GLS stability parameter δ and the mesh size h?

Exercise 12.3. Modify `CRStokesSolver` and solve the following problem called the colliding flow problem. The domain is the square $\Omega = [-1,1]^2$, with Dirichlet boundary conditions on the whole boundary $\partial\Omega$ given by the manufactured solution

$$u_1 = 20x_1x_2^3, \quad u_2 = 5x_1^4 - 5x_2^4, \quad p = 60x_1^2x_2 - 20x_2^3$$

which satisfies the Stokes equations with $\nu = 1$, zero body force $f = 0$, and zero mean pressure.

Exercise 12.4. A simple way of iteratively solving saddle-point linear systems of the form (12.33) is the Uzawa method, which is defined by the following iteration scheme. Set $u^0 = p^0 = 0$. For $k = 1,2,\dots$ until convergence do

$$\begin{aligned}\xi^k &= \xi^{k-1} + A^{-1}(b - A\xi^k - B^T\varpi^{k-1})\\ \varpi^k &= \varpi^{k-1} + \tau M^{-1}B\xi^k\end{aligned}$$

where $0 < \tau < 2\nu$ is a relaxation parameter, and M a preconditioner.

Write a routine `DoUzawa` for computing the solution `SOL` to the saddle-point linear system `LHS*SOL=RHS` in `CRStokesSolver`. The calling syntax should be `SOL(free)=doUzawa(A,Bt,b,areas,nu);`. The relevant matrices and vectors can be extracted from `LHS` and `RHS` with the code

```
A=LHS(1:n,1:n); Bt=LHS(1:n,n+1:n+nt); b=RHS(1:n);
```

Note that the zero mean pressure condition must be enforced at each iteration k. That is, the constant vector 1 must be filtered out of ϖ^k. This can be done by setting

$$\varpi^k = \varpi^k - (a^T\varpi^k)/(a^T 1)1$$

at the end of each iteration. Here, a is the `areas` vector.

For simplicity, let $M = \text{diag}(a)$.

Exercise 12.5. How would the Chorin projection method look with Euler backward time stepping? What is the pros and cons of this as compared to Euler forward time stepping?

Exercise 12.6. Run a sequence of simulations on the DFG benchmark with varying viscosity from $\nu = 0.1$ to 0.005. In each run make 1000 timesteps using $k_l = 0.01$. Study the transition from laminar to almost turbulent flow when you decrease ν. Make plots of the velocity and pressure.

Exercise 12.7. Simulate the Lid-Driven cavity problem using Chorin's projection method and with viscosity $\nu = 0.1$ and 0.005. To fix the pressure you can set $p = 0$ at $(-1,0)$. Make plots of the velocity magnitude. Can you say something about the effect of the non-linear term $u(\cdot\nabla u)$?

Exercise 12.8. Since the Navier-Stokes equations are non-linear it is possible to use Newton's method to solve them. This is particularly effective in the stationary case. However, this requires the linearization of the 3×1 vector $[-\nu\Delta u + (u \cdot \nabla u) + \nabla p, \nabla \cdot u]^T$. Do this by setting $u_i = u_i^0 + \delta u_i$, $i = 1,2$, and $p = p^0 + \delta p$, and discard all terms proportional to δ^2.

Appendix A
Some Additional Matlab Code

A.1 Tri2Edge.m

The following routine numbers the edges of a triangle mesh.

```
function edges = Tri2Edge(p,t)
np=size(t,2); % number of vertices
nt=size(t,2); % number of triangles
i=t(1,:); % i=1st vertex within all elements
j=t(2,:); % j=2nd
k=t(3,:); % k=3rd
A=sparse(j,k,-1,np,np);   % 1st edge is between (j,k)
A=A+sparse(i,k,-1,np,np); % 2nd                (i,k)
A=A+sparse(i,j,-1,np,np); % 3rd                (i,j)
A=-((A+A.')<0);
A=triu(A); % extract upper triangle of A
[r,c,v]=find(A); % rows, columns, and values(=-1)
v=[1:length(v)]; % renumber values (ie. edges)
A=sparse(rows,cols,entries,np,np); % reassemble A
A=A+A'; % expand A to a symmetric matrix
edges=zeros(nt,3);
for k=1:nt
  edges(k,:)=[A(t(2,k),t(3,k))
              A(t(1,k),t(3,k))
              A(t(1,k),t(2,k))]';
end
```

Input is the standard point and triangle matrix `p` and `t`. Output is a $n_t \times 3$ matrix, with n_t the number of triangles, `edges` contaning the edge numbers. In element i the global edge number of local edge j is given by `edges(i,j)`. In triangle i local edge j lies opposite local vertex j.

A.2 Tri2Tri.m

The following routine finds neighbouring elements in a triangle mesh.

```
function neighbors = Tri2Tri(p,t)
edges=tri2edg(p,t); % get edge numbers
ned=max(edges(:)); % number of edges
e1=edges(:,1); e2=edges(:,2); e3=edges(:,3);
nel=size(t,2); % number of edges
tris=[1:nel]; % all triangle numbers
% Build edge-to-triangle adjacency matrix A.
% If edge i is local edge j, j=1,2,3, in triangel k,
% then A(i,k)=j.
A=sparse(e1,tris,1,ned,nel);
A=A+sparse(e2,tris,2,ned,nel);
A=A+sparse(e3,tris,3,ned,nel);
neighbors=-ones(nel,3); % allocate element neighbours
for i=1:ned % loop over edges
  % Get elements sharing edge i.
  [crap,elnbrs,locedgs]=find(edg2tri(i,:));
  if length(elnbrs)==2 % edge i is shared by 2 elements,
                       % so they are neighbors
    neighbors(elnbrs(1),locedgs(1))=elnbrs(2);
    neighbors(elnbrs(2),locedgs(2))=elnbrs(1);
  end
end
```

Input is the standard point and triangle matrix `p` and `t`. Output `neighbors` is a $n_t \times 3$ matrix, with n_t the number of triangles, in which row i contanins the three element neighbours to element i. No neighbour is indicated by -1. Each row is ordered in the sense that the first neighbour shares edge one with the element, the second neighbour shares edge two, and so on.

A.3 Dslit.m

Geometry matrix for the double slit geometry.

```
function g = Dslit()
g=[2        0    1.0000         0         0  1  0
   2   1.0000    1.0000         0    1.0000  1  0
   2   1.0000         0    1.0000    1.0000  1  0
   2  -0.2500         0    0.3333    0.3333  2  0
   2        0   -0.2500    0.4167    0.4167  2  0
   2  -0.2500         0    0.5833    0.5833  3  0
```

```
    2        0   -0.2500    0.6667    0.6667  3  0
    2        0         0         0    0.3333  0  1
    2        0         0    0.3333    0.4167  2  1
    2        0         0    0.4167    0.5833  0  1
    2        0         0    0.5833    0.6667  3  1
    2        0         0    0.6667    1.0000  0  1
    2  -0.2500   -0.2500    0.3333    0.4167  0  2
    2  -0.2500   -0.2500    0.5833    0.6667  0  3]';
```

A.4 Airfoil.m

Geometry matrix for a wing.

```
function g=Airfoil()
g=[2   17.7218   16.0116    1.5737    1.6675    1    0
   2   16.0116    9.0610    1.6675    1.3668    1    0
   2    9.0610   -0.5759    1.3668   -0.1102    1    0
   2   -0.5759   -9.5198   -0.1102   -1.8942    1    0
   2   -9.5198  -15.6511   -1.8942   -2.5938    1    0
   2  -15.6511  -18.1571   -2.5938   -1.7234    1    0
   2  -18.1571  -16.9459   -1.7234    0.2051    1    0
   2  -16.9459  -12.4137    0.2051    2.2238    1    0
   2  -12.4137   -5.4090    2.2238    3.4543    1    0
   2   -5.4090    2.8155    3.4543    3.5046    1    0
   2    2.8155   10.6777    3.5046    2.6664    1    0
   2   10.6777   16.3037    2.6664    1.7834    1    0
   2   16.3037   17.7218    1.7834    1.5737    1    0
   2  -30.0000   30.0000  -15.0000  -15.0000    1    0
   2   30.0000   30.0000  -15.0000   15.0000    1    0
   2   30.0000  -30.0000   15.0000   15.0000    1    0
   2  -30.0000  -30.0000   15.0000  -15.0000    1    0}';
```

A.5 RectCirc.m

Geometry matrix for a rectangle with a circle cut-out.

```
function g = RectCirc()
g=[ 2    2    2    2    1    1    1    1
    6    6   -2   -2   -1    0    1    0
    6   -2   -2    6    0    1    0   -1
   -2    2   -2   -2   -0   -1    0    1
    2    2    2   -2   -1    0    1    0
```

```
      1     1     0     1     0     0     0     0
      0     0     1     0     1     1     1     1
      0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0
      0     0     0     0     1     1     1     1];
```

A.6 DFGg.m

Geometry matrix for the DFG benchmark.

```
function g = DFGg()
g=[2      2      2      2      1      1      1      1
   2.20  2.20  0      0      0.15  0.20  0.25  0.20
   2.20  0      0      2.20  0.20  0.25  0.20  0.15
   0      0.41  0      0      0.20  0.15  0.20  0.25
   0.41  0.41  0.41  0      0.15  0.20  0.25  0.20
   1      1      0      1      0      0      0      0
   0      0      1      0      1      1      1      1
   0      0      0      0      0.20  0.20  0.20  0.20
   0      0      0      0      0.20  0.20  0.20  0.20
   0      0      0      0      0.05  0.05  0.05  0.05];
```