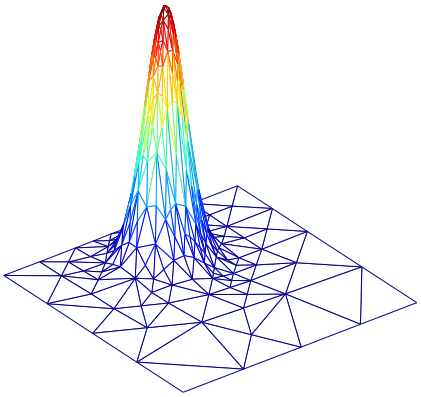
The Spectral/*hp*-Finite Element Method for Partial Differential Equations



A. P. Engsig-Karup

Scientific Computing Section, DTU Compute
Technical University of Denmark, Bldg. 321
DK-2800 Kgs.-Lyngby, Denmark

December 25, 2025

Contents

List of Algorithms

Preface

This set of lecture notes provides an elementary introduction to both the classical Finite Element Method (FEM) and the extended Spectral/*hp*-Finite Element Method for solving Partial Differential Equations (PDEs). Many problems in science and engineering can be formulated mathematically and involves one or more PDEs. The FEM is nowadays an important numerical discretization technique for approximately solving such mathematical equations on a computer.

The set of lecture notes has been written for engineering students for use in the short three-week course *02623 The Finite Element Method for Partial Differential Equations* given at the Technical University of Denmark. The basic aim of the current lecture notes follows that of the earlier lecture notes for the course [9], which is to describe the FEM in a way that supports the reader in implementing the method independently. The original set of course notes has been modified and updated and additional chapters describing the high-order extensions to form the Spectral/*hp*-Finite Element Method have been included. Thus the significant contributions of Chapters 1, 2 and 5 covering the classical Finite Element Method are in large parts due to V. A. Barker and J. Reffstrup.

With this set of lecture notes it should be possible for the reader to make a Spectral/*hp*-FEM toolbox in successive steps with the support given in the text. Emphasis is on the practical details supported with basic and sufficient theory to build the foundation in a three weeks period where the tools are developed and applied immediately. Furthermore, the aim of guiding the reader to develop a Spectral/*hp*-FEM toolbox is to provide a simple and generic framework for developing small prototype applications rather than directly approaching large-scale models. With this in mind, the goal is to let the reader encounter the typical problems involved in the practical implementation of these models and thereby gain a fundamental understanding of the algorithms and their practical implementation.

For the practical work, a number of templates described using pseudo programming code, should be understood and converted by the reader to a programming language in a concrete implementation. A number of exercises is given which in a step-by-step manner guides the reader toward developing the necessary subroutines which can be used to solve typical and fairly general PDEs in one or two spatial dimensions. In the course the chosen programming environment is Matlab, however, this is by no means a necessary requirement.

The mathematical level needed to grasp the details of this set of notes requires an elementary background in mathematical analysis and linear algebra. Each chapter is supplemented with hands-on exercises and the amount of material covered is intended to be balanced in such a way that each subject amounts to approximately one week of work including producing a small report to document and communicate the work effort.

This set of lecture notes is currently a working draft and may contain some (hopefully) minor errors. Any suggestions for improving the notes or feedback on errors and the content and its structure will be highly appreciated. Please report to the email apek@dtu.dk.

Allan P. Engsig-Karup, December 25, 2025.

Introduction

Why should we be interested in the Finite Element Method (FEM)? The FEM is a general numerical method for solving both ordinary and partial differential equations in science and engineering, e.g. structural mechanics, fluid dynamics, electromagnetic, elasticity, etc. This set of lecture notes seek to build the basic foundation for understanding how to setup a discrete set of coupled equations to solve such differential equations numerically. Focus will be on a few prototype differential equations, e.g. the poisson problem which appear in steady-state thermodynamics for heat problems, which is useful to demonstrate the practical discretization and solution procedure. Generally speaking, the FEM is applied to problems where the solution is not easily obtained or impossible to obtain by analytical means. The name of the method has its origin in the fundamental idea to split up the solution domain into a number of smaller sub-domains or elements. The main advantages of the FEM are the ability to solve both linear and nonlinear problems and handle solution domains with arbitrarily complex geometry. Furthermore, the FEM framework is nowadays very extensively covered and applied in science and engineering and has a very mature theoretical foundation securing the methodological framework for developing robust numerical codes.

The first use of what we today call finite element methods appeared in the 1940s and the first commercial software package appeared 1964. R. W. Clough was the first to coin the term "finite elements" in 1960. In 1981 I. Babuška and co-workers for the first time introduced the ideas of p-order Finite Element Methods in theoretical papers [7, 8]. In 1984 A. T. Patera [43] coined what is referred to as "Spectral element methods" where the advantage of Spectral Methods [11, 13, 14, 29, 33] is combined with the ability of the FEM to handle complex geometries. The first book covering both p- and hp-versions of the FEM was published in 1991 by B. A. Szabo and I. Babuška [54]. The interested reader may consult more recent text books such as [18, 24, 36, 45, 58] to learn more about the theory and implementation details of high-order finite element methods.

The FEM is based on a variational method of approximation of the PDE. A global domain is divided into sub-domains which we will refer to as elements in what follows. By this approach the FEM becomes very well-suited to problems with complex geometries since we only need a way of representing the solution on each element which in principle can be of any shape. On the set of elements, a set of local approximations is constructed which can be patched together to form a global solution to the problem. Typically, the local approximations is based on polynomials and the order of these polynomials directly affects the formal accuracy of the method. The use of polynomials is a typical choice in FEM because they are straightforward to differentiate and integrate which is convenient for computer implementations.

In the classical FEM linear polynomials are employed and usually referred to as the h-version of the FEM because convergence can only be achieved by increasing the number of elements since the polynomial order is fixed on each element. In the p-version the number of elements is kept fixed and instead convergence is achieved through increasing the polynomial order of the elements. Combining the h- and the p-version leads to the so-called hp-FEM version where convergence can be achieved by combining the two strategies. The latter type of FEM discretization is also referred to as Spectral/hp Element Methods when the element interpolation nodes are placed at the zeros of an appropriate family of orthogonal polynomials, since it is possible to employ high order polynomial basis functions and thereby fast exponential or spectral convergence in a multi-domain

setting can be achieved for smooth problems.

The FEM methods presented in this work, can be characterized as nodal methods where the solution can be interpreted directly according to the physics of the problem and the unknowns of the discrete problem is associated with a discrete representation of the domain referred to as a mesh. This is in contrast to the modal Galerkin methods where the unknowns are the modal coefficients to the local basis functions which have no direct physical interpretation.

Chapter 1

FEM in one space dimension

Assume we have a mathematical model which describes some physical systems in terms of a differential equation. To create a numerical model we need to make two fundamental choices

- How to satisfy the differential equation of interest?
- How to represent the solution?

By these two choices the outcome for any numerical method is a discrete finite representation of the differential equations, typically in the form of a system of coupled equations. The number of equations then corresponds to the number of unknowns in the discrete representation.

In the Finite Element Method (FEM) we divide the spatial domain into a number of so-called elements (or subdomains) which are non-overlapping. Thus, for the representation of the solution we are also faced with the task of

- How to generate a mesh?

On each of the elements which make up a finite element mesh, we seek to represent the solution using some polynomial expansion of arbitrary order within each element. Since the order is arbitrary and we prefer a computational environment where the setup is made generic, we need to determine

- How to compute polynomial expansions?
- How to evaluate integrals and derivatives?

in order to be able to develop formulation based on an element subdivision of the domain of interest.

To satisfy a differential equation using FEM, it is typical to use a Galerkin method. The Galerkin method can be considered a special case of the Method of Weighted Residuals (MWR) coined by Crandall [16] and surveys of the method can for example be found in [13, 22]. In short, the MWR is a fundamental technique to determine how to satisfy (or minimize the residual or truncation errors of) problems defined in terms of a set of governing differential equations when we seek to determine an approximate solution hereof.

With these choices and questions in mind we embark on the road toward understanding the basics of FEM together with focus on implementation details. We will start out by focussing on one-dimensional problems.

We shall see that the standard FEM discretization of model problems consists of a series of steps

1. Represent domain of interest with a Finite Element Mesh (mesh generation problem).

2. Choose appropriate basis functions to represent functions on the mesh.

3. Establish a weak formulation to be used as mathematical basis for constructing Finite Element approximations to the solution of boundary value or initial value problems.

4. For each element in mesh, determine local elemental contributions to global integrals in the weak formulation.

5. From local elemental contributions a procedure of global assembly makes it possible to compute systems of algebraic equations.

6. Modify the system of algebraic equation to incorporate and impose appropriate boundary conditions in correspondence with the model problem.

7. Compute Finite Element Solution by solving the system of algebraic equations.

8. Post process computed results.

1.1 Finite Element Mesh

Consider an interval $0 \leq x \leq L$ together with a set of points

$$0 = x_1 < x_2 < \cdots < x_M = L$$

For each of the M points let e_i denote the subinterval $x_i \leq x \leq x_{i+1}$. We call the points $x_1, x_2, ..., x_M$ nodes and the subintervals $e_1, e_2, ..., e_{M-1}$ elements. The nodes and elements make up a finite element mesh. We denote the length of each element e_i by h_i and determine it as

$$h_i = x_{i+1} - x_i, \quad i = 1, 2, ..., M-1 \tag{1.1}$$

making it possible to create elements with uniform as well as nonuniform lengths. The use of nonuniform sizes for the elements can be used adaptively to improve the overall accuracy and efficiency of computations. To completely describe the finite element mesh we can introduce two

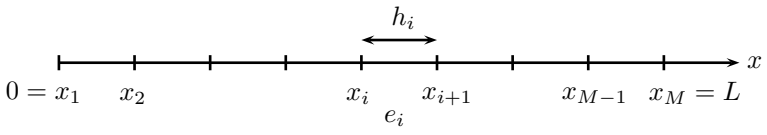


Figure 1.1: Sketch and notation for a one-dimensional mesh.

mesh data tables which can be generated by any suitable mesh generator. A data table `VX` stores the x-coordinates of the set of unique global vertex nodes, and an element connectivity table `EToV` stores for each element the numbers of the vertex nodes that are used to form the element.

The mesh element data tables for the set of points on the given interval can be setup as illustrated in table 1.1. The use of mesh data tables makes it possible to treat the elements of the mesh in a generic way and is a cornerstone in the practical implementation of any general FEM solver where the mesh generation procedure is considered independently of the discretization problem and setup. The strength of this approach is that once the numerical solver has been developed, different problems can subsequently be solved with minimal effort, since for every new problem the solver can be reused. Thus, with this type of setup the main user effort will be the pre- and post-processing steps where user input needs to be supplied in the initial stage of the mesh generation procedure and changing the setup parameters for the new problem.

VX	
i	x_i
1	x_1
2	x_2
⋮	⋮
m	x_m
⋮	⋮
M	x_M

EToV		
n	1	2
1	1	2
2	2	3
⋮	⋮	⋮
m	m	m+1
⋮	⋮	⋮
M-1	M-1	M

Table 1.1: Element tables for a generic mesh in one dimension. Left: `VX`. Right: `EToV`. The first column with indices i o n does not need to be stored in the tables.

1.2 Finite Element Basis Functions

We seek to represent functions defined in the discrete topology defined by the mesh in a representation based on piecewise polynomial functions of the form

$$\hat{u}(x) = \sum_{i=1}^{M} \hat{u}_i N_i(x) \tag{1.2}$$

such that $\hat{u}(x) \approx u(x)$, where M is the number of global nodes, $\hat{u}_i \cong u(x_i)$ is either a set of interpolated function values of the solution or approximate solution. The set of functions $N_i(x)$, $i = 1, ..., M$ is the set of global finite element basis functions. The global finite element basis functions are defined such that

$$N_i(x_i) = 1, \quad N_i(x_j) = 0, \quad j \neq i \tag{1.3}$$

and is therefore said to have the Cardinal property, which can also be expressed as

$$N_i(x_j) = \delta_{i,j} = \begin{cases} 0 & , i \neq j \\ 1 & , i = j \end{cases} \tag{1.4}$$

where $\delta_{i,j}$ is the Kronecker's delta. This property implies that the i'th global finite element basis function takes the value of unity at the i'th global node and zero at all other nodes. This implies that the coefficients u_i in (1.2) corresponds to the discrete function values at specific points of the domain. The word "global" stresses that each function $N_i(x)$ is defined on the entire interval $x \in [0, L]$ even though each function can be made to have local support only, e.g. by construction such that it is nonzero on at most two adjacent elements.

Each of the global finite element basis functions can be represented in terms of the local basis functions which are defined on each of the elements. If the local basis functions are linear functions, i.e. polynomials of order one, then each of the global basis functions $N_i(x)$ is continuous on the solution domain $x \in [0, L]$ and linear on each element. For a one-dimensional mesh with $M = 3$ nodes, this is illustrated in Figure 1.2 for a nonuniform mesh consisting of two elements. For general meshes in one spatial dimension a typical global basis function is resembling a "hat" as illustrated in Figure 1.3.

In the following, we start out by defining the global basis functions analytically under the assumption that they are piecewise linear as in the classical finite element method. In chapter 3, we will generalize this approach to enable the use of higher order polynomials in the element basis.

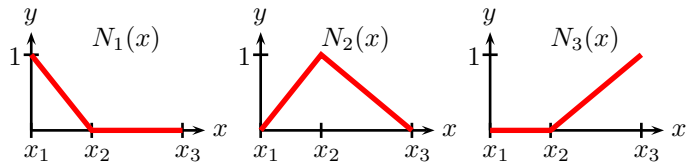


Figure 1.2: Illustration of the three global basis functions on a nonuniform mesh consisting of two elements where the local polynomial order is one for the basis functions within each element.

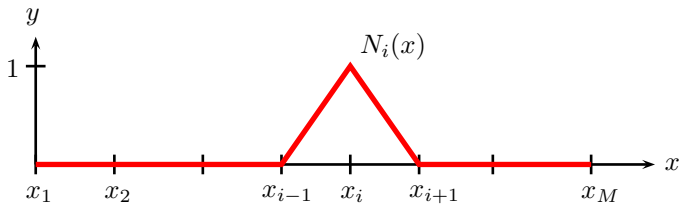


Figure 1.3: A typical global finite element basis function.

The global finite element basis functions are defined analytically as

$$
\begin{aligned}
N_1(x) &= \begin{cases} 1 - \frac{x-x_1}{h_1} & , x_1 \le x \le x_2 \\ 0 & , \text{otherwise} \end{cases} \\
N_i(x) &= \begin{cases} \frac{x-x_{i-1}}{h_{i-1}} & , x_{i-1} \le x \le x_i \\ 1 - \frac{x-x_i}{h_i} & , x_i \le x \le x_{i+1} \\ 0 & , \text{otherwise} \end{cases} \qquad , i = 2, 3, ..., M-1 \\
N_M(x) &= \begin{cases} \frac{x-x_{M-1}}{h_{M-1}} & , x_{M-1} \le x \le x_M \\ 0 & , \text{otherwise} \end{cases}
\end{aligned}
\tag{1.5}
$$

Since $N_i(x)$ is nonzero only on elements e_{i-1} and e_i, it follows that on each element e_i the only two nonzero functions are $N_i(x)$ and $N_{i+1}(x)$ as illustrated in Figure 1.4.

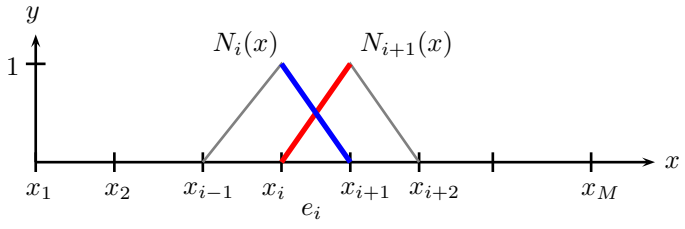


Figure 1.4: Global finite element basis functions which are nonzero over the i'th element.

This leads us to define two local finite element basis functions for element e_i as

$$\begin{aligned} N_1^{(i)}(x) &= N_i(x) = 1 - \frac{x - x_i}{h_i}, \quad x_i \leq x \leq x_{i+1} \\ N_2^{(i)}(x) &= N_{i+1}(x) = \frac{x - x_i}{h_i}, \quad x_i \leq x \leq x_{i+1} \end{aligned} \tag{1.6}$$

The word "local" stresses that each function is defined only on a single element. We will see in section 1.6 that for the implementation of the finite element method for solving a differential equation we can exploit that calculations on the interval $x \in [0, L]$ can be reduced to calculations on individual elements. In this process the local basis functions are of central importance. The local basis functions of an element is illustrated in Figure 1.5.

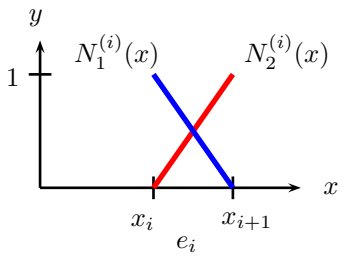


Figure 1.5: The local nodal finite element basis functions when the local polynomial order is one.

1.3 Interpolation

Finite element basis functions are easily used for linear interpolation. Let there be given a function $u(x)$ defined on the interval $0 \leq x \leq L$ and construct an interpolating function using the global finite element functions as

$$u_I(x) = \sum_{i=1}^{M} u(x_i) N_i(x), \quad 0 \leq x \leq L \tag{1.7}$$

where $N_i(x)$, $i = 1, ..., M$ are the global finite element basis functions for a given mesh. Since each $N_i(x)$ is continuous and piecewise linear, so is the interpolating function $u_I(x)$. Further, it follows from the Cardinal property of the basis functions that

$$u_I(x_i) = u(x_i), \quad i = 1, 2, .., M \tag{1.8}$$

Thus the interpolating function $u_I(x)$ is the continuous, piecewise linear function that interpolates $u(x)$ at the nodes. Thus, the interpolating function can be used to represent a discrete representation of a function. The functions $u(x)$ and $u_I(x)$ are illustrated in Figure 1.6.

Regarding the interpolation error $\epsilon(x) = u_I(x) - u(x)$, it can be shown (e.g. see [20]) that if $u(x)$ is twice differentiable then for $i = 1, 2, ..., M - 1$ the error can be bounded from above as

$$\max_{x \in e_i} |\epsilon(x)| \leq \frac{1}{8} h_i^2 \max_{x \in e_i} |u''(x)| \tag{1.9}$$

where u'' corresponds to the second derivative with respect to the dependent variable, i.e. $u'' = \frac{d^2 u}{dx^2}$. These are "local" error bounds. They lead directly to the "global" bound

$$\max_{0 \leq x \leq L} |\epsilon(x)| \leq \frac{1}{8} h^2 \max_{0 \leq x \leq L} |u''(x)| \tag{1.10}$$

where the largest element length h is determined as

$$h = \max_{i=1,2,...,M-1} h_i \tag{1.11}$$

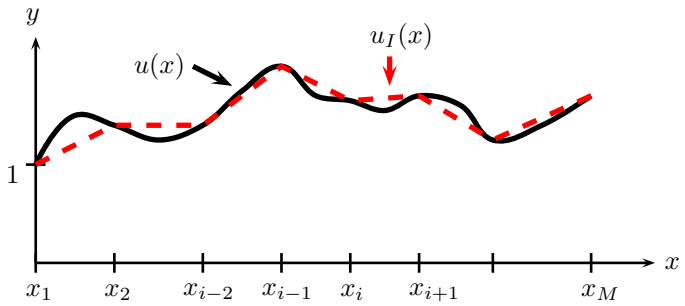


Figure 1.6: A global interpolating function $u_I(x)$ to a function $u(x)$.

Sometimes there is given a set of values $w_1, w_2, ..., w_M$, and it is of interest to construct a function $w(x)$, e.g. for providing estimates of the function values in between the nodal values, $x \in [0, L]$ such that it interpolates the values exactly

$$w(x_i) = w_i, \quad i = 1, 2, ..., M. \tag{1.12}$$

This can be achieved by taking $w(x)$ as the continuous, piecewise-linear function

$$w(x) = \sum_{i=1}^{M} w_i N_i(x), \quad 0 \leq x \leq L. \tag{1.13}$$

1.4 The Finite Element Method

To formulate a finite element method for solving a differential equation we need to choose how to satisfy the PDE. This choice in conjunction with the choice of representing the approximate solution to the PDE using global basis functions defines the numerical method.

It is typical to use a Galerkin formulation in the Finite Element Method, and we will illustrate the method for solving (approximately) a model boundary value problem for a second-order differential equation in one space variable.

The problem is the following: Find a function $u(x)$, $0 \leq x \leq L$ that satisfies the differential equation

$$u'' - u = 0, \quad 0 \leq x \leq L \tag{1.14}$$

together with the boundary conditions

$$u(0) = c, \quad u(L) = d \tag{1.15}$$

The exact solution of this problem can be shown to be of the form

$$u(x) = c_1 e^x + c_2 e^{-x}, \quad 0 \leq x \leq L \tag{1.16}$$

where c_1 and c_2 are real-valued constants that can be determined using the boundary conditions.

Since we will be approximating $u(x)$ by a function whose first-order derivative has jump discontinuities at the nodes, it turns out to be necessary to reformulate the problem so as to remove the second-order derivative in (1.14). This is done by multiplying both sides of (1.14) by a function v that satisfies the boundary conditions

$$v(0) = v(L) = 0 \tag{1.17}$$

and integrating over the interval $0 \leq x \leq L$, and then applying integration by parts. Thus

$$\int_0^L (u'' - u)v \, dx = 0$$

or

$$\int_0^L u''v \, dx - \int_0^L uv \, dx = 0$$

or

$$u'(L)v(L) - u'(0)v(0) - \int_0^L u'v' \, dx - \int_0^L uv \, dx = 0 \tag{1.18}$$

or, using (1.17),

$$\int_0^L (u'v' + uv) \, dx = 0 \tag{1.19}$$

This establishes that the solution of problem (1.14), (1.15) is also a solution of the problem of finding a function $u(x)$ that

1. satisfies (1.19) for all sufficiently smooth functions v with property (1.17),
2. satisfies (1.15).

One can also show the converse: If a function $u(x)$ is a solution of this new problem then it is also a solution of (1.14), (1.15). We call this new problem the *weak formulation* of problem (1.14), (1.15). The weak formulation is defined from the differential equations and can during the derivation take into account the boundary conditions in the expression of the surface terms that result from integration by parts.

It should be realized that the requirement "sufficiently smooth" (which in fact is a requirement on $u(x)$ as well as $v(x)$) is a vague one. How smooth, precisely, is "sufficiently smooth"? A very rough answer is that u and v must be smooth enough for the integral in (1.19) to make sense. A rigorous discussion of this matter lies far beyond the scope of these notes. Fortunately, it turns out that continuous, piecewise polynomials, the kind of functions used in the finite element method, are smooth enough.

We will now apply the finite element method to the weak formulation, considering first the simple mesh shown in Figure 1.2. The idea is to approximate $u(x)$ by a function of the form

$$\hat{u}(x) = \hat{u}_1 N_1(x) + \hat{u}_2 N_2(x) + \hat{u}_3 N_3(x) \tag{1.20}$$

where $N_1(x)$, $N_2(x)$ and $N_3(x)$ are the global finite element basis functions for the given mesh and $\hat{u}_1$, $\hat{u}_2$ and $\hat{u}_3$ are coefficients to be determined. Since $\hat{u}(x)$ is to be an approximation of $u(x)$, we impose the boundary conditions (1.15) on $\hat{u}(x)$ and require that

$$\hat{u}(0) = c, \quad \hat{u}(L) = d$$

From (1.20) and (1.4) we see that

$$\hat{u}(0) = \hat{u}_1 N_1(0) + \hat{u}_2 N_2(0) + \hat{u}_3 N_3(0) = \hat{u}_1$$

$$\hat{u}(L) = \hat{u}_1 N_1(L) + \hat{u}_2 N_2(L) + \hat{u}_3 N_3(L) = \hat{u}_3$$

Consequently, $\hat{u}(x)$ satisfies the given boundary conditions when we put

$$\hat{u}_1 = c, \quad \hat{u}_3 = d \tag{1.21}$$

Since $\hat{u}_1$ and $\hat{u}_3$ are now known, it remains only to determine $\hat{u}_2$. We do this by substituting $\hat{u}(x)$ for $u(x)$ in (1.19) and putting $v(x)$ equal to $N_2(x)$. Note that $N_2(x)$ vanishes at the endpoints of the interval, as required by (1.17). Thus we have

$$\int_0^L (\hat{u}' N_2' + \hat{u} N_2)\, dx = 0$$

or, using (1.20) and collecting terms,

$$\hat{u}_1 \int_0^L (N_1' N_2' + N_1 N_2)\, dx + \hat{u}_2 \int_0^L (N_2' N_2' + N_2 N_2)\, dx + \hat{u}_3 \int_0^L (N_3' N_2' + N_3 N_2)\, dx = 0$$

or, switching the order of the factors in the terms of the integrands,

$$\hat{u}_1 \int_0^L (N_2' N_1' + N_2 N_1)\, dx + \hat{u}_2 \int_0^L (N_2' N_2' + N_2 N_2)\, dx + \hat{u}_3 \int_0^L (N_2' N_3' + N_2 N_3)\, dx = 0$$

Finally, we write this in the form

$$a_{2,1}\hat{u}_1 + a_{2,2}\hat{u}_2 + a_{2,3}\hat{u}_3 = 0 \tag{1.22}$$

where

$$a_{2,1} = \int_0^L (N_2' N_1' + N_2 N_1)\, dx\,, \quad a_{2,2} = \int_0^L (N_2' N_2' + N_2 N_2)\, dx$$

$$a_{2,3} = \int_0^L (N_2' N_3' + N_2 N_3)\, dx$$

After computing these coefficients and using (1.21), we can solve (1.22) for $\hat{u}_2$.

We consider now the case of a general mesh on $x \in [0, L]$, as illustrated in Figure 1.1. Here $\hat{u}(x)$ has the form

$$\hat{u}(x) = \sum_{j=1}^{M} \hat{u}_j N_j(x) \tag{1.23}$$

As before, we impose the conditions

$$\hat{u}_1 = c, \quad \hat{u}_M = d \tag{1.24}$$

to make $\hat{u}(0) = c$ and $\hat{u}(L) = d$. To find the remaining $(M-2)$ coefficients, we substitute (1.23) for $u(x)$ in (1.19) and put $v(x)$ equal to, successively, $N_2(x)$, $N_3(x)$,...,$N_{M-1}(x)$. Note that each of these functions satisfies requirement (1.17). The result of this procedure, after simplification, may be expressed as

$$\sum_{j=1}^{M} a_{i,j}\hat{u}_j = 0\,, \qquad i = 2, 3, ..., M-1$$

where

$$a_{i,j} = \int_0^L (N_i' N_j' + N_i N_j)\, dx$$

Since $N_i(x)$ and $N_j(x)$ 'overlap' only for $j = i-1, i, i+1$, this system of equations reduces to

$$a_{i,i-1}\hat{u}_{i-1} + a_{i,i}\hat{u}_i + a_{i,i+1}\hat{u}_{i+1} = 0\,, \qquad i = 2, 3, ..., M-1 \tag{1.25}$$

where

$$a_{i,i-1} = \int_{x_{i-1}}^{x_i} (N_i' N_{i-1}' + N_i N_{i-1})\, dx$$

$$a_{i,i} = \int_{x_{i-1}}^{x_i} ((N_i')^2 + N_i^2)\, dx + \int_{x_i}^{x_{i+1}} ((N_i')^2 + N_i^2)\, dx$$

$$a_{i,i+1} = \int_{x_i}^{x_{i+1}} (N_i' N_{i+1}' + N_i N_{i+1})\, dx$$

This is a system of $(M-2)$ linear algebraic equations in the $(M-2)$ unknowns $\hat{u}_2$, $\hat{u}_3$,...,$\hat{u}_{M-1}$. It can be noticed that we obtain (1.22) when $M = 3$.

1.5 Element Matrices

We have expressed $a_{i,i-1}$, $a_{i,i}$ and $a_{i,i+1}$ above in terms of integrals over individual elements. When these integrals are rewritten in terms of the local finite element basis functions (1.6) illustrated in Figure 1.5, the result is

$$
\begin{aligned}
a_{i,i-1} &= \int_{x_{i-1}}^{x_i} [(N_2^{(i-1)})'(N_1^{(i-1)})' + N_2^{(i-1)} N_1^{(i-1)}]\, dx \\
a_{i,i} &= \int_{x_{i-1}}^{x_i} [(N_2^{(i-1)})'^{\,2} + (N_2^{(i-1)})^2]\, dx + \int_{x_i}^{x_{i+1}} [(N_1^{(i)})'^{\,2} + (N_1^{(i)})^2] dx \\
a_{i,i+1} &= \int_{x_i}^{x_{i+1}} [(N_1^{(i)})'(N_2^{(i)})' + N_1^{(i)} N_2^{(i)}] dx
\end{aligned}
$$

It is convenient to associate with each element e_i an *element matrix* defined by

$$
\mathcal{K}^{(i)} = \begin{bmatrix} k_{1,1}^{(i)} & k_{1,2}^{(i)} \\ \\ k_{2,1}^{(i)} & k_{2,2}^{(i)} \end{bmatrix}
$$

where

$$
k_{r,s}^{(i)} = \int_{x_i}^{x_{i+1}} [(N_r^{(i)})'(N_s^{(i)})' + N_r^{(i)} N_s^{(i)}]\, dx\,, \qquad r, s = 1, 2 \tag{1.26}
$$

It is possible to derive from the analytical expressions for $N_r^{(i)}$ and $N_s^{(i)}$ given in (1.6) the result

$$
\mathcal{K}^{(\mathbf{i})} = \begin{bmatrix} (\frac{1}{h_i} + \frac{h_i}{3}) & (-\frac{1}{h_i} + \frac{h_i}{6}) \\ \\ (-\frac{1}{h_i} + \frac{h_i}{6}) & (\frac{1}{h_i} + \frac{h_i}{3}) \end{bmatrix} \tag{1.27}
$$

Observing that

$$
a_{i,i-1} = k_{2,1}^{(i-1)}, \quad a_{i,i} = k_{2,2}^{(i-1)} + k_{1,1}^{(i)}, \quad a_{i,i+1} = k_{1,2}^{(i)} \tag{1.28}
$$

we can substitute these relations in (1.25). When $M = 3$ we obtain (1.22) in the form

$$
k_{2,1}^{(1)} \hat{u}_1 + (k_{2,2}^{(1)} + k_{1,1}^{(2)}) \hat{u}_2 + k_{1,2}^{(2)} \hat{u}_3 = 0 \tag{1.29}
$$

In the general case of (1.25) it is convenient to express the system in matrix-vector form with one equation for each unknown coefficient, as illustrated for $M = 6$ by

$$
\begin{bmatrix}
k_{2,1}^{(1)} & (k_{2,2}^{(1)} + k_{1,1}^{(2)}) & k_{1,2}^{(2)} & & & \\
k_{2,1}^{(2)} & (k_{2,2}^{(2)} + k_{1,1}^{(3)}) & k_{1,2}^{(3)} & & & \\
 & k_{2,1}^{(3)} & (k_{2,2}^{(3)} + k_{1,1}^{(4)}) & k_{1,2}^{(4)} & & \\
 & & k_{2,1}^{(4)} & (k_{2,2}^{(4)} + k_{1,1}^{(5)}) & k_{1,2}^{(5)} &
\end{bmatrix}
\begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \\ \hat{u}_4 \\ \hat{u}_5 \\ \hat{u}_6 \end{bmatrix}
= \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}
$$

Since $\hat{u}_1 = c$ and $\hat{u}_6 = d$, the terms $k_{2,1}^{(1)} \hat{u}_1$ and $k_{1,2}^{(5)} \hat{u}_6$ in the first and last rows, respectively, of this system are known and can be moved to the right-hand side. This yields

$$
\begin{bmatrix}
(k_{2,2}^{(1)} + k_{1,1}^{(2)}) & k_{1,2}^{(2)} & & \\
k_{2,1}^{(2)} & (k_{2,2}^{(2)} + k_{1,1}^{(3)}) & k_{1,2}^{(3)} & \\
 & k_{2,1}^{(3)} & (k_{2,2}^{(3)} + k_{1,1}^{(4)}) & k_{1,2}^{(4)} \\
 & & k_{2,1}^{(4)} & (k_{2,2}^{(4)} + k_{1,1}^{(5)})
\end{bmatrix}
\begin{bmatrix} \hat{u}_2 \\ \hat{u}_3 \\ \hat{u}_4 \\ \hat{u}_5 \end{bmatrix}
= \begin{bmatrix} -k_{2,1}^{(1)} c \\ 0 \\ 0 \\ -k_{1,2}^{(5)} d \end{bmatrix}
$$

Note the coefficient matrix is now square. By a trivial extension of this system we can include $\hat{u}_1$ and $\hat{u}_6$ in the vector of unknowns as follows:

$$\begin{bmatrix} 1 & & & & & \\ & (k_{2,2}^{(1)} + k_{1,1}^{(2)}) & k_{1,2}^{(2)} & & & \\ & k_{2,1}^{(2)} & (k_{2,2}^{(2)} + k_{1,1}^{(3)}) & k_{1,2}^{(3)} & & \\ & & k_{2,1}^{(3)} & (k_{2,2}^{(3)} + k_{1,1}^{(4)}) & k_{1,2}^{(4)} & \\ & & & k_{2,1}^{(4)} & (k_{2,2}^{(4)} + k_{1,1}^{(5)}) & \\ & & & & & 1 \end{bmatrix} \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \\ \hat{u}_4 \\ \hat{u}_5 \\ \hat{u}_6 \end{bmatrix} = \begin{bmatrix} c \\ -k_{2,1}^{(1)}\, c \\ 0 \\ 0 \\ -k_{1,2}^{(5)}\, d \\ d \end{bmatrix}$$

We henceforth denote this M'th-order system (for any $M \geq 3$) by

$$\mathbf{A}\hat{\mathbf{u}} = \mathbf{b} \tag{1.30}$$

The coefficient matrix $\mathbf{A}$ is seen to be tridiagonal and symmetric (since by (1.28) $k_{i,j}^{(n)} = k_{j,i}^{(n)}$) for the model problem in one space dimension we have considered. One can show it is also positive definite; i.e., the eigenvalues of $\mathbf{A}$ are positive. Knowledge about such properties makes it possible to employ efficient solvers for solving the system.

1.6 The Computation

The computational tasks are those of computing $\mathbf{A}$ and $\mathbf{b}$ and of solving (1.30). Algorithms 1 and 2 below deal with the computation of $\mathbf{A}$ and $\mathbf{b}$. The process of constructing the system coefficient matrix and right hand side vector is known as "global assembly". Since $\mathbf{A}$ is symmetric, it suffices to compute the entries on the main diagonal ($a_{i,i}$, $i = 1, 2, ..., M$) and those on the upper bidiagonal ($a_{i,i+1}$, $i = 1, 2, ..., M-1$). The algorithms are written in a pseudo-programming language which makes use of an $M \times M$ array, $\mathbf{A}$, and an $M \times 1$ array, $\mathbf{b}$, for the storage of $\mathbf{A}$ and $\mathbf{b}$, respectively. The choice of an $M \times M$ array for $\mathbf{A}$ is based on considerations of clarity alone; it makes it easier to show what needs to be computed. In a practical implementation a sparse data structure should always be used for $\mathbf{A}$.

Algorithm 1: Global assembly of upper triangular part of coefficient matrix $\mathbf{A}$ (1D).

Allocate storage for $\mathbf{A}$ and $\mathbf{b}$
for $i := 1$ **to** $M - 1$
  Compute $k_{1,1}^{(i)}$, $k_{1,2}^{(i)}$ and $k_{2,2}^{(i)}$ from (1.27).
  $a[i,i] := a[i,i] + k_{1,1}^{(i)}$
  $a[i,i+1] := k_{1,2}^{(i)}$
  $a[i+1,i+1] := k_{2,2}^{(i)}$

This computation has ignored the boundary conditions and the system for the case $M = 3$ takes the preliminary form

$$\begin{bmatrix} k_{1,1}^{(1)} & k_{1,2}^{(1)} & \\ k_{2,1}^{(1)} & (k_{2,2}^{(1)} + k_{1,1}^{(2)}) & k_{1,2}^{(2)} \\ & k_{2,1}^{(2)} & k_{2,2}^{(2)} \end{bmatrix} \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{1.31}$$

Comparing the result of Algorithm 1 with $\mathbf{A}$ and $\mathbf{b}$ as illustrated immediately before (1.30) for a case with $M = 6$, one sees that the corrections done as a part of Algorithm 2 are needed.

Algorithm 2: Imposing boundary conditions by modification of system (1D).

$b[1] := c, \quad b[2] := b[2] - a[1,2] * c, \quad a[1,1] := 1, \quad a[1,2] := 0$
$b[M] := d, \quad b[M-1] := b[M-1] - a[M-1,M] * d$
$a[M,M] := 1, \quad a[M-1,M] := 0$

Note that Algorithm 2 exploits symmetry in the element matrix in computing b_2:

$$b[2] := b[2] - a[1,2] * c \;=\; b[2] - k_{1,2}^{(1)} * c \;=\; b[2] - k_{2,1}^{(1)} * c$$

Note, too, that if the computation of $b[M-1]$ were changed to

$$b[M-1] := -a[M-1,M] * d$$

then the result would be incorrect in the case $M = 3$, where (1.30) reduces to

$$\begin{bmatrix} 1 & & \\ & (k_{2,2}^{(1)} + k_{1,1}^{(2)}) & \\ & & 1 \end{bmatrix} \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \end{bmatrix} = \begin{bmatrix} c \\ -k_{2,1}^{(1)} c - k_{1,2}^{(2)} d \\ d \end{bmatrix} \tag{1.32}$$

As mentioned above, an $M \times M$ array is an impractical data structure for $\mathbf{A}$ since only $(2M-1)$ of the M^2 entries of this matrix are needed. The recommended procedure is either to store the nonzero elements appearing on the main diagonal and upper bidiagonal of $\mathbf{A}$ in two single-indexed arrays of length M and $M-1$, respectively, or to store these two diagonals as columns in an $M \times 2$ array. However, in Matlab it is possible to make use of the compressed row storage format which is supported in Matlab by just allocating the necessary sparse storage using the `spalloc` command. Note that to use the sparse capabilities of Matlab one has to be careful not to do significant manipulations of the matrix structure and nonzero elements, since this can result in significant overhead due to the involved data movements in the memory space.

We consider now the problem of solving (1.30), given $\mathbf{A}$ and $\mathbf{b}$. Since $\mathbf{A}$ is symmetric positive definite and tridiagonal, the best numerical algorithms are Gaussian elimination without pivoting and the Cholesky method, e.g. see [25]. The computational work in both cases is $\mathcal{O}(M)$, but it should be noted that the Cholesky method requires about half the storage and half of the computational cost of the LU factorization resulting from Gaussian elimination.

Finally, rather than constructing and solving the M'th-order system (1.30), one could easily work directly with a system of reduced order $(M-2)$ where the known coefficients are eliminated from the system of equations. For example, compare the systems of order four and six immediately before (1.30). Our main reason for not doing this is to prepare the reader for solving problems in two space dimensions, where the larger system without any elimination of the equations for the coefficients is much the simpler to deal with.

1.7 Error Bounds

Let $u(x)$ be the solution of problem (1.14), (1.15), and let $\hat{u}(x)$ be given by

$$\hat{u}(x) = \sum_{i=1}^{M} \hat{u}_i N_i(x)$$

where the coefficients $\hat{u}_1$, $\hat{u}_1$,...,$\hat{u}_M$ are determined by (1.30). It can be shown that there exists a constant C such that

$$\max_{0 \le x \le L} |\hat{u}(x) - u(x)| \le Ch^2 \tag{1.33}$$

where h is the largest element length in the mesh, cf. (1.11).

The function $\hat{u}(x)$ is in general different from the interpolating function $u_I(x)$ in (1.7) which interpolates $u(x)$ at the nodes. However, (1.10) and (1.33) show that both functions are second-order approximations to $u(x)$. It should be mentioned that the interpolating property

$$|u_I(x_i) - u(x_i)| = 0, \qquad i = 1, 2, ..., M$$

which is valid for specific nodes, does *not* imply

$$\max_{0 \le x \le L} |u_I(x) - u(x)| \le \max_{0 \le x \le L} |\hat{u}(x) - u(x)|$$

That is, we need to take into account that the largest point-wise errors is not guaranteed to be found on a node point. It might as well be in between nodes.

There are a number of ways of measuring the error function $(\hat{u}(x) - u(x))$. One is to consider the so-called *infinity norm*

$$||\hat{u}(x) - u(x)||_\infty \equiv \max_{0 \le x \le L} |\hat{u}(x) - u(x)|$$

which is just what we have done in (1.33) above. Another is to look at the *energy norm*

$$||\hat{u}(x) - u(x)||_E \equiv \sqrt{\int_0^L [(\hat{u}(x) - u(x))^2 + (\hat{u}'(x) - u'(x))^2]\, dx}$$

which is associated specifically with the weak formulation (1.19) of the differential equation (1.14).

It turns out that the function $\hat{u}(x)$ produced by the finite element method *minimizes* the energy norm of the error. More precisely, if $w(x)$ is a function of the form

$$w(x) = c\, N_1(x) + d\, N_M(x) + \sum_{i=2}^{M-1} c_i N_i(x)$$

where c_2, c_3,..., c_{M-1} are arbitrary, then

$$||\hat{u}(x) - u(x)||_E \le ||w(x) - u(x)||_E$$

Note that this implies

$$||\hat{u}(x) - u(x)||_E \le ||u_I(x) - u(x)||_E$$

For further discussion of errors see for example [53] and [4].

Exercises

Exercise 1.1

The goal of this exercise is to derive analytical expressions for defining a FEM method to solve a boundary value problem and solve using a discrete set of mesh nodes defining both uniform and nonuniform meshes .

Consider the FEM applied to the boundary value problem for a second-order differential equation

$$u'' - u = 0, \quad 0 \leq x \leq L$$

with the following boundary conditions

$$u(0) = c, \quad u(L) = d$$

a) Derive the exact expression for the element matrix K_i for the i'the element using the variable transformation $y = (x - x_i)/h_i$. Assume that linear polynomials are employed on each element.

b) Let $L = 2$, $c = 1$ and $d = e^2$. Let $M = 3$, i.e. the interval is divided into only two elements, e_1 and e_2, of lengths h_1 and h_2, respectively. Solve for the unknown $\hat{u}_i$ in the following cases: i) uniform mesh with $h_1 = h_2$ and ii) nonuniform mesh with $h_1 = 2h_2$.

c) Consider the case with a uniform mesh with mesh sizes $h_1 = h_2$. The solution to the FEM equations determines a continuous, piecewise linear function $\hat{u}(x)$ that approximates the exact solution $u(x)$ to the problem. Plot $\hat{u}(x)$ and $u(x)$. In the same figure, include a plot of the interpolating function $u_I(x)$ that interpolates $u(x)$ at the nodes x_i, $i = 1, 2, ..., M$.

d) Repeat c) for the nonuniform case ii) in b).

e) Explain the difference between $u_I(x)$ and $\hat{u}(x)$. (HINT: How do we represent and define the two functions on the discrete domain? What distinguishes these definitions from each other?)

f) Describe the basic steps that are needed to generate a set of FEM equations to the boundary value problem.

Exercise 1.2

The goal of this exercise is to write your first FEM procedure for solving a boundary value problem in one dimension. This requires that some of the algorithms of Chapter 1 is implemented.

a) Write a Matlab program based on the skeleton code on the next page that solves the boundary value problem given in Exercise 1.1 using the classical FEM where the local solution is represented using linear polynomials. (HINT: for better reuse of code and for easy testing it is recommended to write your own code pieces as Matlab functions whenever it is appropriate.)

Head:
`function [u] = BVP1D(L,c,d,x)`

Test case:
Input: `L=2, c=1, d=exp(2),`
`x = [0.0, 0.2, 0.4, 0.6, 0.7, 0.9, 1.4, 1.5, 1.8, 1.9, 2.0]`

b) Make appropriate changes to the function `BVP1D` so that the number of nodes M can replace the array `x` as input in the argument list. If M is used in the call, then `x` should be computed such that the nodes are equidistant within the script. Note that the uniformity of the elements makes it possible to simplify the assembly process.

Head:
`function [u,x] = BVP1D(L,c,d,M)`

Test case
Input: `L=2, c=1, d=exp(2), M=11.`

c) Validate your program by comparing with solutions of Exercise 1.1.

d) Compare the results computed using the function with (1.33). What is the convergence rate of the solution when the mesh size decreases, i.e. $h \to 0$? (Hint: determine an estimate for p valid in the asymptotic limit $h \to 0$ assuming that the convergence rate is $\mathcal{O}(h^p)$.)

Skeleton of Matlab program code

```
function [u] = BVP1D(L,c,d,x)
% Purpose: Solve second-order boundary value problem using FEM.
% Author(s): <YOUR NAMES HERE>

%% INPUT PARAMETERS
%     L : Domain length
%     c : Left boundary condition
%     d : Right boundary condition
%     x : 1D mesh vector x(1:{M})

%% GLOBAL ASSEMBLY
% Assemble A (the upper triangle only) and b. (Algorithm 1)

<INSERT YOUR CODE HERE>

%% IMPOSE BOUNDARY CONDITIONS
% (Algorithm 2)

<INSERT YOUR CODE HERE>

%% SOLVE SYSTEM
% Solve using the Cholesky factorization of A to solve A*u=b
[U,flag] = chol(A);
if flag == 0
  u = U \ (U' \ b);
else
  disp('A is not positive definite'), return
end

%% OUTPUT
% Visualize solution and output solution

<INSERT YOUR CODE HERE>
```

Exercise 1.3

The goal of this exercise is develop a procedure for adaptively finding an optimal mesh node distribution which both minimizes the global error and has a uniform error distribution.

First, we will consider how to determine the largest error on an element when we assume that the exact solution is $u(x) = e^x$. With this in mind, we derive expressions for determining the maximum error within an element.

Let $v(x)$ be an arbitrary continuous, piecewise linear function with respect to the nodes

$$0 = x_1 < x_2 < x_3 < ... < x_M = 2$$

and let

$$v_i = v(x_i), \quad i = 1, 2, ..., M$$

If we regard $v(x)$ as an approximation to $u(x)$, then the corresponding error function is

$$F(x) = v(x) - u(x), \quad 0 \leq x \leq 2$$

It is convenient to describe the error with a single number, and we therefore define

$$E = \max_{0 \leq x \leq 2} |F(x)|$$

Consider now the computation of E. Since

$$E = \max\{E_1, E_2, ..., E_{M-1}\}$$

where

$$E_i = \max_{x_i \leq x \leq x_{i+1}} |F(x)|, \quad i = 1, 2, ..., M-1$$

we can turn our attention to the problem of computing E_i. Let's assume that the exact solution is $u(x) = e^x$. Because $v(x)$ is linear on element e_i, it is found that

$$F(x) = v(x) - u(x) = A_i x + B_i - e^x, \quad x_i \leq x \leq x_{i+1}$$

where

$$A_i = \frac{v_{i+1} - v_i}{x_{i+1} - x_i}, \quad B_i = v_i - A_i x_i$$

It is well known that a function that is monotonic on a closed finite interval attains its greatest absolute value at one (or both) of the end points. Applying this to the error function $F(x)$, we find the following:

Case 1: $v_{i+1} \leq v_i$**.**

Here $v(x)$ is constant or monotonically decreasing while e^x is monotonically increasing. Hence $F(x)$ is monotonically decreasing, and

$$E_i = \max\{|F(x_i)|, |F(x_{i+1})|\}$$

Case 2: $v_{i+1} > v_i$ **and** $\xi_i \in [x_i, x_{i+1}]$**, where** $\xi_i = \ln(A_i)$**.**

$F(x)$ has a relative extremum at $x = \xi_i$ (i.e., $F'(\xi_i) = 0$) and is monotonic on each of the subintervals $[x_i, \xi_i]$ and $[\xi_i, x_{i+1}]$. Hence

$$E_i = \max\{|F(x_i)|, |F(x_{i+1})|, |F(\xi_i)|\}$$

Case 3: $v_{i+1} > v_i$ **and** $\xi_i \notin [x_i, x_{i+1}]$**, where** $\xi_i = \ln(A_i)$**.**

Here $F(x)$ is monotonic on $[x_i, x_{i+1}]$ (because there is no extremum in this interval), so

$$E_i = \max\{|F(x_i)|, |F(x_{i+1})|\}$$

From the above we obtain a simple algorithm for computing E_i:

Algorithm 3: Compute error on an element (specific to exercise 1.3).

Compute A_i and B_i
$E_i := \max\ \{\ |F(x_i)|, |F(x_{i+1})|\ \}$
if $A_i > 0$
    Compute ξ_i
    if $\xi \in [x_i, x_{i+1}]$
        $E_i := \max\{E_i, |F(\xi_i)|\}$

a) The purpose of this question it to investigate the error function $e(x) = \hat{u} - u(x)$, where $\hat{u}(x)$ is the FEM solution of the boundary values problem from Exercise 1.1 and $u(x)$ is the exact solution. Write a Matlab function based on Algorithm 3 that computes the element errors `Eelm(1:M-1)` and global error `E` from `x(1:M)` and `v(1:M)`.

Head:
```
function [Eelm,E] = errelm(x,v)
```

Test case:
$L = 2, c = 1, d = e^2$ which makes the exact solution $u(x) = e^x$, $0 \le x \le 2$.

b) Let $v(x)$ be the finite element solution computed in Exercise 1.2 b). Compute the local element errors E_i and the global error E using `errelm`.

c) Let $M = 11$. Define a simple algorithm to find the interior node coordinates $x_2, x_3, ..., x_{M-1}$ that minimize E to have a desired accuracy in the interval $E - \min_{1 \le i \le M-1} E_i \le 10^{-6}$. It may be assumed that

i) The optimal set of nodes is characterised by the property $E_1 = E_2 = ... = E_{M-1}$.

ii) $E_i \cong C_i h_i^2$, where C_i is independent of h_i.

(HINT: How can we update the individual node position using the local error estimates for each element given in ii)?).

Exercise 1.4

a) Repeat Exercise 1.3 b) and c) where now $v(x) = u_I(x)$ the finite element interpolant of $u(x)$. Hence $v_i = e^{x_i}$, $i = 1, 2, ..., M$. In part c), the computed results should be compared with the error estimate (1.10).

b) Compare the results of this exercise with those of Exercise 1.3.

Exercise 1.5

The goal of this exercise is go through the fundamental steps of formulating and implementing a Finite Element Method for a linear advection-diffusion equation. We consider the following model boundary value problem for the linear advection-diffusion equation

$$\begin{aligned} -(\epsilon u')' + (\Psi u)' &= f, \qquad x \in]0,1[\\ u(0) = u(1) &= 0. \end{aligned} \tag{1.34}$$

For simplicity we assume that the coefficients $\epsilon > 0$ and Ψ are real-valued constants on $]0,1[$.

a) Assume that u is a twice continuously differentiable function solving (1.34). Verify that u also solves the variational problem

$$a(u,v) = \ell(v), \tag{1.35}$$

for every continuous piecewise-smooth function v, such that $v(0) = v(1) = 0$, where $a(u,v) = \int_0^1 \epsilon u'v' \mathrm{d}x - \int_0^1 \Psi uv' \mathrm{d}x$ and $\ell(v) = \int_0^1 fv \mathrm{d}x$.

b) Follow the procedure outlined in Sections 1.4-1.5 of the course notes and arrive at the system of linear algebraic equations $\mathbf{A}\hat{\mathbf{u}} = \mathbf{b}$, corresponding to the finite element discretization of variational problem (1.35). Write down the expressions for the elemental stiffness matrices and right hand sides. Is the resulting matrix $\mathbf{A}$ symmetric?

c) If the matrix $\mathbf{A}$ is *positive definite*, that is, $\mathbf{v}^{\mathrm{T}}\mathbf{A}\mathbf{v} > 0$ for all non-zero vectors $\mathbf{v}$, then it is necessarily non-singular and therefore the linear algebraic system $\mathbf{A}\hat{\mathbf{u}} = \mathbf{b}$ admits a unique solution for every right-hand side $\mathbf{b}$.

Show that $\mathbf{v}^{\mathrm{T}}\mathbf{A}\mathbf{v} > 0$ for every vector $\mathbf{v}$, such that $\mathbf{v}_1 = \mathbf{v}_M = 0$ and thus the linear algebraic system resulting from (1.35) always admits a solution.

Note, that you have to show *two* things: (i) $\mathbf{v}^{\mathrm{T}}\mathbf{A}\mathbf{v} \geq 0$ for every vector $\mathbf{v}$ satisfying the boundary conditions; and (ii) $\mathbf{v}^{\mathrm{T}}\mathbf{A}\mathbf{v} = 0$ implies $\mathbf{v} = \mathbf{0}$.

The following hints are given: $\mathbf{v}^{\mathrm{T}}\mathbf{A}\mathbf{v} = a(\sum_{i=1}^{M} v_i N_i, \sum_{j=1}^{M} v_j N_j)$; $\int vv' \mathrm{d}x = 1/2 \int (v^2)' \mathrm{d}x$.

d) Let $f = 1$ (and $\Psi \neq 0$); the analytical solution to (1.34) in this case is

$$u(x) = \frac{1}{\Psi}\left(\frac{1 + (\exp(\Psi/\epsilon) - 1)x - \exp(x\Psi/\epsilon)}{\exp(\Psi/\epsilon) - 1}\right)$$

Plot the solution for fixed $\Psi = 1$ and various values of $\epsilon \in \{\, 1, 0.01, 0.0001 \,\}$.

e) Make appropriate modifications to the program you have written for solving Exercise 1.2, so that is solves the boundary value problem (1.35) instead and verify its convergence. Use the same parameters as in step d). Comment on the variation of the quality of the computed FEM solution with ε.

Exercise 1.6

In this exercise the goal is to develop an algorithm that can be used for automatic Adaptive Mesh Refinement (AMR). Such an algorithm consists of two ingredients, namely, a method for refining the mesh and a method for selecting which mesh elements to be refined. The method for refining the mesh will be based on h-type refinement where existing elements are subdivided into two new element of equal size to reduce errors locally. The selection of elements for refinement will be based on assuming that an optimal mesh have errors distributed equally across elements of the mesh.

In the following, we will consider how to automatically adapt a mesh to the function

$$u(x) = e^{-800(x-0.4)^2} + 0.25e^{-40(x-0.8)^2}$$

across the interval $0 \leq x \leq 1$ such that an interpolating piece-wise polynomial (1.7) achieves a user-defined accuracy level for the accurate representation of $u(x)$.

a) A simple way to guide the selection of mesh elements for refinement, is to estimate the error decrease rate that will result if the i'th element is subdivided in two smaller elements of equal size. We will employ a metric which measures the change in approximate solution across elements by refinement. The metric is defined in terms of the L_2 (mean-square) norm as

$$\Delta err_i = ||\Pi_{h/2,i}u - \Pi_{h,i}u||_{L_2(e_i)}, \quad ||f||_{L_2(\Omega)} \equiv \left(\int_\Omega |f|^2 dx\right)^{1/2}$$

$\Pi_{h,i}u$ is an operator which defines the piece-wise interpolation of $u(x)$ onto the i'th element with mesh size h (see equation (1.7)). How can the metric be computed? Detail this. Then, write a Matlab routine which can compute the estimated error decrease rate based on this metric. The estimate should be based on the mesh tables `VX` and `EToV` (learn about such tables in beginning of Chapter 2) which defines a coarse mesh. `err` is a vector with K elements such that `err(i)` is the estimated error for the i'th element in the coarse mesh.

Head:

```
function [err] = compute_error_decrease(fun,VX,EToV);
```

b) Write a Matlab routine which can subdivide a coarse mesh and produce a refined mesh by refining only elements which has been marked for refinement using an element index vector `idxMarked`.

Head:

```
function [EToVfine,xfine] = refine_marked(EToVcoarse,xcoarse,idxMarked)
```

c) Write a Matlab routine which invokes AMR to produce a mesh for representing $u(x)$ using a piece-wise polynomial representation such that $\Delta err_i < 10^{-4}$ starting from a mesh consisting of three equi-sized elements. Different criteria can be used for selecting elements for refinement, e.g.

$$\Delta err_i > \alpha \cdot \texttt{tol}, \quad 0 < \alpha \leq 1$$

or some kind of fraction of the element with the largest errors

$$\Delta err_i > \alpha \max_i \Delta err_i$$

d) Present plots of the computed solution, the element size distribution for the final mesh and for evaluating the performance of the algorithm that are used in c) (e.g. error vs. mesh points).

If time permits, feel free to experiment with other functions for $u(x)$, metrics for error estimation, error indication (e.g. measuring jumps at interfaces) and element selection criteria.

Exercise 1.7

Consider the FEM applied to the following boundary value problem

$$u''(x) - u(x) = f(x), \quad 0 \leq x \leq 1, \quad u(0) = c, \quad u(1) = d$$

The goal of this exercise will be to write a Matlab program which can solve this boundary value problem using the classical FEM combined with an Adaptive Mesh Refinement (AMR) algorithm. The AMR should be based on h-refinement where the mesh elements are subdivided as in Exercise 1.6. This will require two basic ingredients, namely, a method for doing mesh refinements (reuse method from Exercise 1.6) and a way to estimate the errors (*a posteriori* where the computed FEM solution is used to estimate the accuracy) in order to be able to guide the refinements until a desired overall accuracy level has been achieved.

a) Determine $f(x)$, c and d using the Method of Manufactured solution where it is assumed that the exact solution is $u(x) = e^{-800(x-0.4)^2} + 0.25e^{-40(x-0.8)^2}$. Visualize this solution for $0 \leq x \leq 1$ and comment on expectations to an optimal mesh for the problem. It is not allowed to use the knowledge of this exact solution in the AMR procedure in this exercise.

b) Write a Matlab routine which can compute error estimates based on the L_2-norm on a per-element basis. The error estimate can be expressed in terms of computed solution values (`uhc` and `uhf`) obtained on a coarse (`xc`) and a refined mesh (`xf`). It is advantageous to take into account which elements that have been marked for refinement (`idxmarked`) and store information about how elements are then subdivided (`Old2New`) in appropriate arrays. If you see other ways of doing this setup feel free to do so.

Head: (suggestion)
```
function [err] = errorestimate(xc,xf,uhc,uhf,EToVc,EToVf,Old2New)
```

c) Write a Matlab routine which can subdivide a coarse mesh and produce a refined mesh by refining only elements which has been marked for refinement (Same as in Exercise 1.6 b)).

d) Derive the weak formulation of the boundary value problem in the usual way and identify elemental contributions to A and $\mathbf{b}$ in the linear system that results from the discretization. It is allowed to assume that $f(x)$ can be represented in terms of a continuous piecewise polynomial function (see (1.2)). By this approximation the contributions to a right hand side vector $\mathbf{b}$ can be based on identifying the elemental contributions from

$$\int_0^L f(x)v(x)dx \approx \int_0^L \left(\sum_{j=1}^{M} \hat{f}_j N_j(x) \right) N_i(x)dx = \sum_{j=1}^{M} \hat{f}_j \int_0^L N_i(x)N_j(x)dx$$

e) Write a new Matlab routine by modifying `BVP1D` developed in Exercise 1.2. The new routine should solve the boundary value problem stated in this exercise with a nonzero right hand side function $f(x)$ (`func`).

Head:
```
function [u] = BVP1Drhs(L,c,d,x,func)
```

f) Present relevant plots of computed solutions, element size distributions for converged solutions and evaluate the performance of the FEM + AMR algorithm implemented (e.g. figures showing error vs. DOF and CPU time vs. DOF).

If time permits, feel free to experiment with other functions for $u(x)$, metrics for error estimation, error indication and element selection criteria.

Chapter 2

FEM in two space dimensions

The extension of the ideas for FEM in one space dimension into two dimensions is conceptually trivial, however, do require some extra effort for the implementation. The implementation and solution procedures follow the same steps as in one space dimension.

2.1 Finite Element Basis functions

We consider now problems in two space dimensions. Figure 2.1 shows a two-dimensional closed domain $\overline{\Omega}$ with interior Ω and boundary Γ. A finite element mesh, consisting of triangular elements with nodes at the vertices, is imposed on $\overline{\Omega}$. Note that the boundary of the mesh does not coincide

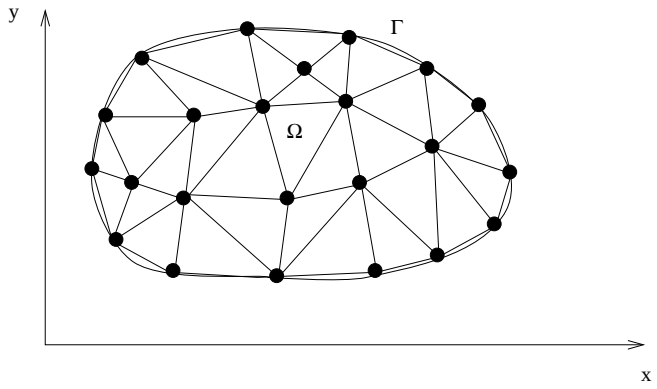


Figure 2.1: A domain with interior Ω and boundary Γ , and a finite element mesh.

with Γ because the latter is curved. As the mesh is refined by increasing the number of element within the domain and on it's boundary, however, the difference between the two boundaries may then be reduced.

We denote the elements of a given mesh by e_n , $n = 1, 2, ..., N$ and the nodes by (x_i, y_i), $i = 1, 2, ..., M$. The value i is the *global node number* of node (x_i, y_i). We also need to assign *local node numbers* to the nodes in any given element. The local node numbers are always 1, 2 and 3, as illustrated in Figure 2.2. (See also Figure 2.5 below).

For each node (x_i, y_i) we introduce a global finite element basis function, $N_i(x, y)$, defined on $\overline{\Omega}$ as follows:

1. $N_i(x, y)$ is continuous on the entire mesh and linear on each element,

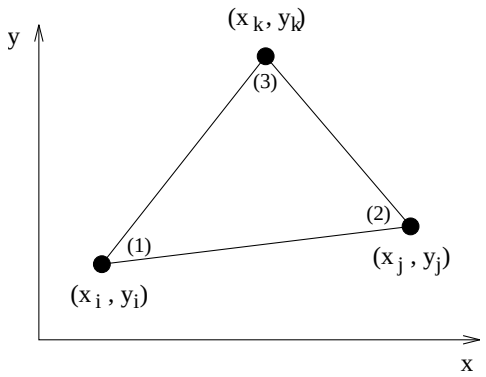


Figure 2.2: Local (1, 2, 3) and global (i, j, k) node numbers in the typical element.

2. $N_i(x_i, y_i) = 1, \quad N_i(x_j, y_j) = 0, \quad j \neq i\,.$

A typical function of this type is shown in Figure 2.3. While it is defined on the whole mesh, it is nonzero only on the elements to which node (x_i, y_i) belongs.

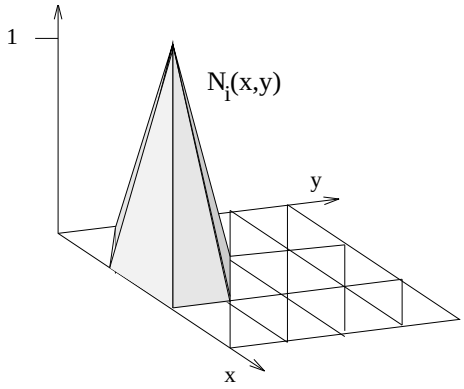


Figure 2.3: A global finite element basis function.

On a given element e_n only three of these functions are nonzero. We call these the local finite element basis functions for e_n and denote them $N_r^{(n)}(x, y)$, $r = 1, 2, 3$. They are defined by

$$N_1^{(n)}(x, y) = N_i(x, y), \quad (x, y) \in e_n$$
$$N_2^{(n)}(x, y) = N_j(x, y), \quad (x, y) \in e_n$$
$$N_3^{(n)}(x, y) = N_k(x, y), \quad (x, y) \in e_n$$

where we are using the correspondence between the local and global node numbers shown in Figure 2.2. An example of these local functions is given in Figure 2.4.

We now derive explicit formulas for the local basis functions. It is convenient for this purpose to use the local ordering of nodes in e_n ; i.e., in this discussion the nodes will be denoted (x_1, y_1), (x_2, y_2) and (x_3, y_3).

Consider first $N_1^{(n)}(x, y)$. Since this function is linear, it is of the form

$$N_1^{(n)}(x, y) = a_1 + b_1 x + c_1 y, \quad (x, y) \in e_n$$

Using the fact that the local basis functions can be constructed from Lagrange polynomials on the n'th element

$$N_1^{(n)}(x_1, y_1) = 1, \quad N_1^{(n)}(x_2, y_2) = 0, \quad N_1^{(n)}(x_3, y_3) = 0$$

we obtain the relations

$$a_1 + b_1 x_1 + c_1 y_1 = 1, \quad a_1 + b_1 x_2 + c_1 y_2 = 0, \quad a_1 + b_1 x_3 + c_1 y_3 = 0$$

This is a system of three linear equations in the three unknowns a_1, b_1 and c_1. Provided that the triangle e_n is not degenerate (i.e., provided the three nodes do not lie on a straight line), it is easy to show that this system has the unique solution

$$a_1 = \frac{1}{2\Delta}\tilde{a}_1, \quad b_1 = \frac{1}{2\Delta}\tilde{b}_1, \quad c_1 = \frac{1}{2\Delta}\tilde{c}_1$$

where

$$\tilde{a}_1 = x_2 y_3 - x_3 y_2, \quad \tilde{b}_1 = y_2 - y_3, \quad \tilde{c}_1 = x_3 - x_2$$

and

$$\Delta = \frac{1}{2}[x_2 y_3 - y_2 x_3 - (x_1 y_3 - y_1 x_3) + x_1 y_2 - y_1 x_2] \tag{2.1}$$

The corresponding formulas for $N_2^{(n)}(x,y)$ and $N_3^{(n)}(x,y)$ can be similarly derived. We summarize the results as follows:

$$\begin{aligned} N_1^{(n)}(x,y) &= \frac{1}{2\Delta}(\tilde{a}_1 + \tilde{b}_1 x + \tilde{c}_1 y) \\ N_2^{(n)}(x,y) &= \frac{1}{2\Delta}(\tilde{a}_2 + \tilde{b}_2 x + \tilde{c}_2 y) \\ N_3^{(n)}(x,y) &= \frac{1}{2\Delta}(\tilde{a}_3 + \tilde{b}_3 x + \tilde{c}_3 y) \end{aligned} \tag{2.2}$$

where

$$\tilde{a}_i = x_j y_k - x_k y_j \ , \quad \tilde{b}_i = y_j - y_k \ , \quad \tilde{c}_i = x_k - x_j \tag{2.3}$$

for $(i,j,k) = (1,2,3), (2,3,1), (3,1,2)$. For example,

$$\tilde{c}_2 = x_1 - x_3, \quad \tilde{a}_3 = x_1 y_2 - x_2 y_1$$

The parameter Δ above is equal to the area of element e_n (and is therefore positive) if the local ordering of the nodes in e_n is counterclockwise. If the local ordering is clockwise then Δ is equal to minus the area.

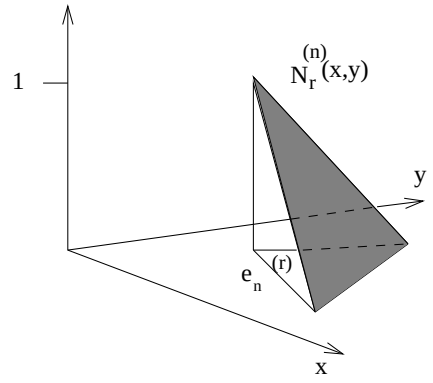


Figure 2.4: A local finite element basis function.

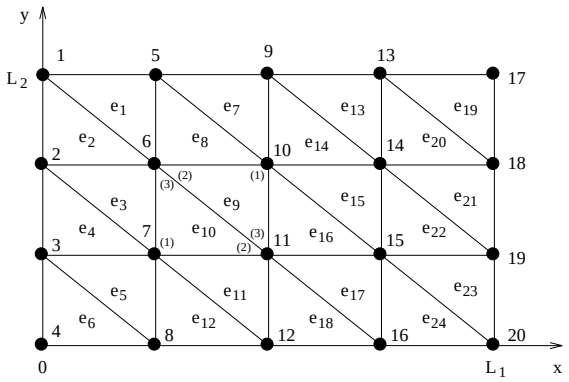


Figure 2.5: A finite element mesh on a rectangular domain.

EtoV			
n	*1*	*2*	*3*
1	5	1	6
2	2	6	1
3	6	2	7
4	3	7	2
.	.	.	.
24	16	20	15

Table 2.1: Element table for the mesh in Figure 2.5. The first column with indices n does not need to be stored in the table.

2.2 A mesh on a rectangular domain

It is convenient to have a simple finite element mesh to refer to later, and for this purpose we give in Figure 2.5 a complete mesh on a rectangular domain. The local counterclock-wise ordering of nodes shown for e_9 and e_{10} applies to all elements.

The connection between the local and global node numbers is shown in Table 2.1. This is an example of an Element-To-Vertex connectivity table `EToV` for a mesh in two space dimensions.

For use in implementations, e.g. in defining and imposing boundary conditions, it is necessary to determine the outer normal vectors to the boundary edges of elements bordering the domain boundaries. If the order of the local element vertex nodes is counterclock-wise, then for a boundary edge defined from two vertex nodes $v_1 = (x_1, y_1)^T$ and $v_2 = (x_2, y_2)^T$ we can determine a tangential vector to the edge from the differences

$$\Delta x = x_2 - x_1, \quad \Delta y = y_2 - y_1 \tag{2.4}$$

The tangential vector for the edge is

$$\mathbf{t}_{12} = (t_1, t_2)^T = (\Delta x, \Delta y)^T \tag{2.5}$$

which is by definition orthogonal to the normal vector. Hence an outer normalized vector to the element edge is found to be

$$\mathbf{n} = (n_1, n_2)^T = \frac{(t_2, -t_1)^T}{\sqrt{t_1^2 + t_2^2}} \tag{2.6}$$

2.3 Interpolation

Let there be given a function $u(x, y)$ defined on $\overline{\Omega}$, and consider the function

$$u_I(x, y) = \sum_{i=1}^{M} u(x_i, y_i) N_i(x, y), \qquad (x, y) \in \overline{\Omega} \tag{2.7}$$

Since each $N_i(x, y)$ is continuous and piecewise linear, so is $u_I(x, y)$. Further, it is easily seen that

$$u_I(x_i, y_i) = u(x_i, y_i), \quad i = 1, 2, ..., M$$

Thus $u_I(x, y)$ interpolates $u(x, y)$ at the nodes. Geometrically, $u_I(x, y)$ describes a surface made up of triangles joined at the edges.

We now give bounds for the interpolation error $(u_I(x, y) - u(x, y))$ under the assumption that the second-order derivatives of $u(x, y)$ are bounded; i.e., there exists a constant M_2 such that

$$|u_{xx}| \leq M_2\,, \quad |u_{xy}| \leq M_2\,, \quad |u_{yy}| \leq M_2\,, \qquad (x, y) \in \overline{\Omega}$$

We suppose for simplicity that the boundary of the mesh coincides with Γ.

Then a constant C exists, independent of u and the mesh, such that for $i = 1, 2, ..., N$,

$$|u_I(x, y) - u(x, y)| \leq C M_2 h_i^2\,, \qquad (x, y) \in e_i$$

where h_i is the largest edge length in e_i. (See [53]). It then follows that

$$|u_I(x, y) - u(x, y)| \leq C M_2 h^2\,, \qquad (x, y) \in \overline{\Omega} \tag{2.8}$$

where h is the largest edge length in the entire mesh.

Finite element basis functions are useful for defining a function $w(x, y)$ everywhere in $\overline{\Omega}$ given only the values w_i, $i = 1, 2, ..., M$ at the nodes. The formula is

$$w(x, y) = \sum_{i=1}^{M} w_i N_i(x, y), \qquad (x, y) \in \overline{\Omega} \tag{2.9}$$

2.4 Stationary heat conduction

In this section we show how the analysis of stationary heat conduction in a two-dimensional body leads to a partial differential equation for its temperature. The relevant quantities for this discussion are:

$q_1(x, y)$:	the heat flux in the x direction
$q_2(x, y)$:	the heat flux in the y direction
$\lambda_1(x, y)$:	the heat conductivity in the x direction
$\lambda_2(x, y)$:	the heat conductivity in the y direction
$u(x, y)$:	temperature
$\tilde{q}(x, y)$:	a heat source (if $\tilde{q}(x, y) > 0$) or heat sink (if $\tilde{q}(x, y) < 0$)

We will need Fourier's law, which states that

$$q_1(x, y) = -\lambda_1(x, y) u_x(x, y), \qquad q_2(x, y) = -\lambda_2(x, y) u_y(x, y), \qquad (x, y) \in \overline{\Omega}$$

In the following discussion (x, y) denotes a *fixed* point in the interior of the body. Consider a small square, S, with center at (x,y) and edges σ_1, σ_2, σ_3 and σ_4, as shown in Figure 2.6. For

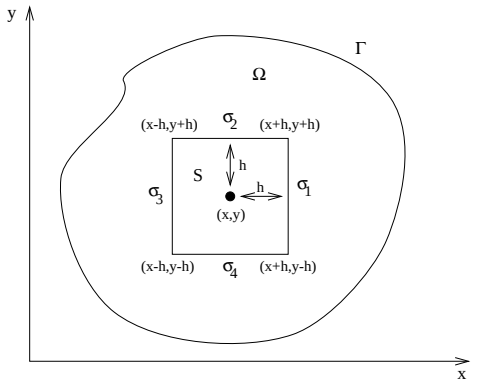


Figure 2.6: A square, S, in Ω.

sufficiently small values of h the entire square is in the body. Whether heat enters or leaves S at a given point on its boundary depends on the sign of u_x or u_y at that point and the part of the boundary the point is on. Assuming for simplicity that these derivatives are nonzero at the point (x, y), then for small enough values of h each derivative will have constant sign everywhere in the interior and on the boundary of S. In the case when both derivatives are negative, it is easily seen that heat enters S at σ_3 and σ_4 and leaves at σ_1 and σ_2. More precisely, we have:

Heat in : $\int_{\sigma_3} q_1\, ds + \int_{\sigma_4} q_2\, ds$

Heat out : $\int_{\sigma_1} q_1\, ds + \int_{\sigma_2} q_2\, ds$

Heat produced : $\int\int_S \tilde{q}\, dx\, dy$

The law of conservation of thermal energy leads to the *heat balance equation*

$$(Heat\ out) \quad - \quad (Heat\ in) \quad = \quad (Heat\ produced)$$

or

$$\int_{\sigma_1} q_1\, ds + \int_{\sigma_2} q_2\, ds - \int_{\sigma_3} q_1\, ds - \int_{\sigma_4} q_2\, ds = \int\int_S \tilde{q}\, dx\, dy \tag{2.10}$$

While we have assumed here that both u_x and u_y are negative, it is easy to confirm that (2.10) is valid regardless of the signs of these derivatives.

Consider now the integral on σ_1. This edge consists of the points $(x + h, y + s)$, $-h \le s \le h$, and we have the expansion

$$q_1(x + h, y + s) = q_1 + h(q_1)_x + s(q_1)_y + \frac{1}{2!}[h^2(q_1)_{xx} + 2hs(q_1)_{xy} + s^2(q_1)_{yy}] + \mathcal{O}(h^3)$$

where, on the right-hand side, q_1 and its derivatives are evaluated at the center, (x, y), of S. A simple calculation shows then that

$$\int_{\sigma_1} q_1\, ds = \int_{-h}^{h} q_1(x + h, y + s)\, ds = 2hq_1 + 2h^2(q_1)_x + h^3[(q_1)_{xx} + \frac{1}{3}(q_1)_{yy}] + \mathcal{O}(h^4)$$

The same analysis for the integral on σ_3 yields

$$\int_{\sigma_3} q_1\, ds = \int_{-h}^{h} q_1(x - h, y + s)\, ds = 2hq_1 - 2h^2(q_1)_x + h^3[(q_1)_{xx} + \frac{1}{3}(q_1)_{yy}] + \mathcal{O}(h^4)$$

so

$$\int_{\sigma_1} q_1\, ds - \int_{\sigma_3} q_1\, ds = 4h^2(q_1)_x + \mathcal{O}(h^4)$$

Similarly, one finds that

$$\int_{\sigma_2} q_2\, ds - \int_{\sigma_4} q_2\, ds = 4h^2 (q_2)_y + \mathcal{O}(h^4)$$

Further computation (we omit the details) shows that

$$\int\int_S \tilde{q}\, dx\, dy = 4h^2 \tilde{q}(x,y) + \mathcal{O}(h^4)$$

Combining the three last results with (2.10) and collecting terms that are $\mathcal{O}(h^4)$ yields the relation

$$4h^2[(q_1)_x + (q_2)_y] = 4h^2\tilde{q} + \mathcal{O}(h^4)$$

Dividing by $4h^2$ and letting $h \to 0$ we obtain then

$$(q_1)_x + (q_2)_y = \tilde{q}$$

Finally, from Fourier's law above we deduce the following partial differential equation for temperature:

$$(\lambda_1(x,y)u_x)_x + (\lambda_2(x,y)u_y)_y = -\tilde{q}(x,y) \tag{2.11}$$

An important special case of (2.11) is that when

$$\lambda_1(x,y) = \lambda_2(x,y) = \lambda \ \ (constant)$$

(2.11) then reduces to *Poisson's equation*

$$u_{xx} + u_{yy} = -\frac{\tilde{q}(x,y)}{\lambda} \tag{2.12}$$

If, in addition, $\tilde{q} = 0$ then we have *Laplace's equation*

$$u_{xx} + u_{yy} = 0 \tag{2.13}$$

2.5 Boundary Conditions

Equation (2.11) has many solutions. To be able to find a unique solution, we must supplement (2.11) with an appropriate boundary condition. The most common boundary conditions are the following:

Dirichlet boundary condition:

$$u = f(x,y), \quad (x,y) \in \Gamma \tag{2.14}$$

Neumann boundary condition:

$$\lambda_1(x,y)\, u_x\, n_1(x,y) + \lambda_2(x,y)\, u_y\, n_2(x,y) = -q(x,y), \quad (x,y) \in \Gamma \tag{2.15}$$

Robin boundary condition:

$$\lambda_1(x,y)\, u_x\, n_1(x,y) + \lambda_2(x,y)\, u_y\, n_2(x,y) = -h(x,y)[u - u_0(x,y)], \quad (x,y) \in \Gamma \tag{2.16}$$

Physically, the function $f(x,y)$ is a given temperature on the boundary. $q(x,y)$ is the heat flux on the boundary with respect to the outer normal vector, n_1 and n_2 being the direction cosines of this vector with respect to the x and y axes, respectively. The outer normal vector for an element edge can be determined as described in Section 2.2. (See Figure 2.7). If $q(x,y) > 0$ then heat

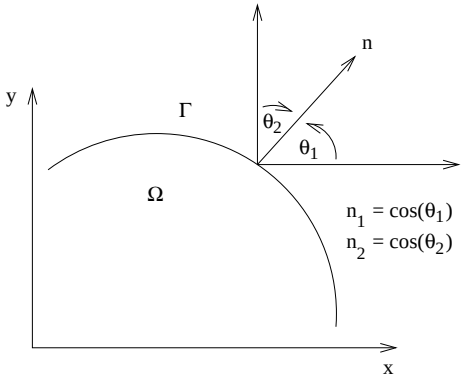


Figure 2.7: The outer normal vector on Γ.

leaves the body at point (x,y); if $q(x, y) < 0$ then heat enters the body at that point. The condition $q(x, y) = 0$ everywhere on the boundary means that the body is insulated.

$h(x, y)$ is the convective heat transfer coefficient at the boundary, and u_0 is the external (ambient) temperature.

Among the above boundary conditions, the Neumann condition (2.15) is special in that it must be combined with one of the others. This is because if a function $u(x, y)$ satisfies (2.11) in Ω and (2.15) on all of Γ then so does the function $(u(x, y) + c)$, where c is any constant. That is, we have a problem with an infinite number of solutions.

Let Γ be decomposed into two pieces, Γ_1 and Γ_2. An example of the valid use of the Neumann boundary condition is

$$\lambda_1 u_x n_1 + \lambda_2 u_y n_2 = -q, \qquad (x, y) \in \Gamma_1$$

together with

$$u = f, \qquad (x, y) \in \Gamma_2$$

or

$$\lambda_1 u_x n_1 + \lambda_2 u_y n_2 = -h(u - u_0), \qquad (x, y) \in \Gamma_2$$

Cases such as this, where boundary conditions of different types are imposed on different parts of the boundary, are called *mixed boundary conditions*.

2.6 The Finite Element Method

Here we show how to apply the finite element method to the problem consisting of the partial differential equation (2.11) together with the Dirichlet boundary condition (2.14). First, however, we must derive the so-called weak formulation associated with this problem (just as we had to derive (1.19) from (1.14) in Section 1.4).

Let $v(x, y)$ be an arbitrary smooth function defined on $\overline{\Omega}$ with the property that

$$v(x, y) = 0, \qquad (x, y) \in \Gamma \tag{2.17}$$

Multiplying both sides of (2.11) by v and integrating over $\overline{\Omega}$, we obtain

$$\int\int_\Omega [(\lambda_1 u_x)_x + (\lambda_2 u_y)_y] v \, dx \, dy = -\int\int_\Omega \tilde{q} v \, dx \, dy$$

Integration by parts with respect to x and y leads to the following fundamental identities:

$$\int\int_\Omega (\lambda_1 u_x)_x v \, dx \, dy = \int_\Gamma \lambda_1 u_x v n_1 \, ds - \int\int_\Omega \lambda_1 u_x v_x \, dx \, dy$$

$$\int\int_\Omega (\lambda_2 u_y)_y v \, dx \, dy = \int_\Gamma \lambda_2 u_y v n_2 \, ds - \int\int_\Omega \lambda_2 u_y v_y \, dx \, dy$$

Here n_1 and n_2 are the direction cosines shown in Figure 2.7. Adding the above we obtain

$$\int_\Gamma (\lambda_1 u_x n_1 + \lambda_2 u_y n_2) v \, ds - \int\int_\Omega (\lambda_1 u_x v_x + \lambda_2 u_y v_y) \, dx \, dy = -\int\int_\Omega \tilde{q} v \, dx \, dy \tag{2.18}$$

Since (2.17) makes the first integral vanish, we have then

$$\int\int_\Omega (\lambda_1 u_x v_x + \lambda_2 u_y v_y) \, dx \, dy = \int\int_\Omega \tilde{q} v \, dx \, dy \tag{2.19}$$

Thus the weak formulation of problem (2.11), (2.14) is the problem of finding a function u that

1. satisfies (2.19) for all sufficiently smooth functions v with property (2.17),
2. satisfies (2.14) on Γ.

It is this formulation that provides the basis for applying the finite element method.

Consider now a mesh on $\overline{\Omega}$ with elements e_n, $n = 1, 2, ..., N$ and nodes (x_i, y_i), $i = 1, 2, ..., M$. We seek an approximation $\hat{u}(x, y)$ to $u(x, y)$ of the form

$$\hat{u}(x, y) = \sum_{j=1}^{M} \hat{u}_j N_j(x, y) \tag{2.20}$$

where $N_j(x, y)$ is the typical global basis function. From (2.20) and the fact that $N_i(x_i, y_i) = 1$, $N_j(x_i, y_i) = 0$, $j \neq i$, we see that if we put

$$\hat{u}_i = f(x_i, y_i) \quad \text{for all} \quad i \quad \text{such that} \quad (x_i, y_i) \in \Gamma \tag{2.21}$$

then we will have

$$\hat{u}(x_i, y_i) = f(x_i, y_i) \quad \text{for all} \quad i \quad \text{such that} \quad (x_i, y_i) \in \Gamma$$

In other words, (2.21) enforces the Dirichlet boundary condition at the boundary nodes.

Let M_Ω and M_Γ denote the number of nodes in Ω and Γ, respectively. We know M_Γ of the coefficients $\hat{u}_1$,$\hat{u}_2$, ...,$\hat{u}_M$ from (2.21). To find the remaining M_Ω coefficients, we substitute (2.20) in (2.19) and let v be, successively, the functions $N_i(x, y)$ for every $(x_i, y_i) \in \Omega$. (Note that these functions vanish on Γ, as required). This procedure leads to M_Ω equations of the type

$$\int\int_\Omega [\lambda_1 \, \hat{u}_x \, (N_i)_x + \lambda_2 \, \hat{u}_y \, (N_i)_y] \, dx \, dy = \int\int_\Omega \tilde{q} N_i \, dx \, dy$$

or

$$\int\int_\Omega [\lambda_1 \sum_{j=1}^{M} \hat{u}_j \, (N_j)_x (N_i)_x + \lambda_2 \sum_{j=1}^{M} \hat{u}_j \, (N_j)_y (N_i)_y] \, dx \, dy = \int\int_\Omega \tilde{q} N_i \, dx \, dy$$

or

$$\sum_{j=1}^{M} a_{i,j} \hat{u}_j = b_i \quad \text{for all} \quad i \quad \text{such that} \quad (x_i, y_i) \in \Omega \tag{2.22}$$

where

$$a_{i,j} = \int\int_\Omega [\lambda_1 \, (N_i)_x (N_j)_x + \lambda_2 \, (N_i)_y (N_j)_y] \, dx \, dy \tag{2.23}$$

$$b_i = \int\int_\Omega \tilde{q} N_i \, dx \, dy \tag{2.24}$$

2.7 Element matrices

The form of the global basis functions is such that $a_{i,j}$ is nonzero only if $N_i(x,y)$ and $N_j(x,y)$ overlap, and this happens only if nodes (x_i,y_i) and (x_j,y_j) are directly connected in the mesh. In Figure 2.8 we see, for example, that nodes 11 and 15 are directly connected. Since $N_{11}(x,y)$ is

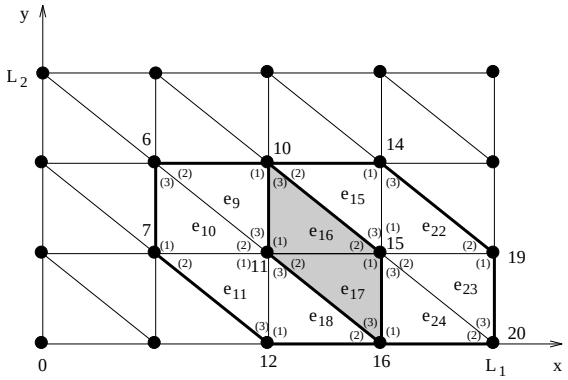


Figure 2.8: A detail of the mesh in Figure 2.5.

nonzero in elements 9, 10, 11, 16, 17 and 18, while $N_{15}(x,y)$ is nonzero in elements 15, 16, 17, 22, 23 and 24, we have

$$\begin{aligned} a_{11,15} &= \int\int_{e_{16}} [\lambda_1 (N_{11})_x (N_{15})_x + \lambda_2 (N_{11})_y (N_{15})_y]\, dx\, dy \\ &\quad + \int\int_{e_{17}} [\lambda_1 (N_{11})_x (N_{15})_x + \lambda_2 (N_{11})_y (N_{15})_y]\, dx\, dy \end{aligned}$$

In terms of the local basis functions this becomes

$$\begin{aligned} a_{11,15} &= \int\int_{e_{16}} [\lambda_1 (N_1^{(16)})_x (N_2^{(16)})_x + \lambda_2 (N_1^{(16)})_y (N_2^{(16)})_y]\, dx\, dy \\ &\quad + \int\int_{e_{17}} [\lambda_1 (N_2^{(17)})_x (N_1^{(17)})_x + \lambda_2 (N_2^{(17)})_y (N_1^{(17)})_y]\, dx\, dy \end{aligned}$$

We now introduce for each element e_n the element matrix

$$\mathbf{K_n} = \begin{bmatrix} k_{1,1}^{(n)} & k_{1,2}^{(n)} & k_{1,3}^{(n)} \\ k_{2,1}^{(n)} & k_{2,2}^{(n)} & k_{2,3}^{(n)} \\ k_{3,1}^{(n)} & k_{3,2}^{(n)} & k_{3,3}^{(n)} \end{bmatrix}$$

where

$$k_{r,s}^{(n)} = \int\int_{e_n} [\lambda_1 (N_r^{(n)})_x (N_s^{(n)})_x + \lambda_2 (N_r^{(n)})_y (N_s^{(n)})_y]\, dx\, dy \tag{2.25}$$

Observe that we can express $a_{11,15}$ above as

$$a_{11,15} = k_{1,2}^{(16)} + k_{2,1}^{(17)}$$

This is typical: each nonzero $a_{i,j}$ in (2.23) is the sum of some number of element matrix entries. If $i \neq j$ then this number is precisely two. If $i = j$ then it is the number of elements to which the ith node belongs.

We regard now b_i given by (2.24). From Figure 2.8 we see, for example, that

$$\begin{aligned} b_{11} &= \sum_{n=9,10,11,16,17,18} \int\int_{e_n} \tilde{q} N_{11}\, dx\, dy \\ &= \int\int_{e_9} \tilde{q} N_3^{(9)}\, dx\, dy + \int\int_{e_{10}} \tilde{q} N_2^{(10)}\, dx\, dy + \int\int_{e_{11}} \tilde{q} N_1^{(11)}\, dx\, dy \\ &\quad + \int\int_{e_{16}} \tilde{q} N_1^{(16)}\, dx\, dy + \int\int_{e_{17}} \tilde{q} N_2^{(17)}\, dx\, dy + \int\int_{e_{18}} \tilde{q} N_3^{(18)}\, dx\, dy \end{aligned}$$

For each element e_n we define an element vector by

$$\tilde{\mathbf{q}}_\mathbf{n} = \begin{bmatrix} \tilde{q}_1^{(n)} \\ \tilde{q}_2^{(n)} \\ \tilde{q}_3^{(n)} \end{bmatrix}$$

where

$$\tilde{q}_r^{(n)} = \int\int_{e_n} \tilde{q} N_r^{(n)}\, dx\, dy \tag{2.26}$$

Every b_i can be expressed as a sum of components of element vectors. For example,

$$b_{11} = \tilde{q}_3^{(9)} + \tilde{q}_2^{(10)} + \tilde{q}_1^{(11)} + \tilde{q}_1^{(16)} + \tilde{q}_2^{(17)} + \tilde{q}_3^{(18)}$$

To evaluate $k_{r,s}^{(n)}$ and $\tilde{q}_r^{(n)}$ we insert in (2.25) and (2.26), respectively, the expressions for the local basis functions given in (2.2). The result is

$$k_{r,s}^{(n)} = \frac{1}{4\Delta^2} \int\int_{e_n} (\lambda_1 \tilde{b}_r \tilde{b}_s + \lambda_2 \tilde{c}_r \tilde{c}_s)\, dx\, dy\,, \qquad r,s = 1,2,3$$

$$\tilde{q}_r^{(n)} = \frac{1}{2\Delta} \int\int_{e_n} \tilde{q}(\tilde{a}_r + \tilde{b}_r x + \tilde{c}_r y)\, dx\, dy\,, \qquad r = 1,2,3$$

In the case when λ_1, λ_2 and $\tilde{q}$ are constant with respect to x and y, one finds that

$$k_{r,s}^{(n)} = \frac{1}{4|\Delta|}(\lambda_1 \tilde{b}_r \tilde{b}_s + \lambda_2 \tilde{c}_r \tilde{c}_s) \tag{2.27}$$

$$\tilde{q}_r^{(n)} = \frac{|\Delta|}{3} \tilde{q} \tag{2.28}$$

If any of these functions is not constant, it can be approximated in (2.27) or (2.28) either by its value at the center $(x_c\,, y_c)$ of e_n, defined by

$$(x_c\,, y_c) = \left(\frac{x_1 + x_2 + x_3}{3}\,,\ \frac{y_1 + y_2 + y_3}{3} \right) \tag{2.29}$$

or by the average of its values at the three nodes. (Note that in (2.29) we are using *local* node numbers).

2.8 The computation

We need to compute the various coefficients $a_{i,j}$ and b_i in system (2.22) and then solve this system. Regarding the first task, it is convenient to divide this into two parts, the first part performing most of the work but ignoring the boundary condition and the second part imposing the Dirichlet boundary condition. This approach leads to Algorithms 4 and 6 below which are analogous to Algorithms 1 and 2, respectively, in Section 1.5.

We have seen that each nonzero $a_{i,j}$ can be expressed as the sum of entries of element matrices and that each nonzero b_i can be expressed as the sum of components of element vectors. This fact makes it efficient to construct the various $a_{i,j}$ and b_i by an 'elementwise' computation. More precisely, we can go through the elements one by one, computing for each element its element matrix and element vector and using this data to update the $a_{i,j}$ and b_i affected by that element. This procedure is carried out by the algorithm below which makes use of an $M \times M$ array, a, for the $a_{i,j}$ and an $M \times 1$ array, b, for the b_i. The use of an $M \times M$ array for the $a_{i,j}$ is for clarity only. In practice a sparse data structure would be used for these values.

It should be realized that although the algorithm affects all rows of arrays a and b, the only rows relevant for system (2.22) are those with a row number i such that $(x_i, y_i) \in \Omega$. The other rows will be modified later so as to express the boundary condition.

Algorithm 4: Global assembly of coefficient matrix **A** and vector **b** (2D).

Allocate full arrays **A** and **b**
for $n := 1$ **to** N
    Look up the global numbers (i, j, k) and (x, y)-coordinates of the nodes in e_n.
    for $r := 1$ **to** 3
        Compute $\tilde{q}_r^{(n)}$ from (2.28).
        $b[i] := b[i] + \tilde{q}_r^{(n)}$ (N.B. i is the global number of node r).
        for $s := 1$ **to** 3
            Compute $k_{r,s}^{(n)}$ from (2.27).
            $a[i, j] := a[i, j] + k_{r,s}^{(n)}$ (N.B. j is the global number of node s).

It will be noted that the algorithm takes no advantage of the symmetry inherent in (2.23) and (2.25), namely $a_{i,j} = a_{j,i}$ and $k_{r,s}^{(n)} = k_{s,r}^{(n)}$. These relations show that it suffices to compute $a_{i,j}$ for $j \geq i$ and $k_{r,s}^{(n)}$ for $s \geq r$. Another weakness is the inefficiency of the initialization of array **A**. We recall that the matrix entry $a_{i,j}$ is nonzero only if nodes (x_i, y_i) and (x_j, y_j) are directly connected in the mesh. Hence, logically, only the corresponding entries of the array need to be addressed. Algorithm 5 below eliminates both types of unnecessary work with only minor changes to Algorithm 4 and making use of sparse storage. In Matlab we can make use of the sparse commands `sparse` and `spalloc` (refer to the Matlab help info for appropriate use hereof).

Algorithm 5: Global assembly of coefficient matrix **A** and vector **b** (2D, exploit symmetry).

Allocate sparse **A** and **b**
for $n := 1$ **to** N
    Look up the global numbers (i, j) and (x, y)-coordinates of the nodes in e_n.
    for $r := 1$ **to** 3
        Compute $\tilde{q}_r^{(n)}$ from (2.28).
        $b[i] := b[i] + \tilde{q}_r^{(n)}$
        for $s := r$ **to** 3
            Compute $k_{r,s}^{(n)}$ from (2.27).
            if $j \geq i$
                $a[i, j] := a[i, j] + k_{r,s}^{(n)}$
            else
                $a[j, i] := a[j, i] + k_{r,s}^{(n)}$

We consider now the handling of the Dirichlet boundary condition. First, to illustrate what we want to achieve, consider the case where $M = 4$ and make the assumption that there is only a single

node on the boundary, namely (x_2, y_2). Let the system of equations as it exists after Algorithm 4 be expressed as

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \\ \hat{u}_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

The boundary condition makes $\hat{u}_2 = f_2$. Hence the system we want to solve may be expressed as

$$\begin{bmatrix} a_{1,1} & 0 & a_{1,3} & a_{1,4} \\ 0 & 1 & 0 & 0 \\ a_{3,1} & 0 & a_{3,3} & a_{3,4} \\ a_{4,1} & 0 & a_{4,3} & a_{4,4} \end{bmatrix} \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \\ \hat{u}_4 \end{bmatrix} = \begin{bmatrix} b_1 - a_{1,2} f_2 \\ f_2 \\ b_3 - a_{3,2} f_2 \\ b_4 - a_{4,2} f_2 \end{bmatrix} \tag{2.30}$$

The purpose of Algorithm 6 is to perform modifications of this type and it should be executed after Algorithm 4.

Algorithm 6: Imposing boundary conditions by modification of system (2D).

```
if (x_i, y_i) ∈ Γ
    a[i,i] := 1, b[i] := f_i
    for j := 1 to M   (j ≠ i)
        a[i,j] := 0
        if (x_j, y_j) ∈ Ω
            b[j] := b[j] − a[j,i] * f_i
            a[j,i] := 0
```

The weaknesses of this computation are that it fails to exploit symmetry and that much of the execution time is spent on pointless work (because most of the $a[i,j]$ and $a[j,i]$ in the innermost for-loop contain zeros from the start). Algorithm 7 below, which should be executed after Algorithm 4, both takes symmetry into account and avoids (most) pointless work. Let S_Γ denote the set of global node numbers of nodes on Γ. For every $i \in S_\Gamma$, let $S_{\overline{\Omega}}^{(i)}$ denote the set of global node numbers of those nodes in $\overline{\Omega}$ that are directly connected to $(x_i\,, y_i)$. For the mesh in Figure 2.5, for example, we have

$$S_\Gamma = \{1,2,3,4,5,8,9,12,13,16,17,18,19,20\}, \quad S_{\overline{\Omega}}^{(5)} = \{1,6,9,10\}$$

Algorithm 7: Imposing boundary conditions by modification of system (2D, exploit symmetry).

```
for  i ∈ S_Γ
    a[i,i] := 1, b[i] := f_i
    for  j ∈ S_Ω̄^(i)
        if j < i
            b[j] := b[j] − a[j,i] * f_i
            a[j,i] := 0
        else
            b[j] := b[j] − a[i,j] * f_i
            a[i,j] := 0
```

After conclusion of Algorithm 7 we have an M'th-order system of equations to be solved that can be expressed in the compact form as a system of equations

$$\boldsymbol{A}\hat{\boldsymbol{u}} = \boldsymbol{b} \tag{2.31}$$

For the mesh shown in Figure 2.5, $\boldsymbol{A}$ and $\boldsymbol{b}$ have the following appearance:

$$\begin{bmatrix}
1 & & & & & & & & & & & & & & & & & & & \\
 & 1 & & & & & & & & & & & & & & & & & & \\
 & & 1 & & & & & & & & & & & & & & & & & \\
 & & & 1 & & & & & & & & & & & & & & & & \\
 & & & & 1 & & & & & & & & & & & & & & & \\
 & & & & & x & x & & & x & x & & & & & & & & & \\
 & & & & & x & x & & & & x & & & & & & & & & \\
 & & & & & & & 1 & & & & & & & & & & & & \\
 & & & & & & & & 1 & & & & & & & & & & & \\
 & & & & & x & & & & x & x & & & x & x & & & & & \\
 & & & & & x & x & & & x & x & & & & x & & & & & \\
 & & & & & & & & & & & 1 & & & & & & & & \\
 & & & & & & & & & & & & 1 & & & & & & & \\
 & & & & & & & & & x & & & & x & x & & & & & \\
 & & & & & & & & & x & x & & & x & x & & & & & \\
 & & & & & & & & & & & & & & & 1 & & & & \\
 & & & & & & & & & & & & & & & & 1 & & & \\
 & & & & & & & & & & & & & & & & & 1 & & \\
 & & & & & & & & & & & & & & & & & & 1 & \\
 & & & & & & & & & & & & & & & & & & & 1
\end{bmatrix}, \quad \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ x \\ x \\ f_8 \\ f_9 \\ x \\ x \\ f_{12} \\ f_{13} \\ x \\ x \\ f_{16} \\ f_{17} \\ f_{18} \\ f_{19} \\ f_{20} \end{bmatrix}$$

For every $i \in S_\Gamma$, the ith row of this system is of the form $\hat{u}_i = f_i$, and these rows are uncoupled from the remainder of the system with inhomogenous boundary contributions now imposed through the right hand vector **b**. This uncoupling is most obvious when the six nodes in the interior of the domain are ordered before those on the boundary. Consider, for example, the node ordering shown in Figure 2.9.

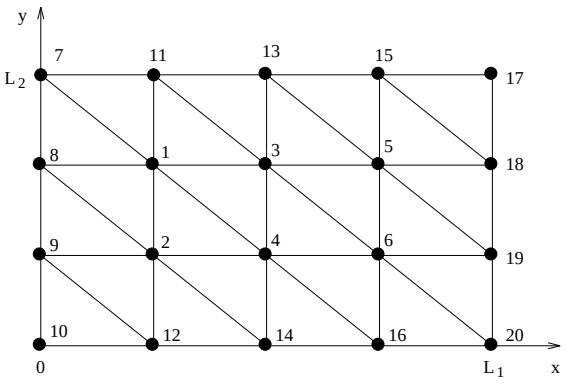


Figure 2.9: Reordering of nodes in the mesh of Figure 2.5.

Then $\boldsymbol{A}$ and $\boldsymbol{b}$ assume the form

$$\begin{bmatrix}
x & x & x & x & & & & & & & & & & & & & & & & \\
x & x & & x & & & & & & & & & & & & & & & & \\
x & & x & x & x & x & & & & & & & & & & & & & \\
x & x & x & x & & x & & & & & & & & & & & & & \\
 & & x & & x & x & & & & & & & & & & & & & \\
 & & x & x & x & x & & & & & & & & & & & & & \\
 & & & & & & 1 & & & & & & & & & & & & & \\
 & & & & & & & 1 & & & & & & & & & & & & \\
 & & & & & & & & 1 & & & & & & & & & & & \\
 & & & & & & & & & 1 & & & & & & & & & & \\
 & & & & & & & & & & 1 & & & & & & & & & \\
 & & & & & & & & & & & 1 & & & & & & & & \\
 & & & & & & & & & & & & 1 & & & & & & & \\
 & & & & & & & & & & & & & 1 & & & & & & \\
 & & & & & & & & & & & & & & 1 & & & & & \\
 & & & & & & & & & & & & & & & 1 & & & & \\
 & & & & & & & & & & & & & & & & 1 & & & \\
 & & & & & & & & & & & & & & & & & 1 & & \\
 & & & & & & & & & & & & & & & & & & 1 & \\
 & & & & & & & & & & & & & & & & & & & 1
\end{bmatrix}, \quad \begin{bmatrix} x \\ x \\ x \\ x \\ x \\ x \\ f_7 \\ f_8 \\ f_9 \\ f_{10} \\ f_{11} \\ f_{12} \\ f_{13} \\ f_{14} \\ f_{15} \\ f_{16} \\ f_{17} \\ f_{18} \\ f_{19} \\ f_{20} \end{bmatrix}$$

The first six unknowns can be found by solving the smaller system

$$\begin{bmatrix}
x & x & x & x & & \\
x & x & & x & & \\
x & & x & x & x & x \\
x & x & x & x & & x \\
 & & x & & x & x \\
 & & x & x & x & x
\end{bmatrix}
\begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \\ \hat{u}_4 \\ \hat{u}_5 \\ \hat{u}_6 \end{bmatrix} = \begin{bmatrix} x \\ x \\ x \\ x \\ x \\ x \end{bmatrix}$$

In general, whenever the boundary condition is of Dirichlet type then one could work with a system of order M_Ω instead of one of order M. However, it is usually easiest *not* to do so, and since the difference in the sizes of the two systems loses significance as the mesh is refined (because there are $\mathcal{O}(M)$ interior nodes and only $\mathcal{O}(\sqrt{M})$ boundary nodes), there is little reason not to work with the larger system.

We consider now methods for solving (2.31) and the related issue of the choice of a practical data structure for $\boldsymbol{A}$. This matrix is symmetric positive definite. In the case of a fine mesh it is also very sparse, since the number of off-diagonal nonzeros in the ith row cannot exceed the number of nodes directly connected to node (x_i, y_i). For a mesh of the type shown in Figure 2.5, for example, where each node is connected to at most 6 other nodes, the number of nonzero entries in a row is at most 7 (including the entry on the main diagonal). Since the total number of entries of $\boldsymbol{A}$ is M^2, the density of the nonzero entries in this case is bounded by $7/M$ and hence goes to zero as the mesh is refined.

An important type of sparse matrix is a *band matrix*, where the nonzero entries are more or less concentrated along the main diagonal. Let $\phi(i)$ denote the column number of the last nonzero in the ith row of $\boldsymbol{A}$, and let

$$h(\boldsymbol{A}) = \max_{1 \le i \le M} \{\phi(i) - i + 1\}$$

$$b(\boldsymbol{A}) = 2h(\boldsymbol{A}) - 1$$

$h(\boldsymbol{A})$ and $b(\boldsymbol{A})$ are called the *half-bandwidth* and *bandwidth* of $\boldsymbol{A}$, respectively. For example, the matrix immediately below (2.31) has $h(\boldsymbol{A}) = 6$ and $b(\boldsymbol{A}) = 11$. Since $\phi(i)$ is the largest node number of all nodes connected directly to (x_i, y_i), it follows that $\boldsymbol{A}$ has a small bandwidth only

if the node ordering is such that every pair of connected nodes have node numbers close to one another. There are effective algorithms for finding node orderings that come close to minimizing the bandwidth of a matrix; see, for example, [25] and [35]. Generally speaking, the possibility of achieving a small bandwidth is greatest in the case of a long and narrow mesh, in which case the node ordering should run along the 'short' dimension. See Figure 2.10 for a simple example.

However, it turns out that the minimal bandwidth for all node orderings is no better than

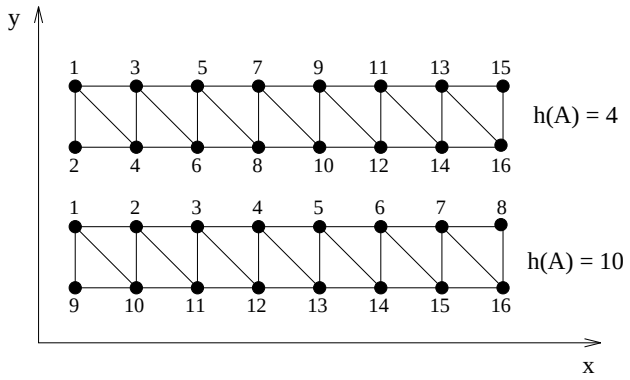


Figure 2.10: Two orderings of a mesh and the corresponding half-bandwidths. (Boundary conditions are ignored).

$b(\boldsymbol{A}) = \mathcal{O}(\sqrt{M})$. Consequently, as the mesh is refined the bandwidth increases and the density of nonzero entries in the band falls like $\mathcal{O}(1/\sqrt{M})$. This is a fundamental difference between problems in one and several space dimensions (the matrix in (1.30) has bandwidth 3 for *all* M) and tends to make band matrix techniques less effective in problems with more than one space dimension.

There are two general groups of methods for solving (2.31): direct and iterative. In the absence of rounding errors, direct methods can solve (2.31) exactly in a finite number of simple arithmetical operations. (Here we regard the computation of a square root as a simple arithmetical operation). Examples are Gaussian elimination and the Cholesky method. Unfortunately, in the case of sparse systems direct methods usually create fill-ins; i.e., nonzero entries in the matrix positions originally occupied by zeros. Nevertheless, there is a strong tradition for applying direct methods to finite element problems. Since these methods are easy to program when $\boldsymbol{A}$ is a band matrix, it is very common to store $\boldsymbol{A}$ in this form. For example, the diagonals in the upper half-band (including the main diagonal) can be stored as columns in an $(M \times h(\boldsymbol{A}))$ array. This is sufficient storage because fill-ins are confined to the band (and tend to fill it). When $\boldsymbol{A}$ is treated as a band matrix, it is naturally desirable that the mesh nodes have been ordered to yield as small a bandwidth as possible. In cases where $\boldsymbol{A}$ is sparse but does not have particular band structure, it is possible to employ various fill-in minimizing algorithms. For example, if the Cholesky method is employed for the solution of a sparse symmetric positive definite system the Minimum Degree (MD) algorithm can often be used to reduce the fill-in. In Matlab, this can be achieved by employing `symmmd` or the approximate multiple minimum degree function `symamd`. For a thorough treatment of direct methods for solving sparse, symmetric positive definite systems, see [25].

An iterative method for solving (2.31) produces a sequence of vectors that converges to the solution vector $\hat{\boldsymbol{u}}$. Examples are the successive overrelaxation (SOR) method and the conjugate gradient (CG) method. Iterative methods, in their simplest form, use only the original matrix entries, so there is no fill-in. The CG method, for example, requires only that the matrix-vector product $\boldsymbol{Au}$ can be computed for any given $\boldsymbol{u}$. The only $a_{i,j}$ entries that need to be stored are precisely those produced by Algorithms 4 and 7. Let the number of these be R. A simple storage scheme for $\boldsymbol{A}$ consists then of three single-indexed arrays of length R: one for the values of i, one for the values of j, and one for the values of $a_{i,j}$, where $j \geq i$.

The literature that deals with solving systems of type (2.31) is very extensive, and the above

discussion has only scratched the surface. For further reading see, for example, [3], [4], [10], [25], [31], [32], [35] and [49].

2.9 The Neumann boundary condition

We consider now the procedure for solving differential equation (2.11) when the boundary condition on a part of Γ is of Neumann type (2.15). As stated earlier, this boundary condition cannot be applied to the whole of Γ because the solution of the problem is then not unique. We therefore suppose that the boundary is divided into two parts,

$$\Gamma = \Gamma_1 \bigcup \Gamma_2$$

and that $u(x,y)$ is required to satisfy

$$\lambda_1(x,y)\, u_x\, n_1(x,y) + \lambda_2(x,y)\, u_y\, n_2(x,y) = -q(x,y)\,, \qquad (x,y) \in \Gamma_1 \tag{2.32}$$

$$u = f(x,y)\,, \qquad (x,y) \in \Gamma_2 \tag{2.33}$$

To derive the weak formulation of this problem, we return to (2.18) and require now that v be zero only on Γ_2. Using (2.32), we can rewrite (2.18) as

$$\int\int_{\Omega} (\lambda_1 u_x v_x + \lambda_2 u_y v_y)\, dx\, dy = \int\int_{\Omega} \tilde{q} v\, dx\, dy - \int_{\Gamma_1} qv\, ds \tag{2.34}$$

The problem of finding a function u that satisfies (2.11) in Ω and boundary conditions (2.32), (2.33) is equivalent to finding a function u that

1. satisfies (2.34) for all sufficiently smooth v such that $v = 0$ on Γ_2,
2. satisfies (2.33).

Let there be given a mesh on $\overline{\Omega}$ with M nodes. We recall that M_Ω and M_Γ denote, respectively, the number of nodes in Ω and the number of nodes on Γ. We define further:

M_{Γ_1}:	The number of nodes on Γ_1
M_{Γ_2}:	The number of nodes on Γ_2
S_Ω	$i \in S_\Omega \;\Leftrightarrow\; (x_i, y_i) \in \Omega$
S_{Γ_1}:	$i \in S_{\Gamma_1} \;\Leftrightarrow\; (x_i, y_i) \in \Gamma_1$
S_{Γ_2}:	$i \in S_{\Gamma_1} \;\Leftrightarrow\; (x_i, y_i) \in \Gamma_2$

We assume that nodes are placed at the points where Γ_1 and Γ_2 meet and that these nodes are assigned to Γ_2.

To solve the above problem by the FEM, we again approximate $u(x,y)$ by a function $\hat{u}(x,y)$ of type (2.20). Because of the Dirichlet boundary condition on Γ_2, we put

$$\hat{u}_i = f_i\,, \quad i \in S_{\Gamma_2} \tag{2.35}$$

which gives us M_{Γ_2} equations. To derive the remaining $(M - M_{\Gamma_2})$ equations, we substitute (2.20) for u in (2.34) and let v be, successively, all of the functions $N_i(x,y)$ such that $(x_i, y_i) \in \Omega \bigcup \Gamma_1$; i.e., such that $i \in S_\Omega \bigcup S_{\Gamma_1}$. The number of these functions is precisely $(M - M_{\Gamma_2})$, and they all vanish on Γ_2 as required. This procedure yields the system

$$\sum_{j=1}^{M} a_{i,j} \hat{u}_j = b_i\,, \quad i \in S_\Omega \bigcup S_{\Gamma_1} \tag{2.36}$$

where

$$a_{i,j} = \int\int_{\Omega} [\lambda_1 (N_i)_x (N_j)_x + \lambda_2 (N_i)_y (N_j)_y] \, dx \, dy \tag{2.37}$$

$$b_i = \int\int_{\Omega} \tilde{q} N_i \, dx \, dy - \int_{\Gamma_1} q N_i \, ds \tag{2.38}$$

Comparing with (2.23) and (2.24), we see that the only complication caused by the Neumann boundary condition is that we must now deal with the *boundary integrals*

$$\int_{\Gamma_1} q N_i \, ds \,, \quad i \in S_\Omega \bigcup S_{\Gamma_1} \tag{2.39}$$

However, since the form of $N_i(x, y)$ implies that (2.39) can be nonzero only if $(x_i, y_i) \in \Gamma_1$, the problem reduces to that of computing

$$\int_{\Gamma_1} q N_i \, ds \,, \quad i \in S_{\Gamma_1} \tag{2.40}$$

Consider, as an example, the case $i = 17$ in Figure 2.11. Since $N_{17}(x, y)$ is zero everywhere

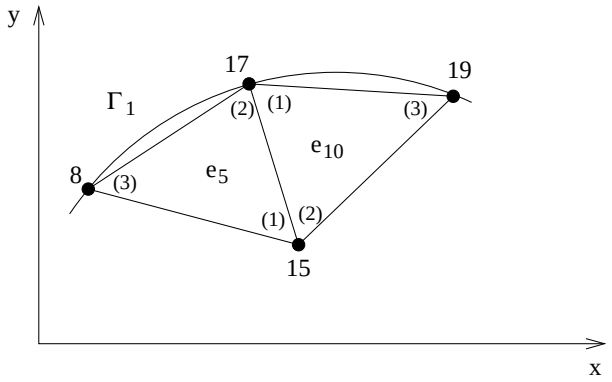


Figure 2.11: Two element edges on Γ_1.

except in elements e_5 and e_{10} , we have

$$\begin{aligned} \int_{\Gamma_1} q N_{17} \, ds &= \int_{(x_{19}, y_{19})}^{(x_8, y_8)} q N_{17} \, ds \\ &= \int_{(x_{19}, y_{19})}^{(x_{17}, y_{17})} q N_{17} \, ds + \int_{(x_{17}, y_{17})}^{(x_8, y_8)} q N_{17} \, ds \\ &= \int_{(x_{19}, y_{19})}^{(x_{17}, y_{17})} q N_1^{(10)} \, ds + \int_{(x_{17}, y_{17})}^{(x_8, y_8)} q N_2^{(5)} \, ds \end{aligned}$$

This example is typical: We can always express (2.40) as the sum of two integrals, each defined on a single element edge on Γ_1. We call an element edge on the boundary a *boundary edge*. Note that a boundary edge on Γ_1 affects only two of the entries b_i in (2.38). From Figure 2.11, for example, we see that the boundary edge from node 17 to node 8 affects only b_{17} and b_8.

The phrase 'from node 17 to node 8' stresses that we have assigned to the boundary edge a *first* node (node 17) and a *second* node (node 8). We call node 17 the first node because it is the first node we reach when making a counterclockwise movement along the boundary of e_5 , starting at the node *not* belonging to the boundary edge (node 15). We can construct a table of boundary edges in which each edge on Γ_1 is represented by two integers: n, the number of the element to which the edge belongs, and r, the local node number of the first node on the edge. Assuming

that the local ordering of nodes in every element is counterclockwise, it is easy to determine s, the local node number of the second node on the edge.

Let E_1 denote the number of boundary edges on Γ_1, and let these edges be ordered in some way. We define

$$\boldsymbol{q}_p = \begin{bmatrix} q_1^{(p)} \\ q_2^{(p)} \end{bmatrix}, \quad p = 1, 2, \ldots, E_1$$

where

$$q_1^{(p)} = \int_{(x_i,y_i)}^{(x_j,y_j)} qN_r^{(n)}\, ds\,, \quad q_2^{(p)} = \int_{(x_i,y_i)}^{(x_j,y_j)} qN_s^{(n)}\, ds$$

To sum up the notation: p is the edge number, n is the element number, r and s are the local numbers of the first and second nodes on the edge, respectively, and i and j are the corresponding global node numbers. When $q(x, y)$ is constant then

$$q_1^{(p)} = q_2^{(p)} = \frac{q}{2}\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \tag{2.41}$$

Otherwise, one can approximate q in (2.41) by either its value at the edge midpoint

$$(x_c, y_c) = \left(\frac{x_i + x_j}{2}\,,\ \frac{y_i + y_j}{2}\right) \tag{2.42}$$

or by the average of its values at the end nodes.

Algorithm 8 performs the updating required by the boundary integrals in (2.38).

Algorithm 8: Imposing Neumann boundary conditions by modification of system (2D).

for $p := 1$ **to** E_1
Look up n and r for the pth edge.
Determine s and look up i and j and the (x,y)-coordinates of the end nodes.
Compute $q_1^{(p)}$ and $q_2^{(p)}$ from (2.41).
$b[i] := b[i] - q_1^{(p)}$
$b[j] := b[j] - q_2^{(p)}$

Conclusion

When the problem to be solved consists of differential equation (2.11) and boundary conditions (2.32) and (2.33), then the computational procedure for constructing the algebraic system of equations $\boldsymbol{A\hat{u}} = \boldsymbol{b}$ without exploiting symmetry of the coefficient matrix A is as follows:

1. Global assembly, Algorithm 4,
2. Modification for imposing Neumman boundary conditions, Algorithm 8,
3. Modification for imposing Dirichlet boundary conditions, Algorithm 6 (with S_Γ replaced by S_{Γ_2}).

The order of steps 2 and 3 must not be reversed, since this can lead to incorrect values of b_i for values of i such that (x_i, y_i) is a node where Γ_1 and Γ_2 meet.

2.10 The Robin boundary condition

We consider now the case when the boundary condition is of the Robin type (2.16) on all of Γ. Returning to (2.18), we must now remove the requirement that v vanish on Γ. Noting that the

integrand in the boundary integral appears in (2.16), we see that the weak formulation of problem (2.11), (2.16) is that of determining the function u that satisfies

$$\int\int_\Omega (\lambda_1 u_x v_x + \lambda_2 u_y v_y)\, dx\, dy + \int_\Gamma huv\, ds = \int\int_\Omega \tilde{q}v\, dx\, dy + \int_\Gamma hu_0\, v\, ds \tag{2.43}$$

for all sufficiently smooth v. Substituting (2.20) for u in (2.43) and letting v be, successively, the M global basis functions, we obtain the system of equations

$$\sum_{j=1}^{M} a_{i,j}\hat{u}_j = b_i\,, \qquad i \;=\; 1, 2, ..., M \tag{2.44}$$

where

$$a_{i,j} = \int\int_\Omega [\lambda_1 (N_i)_x (N_j)_x + \lambda_2 (N_i)_y (N_j)_y]\, dx\, dy + \int_\Gamma hN_i N_j\, ds \tag{2.45}$$

$$b_i = \int\int_\Omega \tilde{q} N_i\, dx\, dy + \int_\Gamma hu_0\, N_i\, ds \tag{2.46}$$

Comparing with the case of the Neumann boundary condition, we see that the only really new feature here is the boundary integral in (2.45). Let E denote the number of boundary edges on Γ, and let these edges be ordered in some way. Let

$$\boldsymbol{H}_p = \begin{bmatrix} h_{1,1}^{(p)} & h_{1,2}^{(p)} \\ \\ h_{2,1}^{(p)} & h_{2,2}^{(p)} \end{bmatrix}, \quad p = 1, 2, \ldots, E$$

where

$$h_{1,1}^{(p)} = \int_{(x_i,y_i)}^{(x_j,y_j)} h\left(N_r^{(n)}\right)^2 ds\,, \qquad h_{1,2}^{(p)} = \int_{(x_i,y_i)}^{(x_j,y_j)} hN_r^{(n)} N_s^{(n)}\, ds$$

$$h_{2,1}^{(p)} = \int_{(x_i,y_i)}^{(x_j,y_j)} hN_s^{(n)} N_r^{(n)}\, ds\,, \qquad h_{2,2}^{(p)} = \int_{(x_i,y_i)}^{(x_j,y_j)} h\left(N_s^{(n)}\right)^2 ds$$

When $h(x, y)$ is constant one finds that

$$h_{1,1}^{(n)} = h_{2,2}^{(n)} = \frac{h}{3}\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \tag{2.47}$$

$$h_{1,2}^{(n)} = h_{2,1}^{(n)} = \frac{h}{6}\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \tag{2.48}$$

Algorithm 9 below performs the updating required by the boundary integrals in (2.45).

Algorithm 9: Imposing Robin boundary conditions by modification of system (2D).

for $p := 1$ **to** E
  Look up n and r for the p'th edge.
  Determine s and look up i, j and the (x,y)-coordinates of the end nodes.
  Compute $h_{1,1}^{(p)}, h_{1,2}^{(p)}$ and $h_{2,2}^{(p)}$ from (2.47) and (2.48).
  $a[i,i] := a[i,i] + h_{1,1}^{(n)}$
  $a[j,j] := a[j,j] + h_{2,2}^{(n)}$
  if $j > i$
    $a[i,j] := a[i,j] + h_{1,2}^{(n)}$
  else
    $a[j,i] := a[j,i] + h_{1,2}^{(n)}$

Conclusion

When the problem to be solved consists of differential equation (2.11) and the Robin boundary condition (2.16), then the computational procedure for constructing the algebraic system of equations $\boldsymbol{A}\hat{\boldsymbol{u}} = \boldsymbol{b}$ without exploiting symmetry of the coefficient matrix $\boldsymbol{A}$ is as follows:

1. Global assembly, Algorithm 4,
2. Modification for imposing Neumman boundary conditions, Algorithm 8, with E_1 replaced by E and q replaced by $(-hu_0)$),
3. Modification for imposing Robin boundary conditions, Algorithm 9.