

# Preface

The finite element method is the most popular general purpose technique for computing accurate solutions to partial differential equations (PDEs). Since PDEs form the basis for many mathematical models in the physical sciences and, increasingly, in other fields as well, it would be difficult to overstate the importance of the finite element method.

There are a number of excellent books, such as Brenner and Scott [13], Strang and Fix [41], and Ciarlet [16], covering the theory of finite elements, but these books tend to devote little attention to the practical details of programming the algorithms. While the occasional talented student can work out a reasonable scheme without further help, I think most students will benefit from a careful explanation of data structures and specific coding strategies. This book explains how to write a finite element code from scratch. In addition, it comes with a collection of MATLAB<sup>®</sup> programs implementing the ideas presented in the book. Students can use these codes to experiment with the method and extend them in various ways to learn more about programming finite elements.

In addition to a careful explanation of computer implementation (Part II), I have also included, in Part I, an overview of the theoretical basis of the finite element method. My purpose is to give the reader a good understanding of the “big picture” without getting bogged down in the technical details of the theory. The overview also serves to define the context and notation for discussing computer codes.

The finite element method reduces a boundary value problem for a linear PDE to a system of linear equations, written in matrix-vector form as  $KU = F$ , that must be solved. Part I derives this system of equations, and the algorithms in Part II show how to compute the matrix  $K$  and the vector  $F$ . Part III presents algorithms for solving  $KU = F$  efficiently even when this system is very large.

The final part of the book discusses the related issues of *a posteriori* error estimation and adaptive error reduction. It is possible to analyze the computed finite element solution so as to estimate the errors present in the solution and to determine the regions of the computational domain where the solution can most profitably be improved. Part IV explains the various aspects of developing an adaptive finite element algorithm.

Throughout the book, my goal is to provide students with a practical, working knowledge of finite elements. This knowledge should provide an excellent foundation for those who wish to delve into advanced texts on the subject.

## Detailed outline of the book

Although I mention *the* finite element method above, in fact, there are a number of finite element methods, sharing common features but with important differences. I focus my attention on the *Galerkin* finite element for steady-state boundary value problems (BVPs). The Galerkin method has a strong and elegant theoretical base that is accessible to undergraduate students with some knowledge of linear algebra.

In Chapter 1, I present the PDEs that are the focus of this book and discuss the physical phenomena that they model. As I mentioned in the previous paragraph, these models are steady state, that is, they describe equilibria in various systems.

The Galerkin finite element method is based on three important ideas, which are presented in Chapters 2, 3, and 4. The first is that a BVP presented in its classical (“strong”) form can be recast in *weak* or *variational* form, as explained in Chapter 2. The weak form of a BVP is an algebraic formulation of the problem that allows the use of the Galerkin method. The Galerkin method, explained in Chapter 3, is a natural way of projecting the (infinite-dimensional) equation onto a finite-dimensional approximating subspace. The result (for a linear BVP) is a (finite-dimensional) system of linear equations whose solution yields an approximate solution to the BVP. In fact, in a certain sense, the approximate solution is the best possible approximation from the given subspace.

The finite element method is the use of certain approximating subspaces in Galerkin’s method, namely, subspaces of piecewise polynomials. Piecewise polynomials make it (relatively) easy to form and solve the finite element equations. Chapter 4 introduces several spaces of piecewise polynomials that are commonly used in the finite element method. Piecewise polynomials are defined relative to a *mesh* on the computational domain. A mesh partitions the domain into simple subdomains, called *elements*. I will concentrate on triangular elements and two-dimensional domains, although I will describe other possibilities, such as quadrilateral elements.

Chapter 5 outlines the convergence theory for Galerkin finite elements. I present the technical theorems, such as the necessary interpolation theory for piecewise polynomials, and show how they fit into the convergence theory. However, the proofs and detailed development of these techniques are beyond the scope of this book. The purpose of Chapter 5 is to show the reader what to expect from the finite element method.

Part II is about the computer implementation of finite elements. I begin, in Chapter 6, with the strategy for organizing the computations. To make the discussion as concrete as possible, it initially focuses on the common case of piecewise linear functions defined on triangles (linear Lagrange triangles).

The strategy described in Section 6.1 determines the information that must be stored to describe the mesh. The mesh data structure (again, restricted to linear Lagrange triangles) is carefully defined in Section 6.2.

I have chosen to base the codes for this book on MATLAB, an interactive system that integrates numerical and symbolic computations with graphics and a programming language. I chose MATLAB for several reasons:

1. It is a popular tool in the numerical analysis community.
2. It provides state-of-the-art routines for handling sparse matrices; in particular, it is simple to solve a sparse system of linear equations (such as those produced by the finite element method).

3. Its graphical capabilities make it easy to visualize meshes and solutions produced by the finite element method.

However, the main algorithms are presented in an informal pseudocode that is independent of MATLAB. An excellent way for the student to ensure his or her understanding of these algorithms is to translate the pseudocode into some other high-level programming language, such as Fortran, C, or C++.

The MATLAB codes discussed in the text can be downloaded from the following Web page:

<http://www.math.mtu.edu/~msgocken/fembook>

The basic computational algorithms are described in Chapter 7. Section 7.2 shows how to compute the *stiffness matrix*  $K$ , which is the finite-dimensional representation of the partial differential operator defining the PDE. Section 7.3 then shows how to compute the *load vector*  $F$ , which represents the right-hand side of the PDE and any nonzero boundary data. An important part of the discussion concerns incorporating various types of boundary conditions into the computations.

The algorithms from Chapter 7 are extended in Chapter 8 to allow for piecewise polynomials of degree greater than one. Besides allowing for greater accuracy in approximating the solution, higher-order polynomials also make it possible to approximate a computational domain with a curved boundary with *isoparametric* elements. The idea of isoparametric finite elements is first presented in Section 4.7, while the implementation details are explained in Section 8.3.

Chapters 7 and 8 focus on a simple model problem, because most of the essential ideas can be explained in a fairly simple setting. Chapter 9 shows how to extend the techniques to more complicated problems.

Having computed the stiffness matrix and load vector, it remains only to solve the resulting matrix-vector equation and interpret the results. I ignored the issue of solving the system  $KU = F$  in Part II, assuming that the built-in solver in MATLAB would be used. However, direct solvers such as the one in MATLAB can use an unacceptably large amount of time and/or computer memory when the system is large. For this reason, iterative methods are often preferred.

Part III discusses both direct and iterative algorithms for solving a large system like  $KU = F$ . One reason the finite element method is so successful is that piecewise polynomials result in a *sparse* stiffness matrix  $K$ , that is, a matrix in which most of the entries are zero. This makes it possible to solve  $KU = F$  even when the number of unknowns is very large. Chapter 10 gives a brief overview of direct methods for solving sparse systems. A direct method produces the exact solution (up to round-off error) in a finite number of steps. I have included Chapter 10 mainly to provide a context for understanding the advantages of iterative methods; a detailed discussion of direct algorithms is beyond the scope of this book.

Chapters 11 to 13 describe a number of different iterative algorithms, which compute a sequence of approximate solutions that converges to the exact solution. Although the exact solution of  $KU = F$  is computed only in the limit (that is, in an infinite number of steps), a good iterative method can often produce an acceptable solution while using much less time and computer memory than a direct method.

Part III includes chapters on conjugate gradients (Chapter 11), Gauss–Seidel and other classical stationary iterations (Chapter 12), and multigrid algorithms (Chapter 13). Multigrid methods use a sequence of increasingly fine meshes to efficiently estimate the solution of  $KU = F$ . In the best cases, the time required for a multigrid method to produce an accurate solution is proportional to the number of unknowns.

Applying the finite element method requires that a mesh be defined on the computational domain; the accuracy of the computed solution is determined by how well the exact solution can be represented on the chosen mesh. In Part IV, I discuss the various components of an adaptive finite element algorithm, which automatically creates a mesh suited for the problem at hand. Chapter 14 explains algorithms for the local refinement of meshes and a strategy for choosing the elements to be refined. In this chapter, I also present a simple but expensive error estimator and use it to form a complete adaptive algorithm. Several examples show the advantage of the adaptive approach.

Many practical error estimators have been proposed. In the final chapter, I describe three such estimators; two of them are *explicit* and the other is *implicit*. The explicit estimators are inexpensive to compute and indicate which triangles should be refined to reduce the error in the computed solution. However, they do not provide a quantitative measure of the error, and are more properly called error *indicators* rather than error *estimators*. The implicit estimator presented in Section 15.3 is somewhat more expensive but acts as a true estimator. Not only does it indicate where the mesh should be refined, but it also gives an accurate measure of the size of the error.

Section 15.4 contains several examples of problems with singular solutions. Adaptive finite element methods are particularly effective on such problems.

Exercises are provided at the end of each chapter. Some of these are theoretical, some ask the student to apply the code provided with the text, and others require programming to extend the capabilities of the code. Beyond the given exercises, there is probably no better way to understand the finite element method than to rewrite the MATLAB codes in another programming language. In the process of translating the code into a different syntax, and particularly in testing and debugging, the details must be mastered.

The bibliography lists the books and papers that I used directly in writing this manuscript. Babuska and Strouboulis [8] provide extensive reference lists with detailed bibliographical comments.

**Acknowledgments.** I would like to thank the staff at SIAM, particularly my editors Alexa Epstein and Elizabeth Greenspan, for their roles in producing this book. The anonymous reviewers contributed many helpful suggestions, and the final form of the book owes much to their comments. In addition, Dr. A. A. Khan and Dr. B. Jadamba read the entire manuscript carefully and found a number of errors. Their help is gratefully acknowledged.

Finally, I am pleased to dedicate this book to my wife Joan and to all of our children: Mary, Mark Jr., Kenric, Lydia, Hope, Nate, Jack, and the newest Gockenbachs, whose arrival is eagerly awaited.

Mark S. Gockenbach  
msgocken@mtu.edu