Computing with

hp-ADAPTIVE FINITE ELEMENTS

I. One- and Two-Dimensional Elliptic and Maxwell Problems

Leszek Demkowicz

Institute for
Computational Engineering and Sciences
The University of Texas at Austin

Austin, Summer 2005

To my Beloved Wife, Stasia

Preface

This book reports a very personal experience and struggle with the subject of hp elements and, I hope, the reader will not get offended if I make the presentation very personal from the very beginning.

Implementing hp elements. Shortly, after my third arrival in Texas, J. Tinsley Oden, my host, boss and friend, presented me with the task of writing an adaptive hp Finite Element (FE) code. The project was motivated with the fundamental results of Babuška and Gui [75], and Babuška and Guo [76] who showed that, for a large class of one-dimensional (1D), and two-dimensional (2D) elliptic problems, one can construct the so called *geometrically graded meshes* that deliver *exponential convergence* - the discretization error is decreasing exponentially with the problem size, i.e. the number of degrees-of-freedom (d.o.f.).

This was a real breakthrough in the subject of FE discretizations. The rate of classical, uniform h-convergence is a function of the polynomial degree p, but it is also limited by the regularity of the solution. In presence of singularities in the (derivatives of the) solution, the convergence rates are limited by the regularity of the solution expressed in terms of Sobolev norms. When I began to study finite elements in late seventies, the consensus was, see e.g. [32], that one should not use elements of higher order for problems with singular solutions at all. This changed with the arrival of h-adaptive methods. In another fundamental contribution, Babuška, Kellogg, and Pitkäranta [19] demonstrated that the h-adaptivity restores always the optimal rate of convergence implied by the order of approximation alone, comp. also [52].

The idea of hp-discretizations was rooted in the pioneering work of Barna Szabo and his students and collaborators at Washington University, St. Louis, on the p-methods, where the convergence is achieved by raising uniformly or adaptively polynomial order p, see [117]. It was already known that the p method, being a representative of a larger class of *spectral finite elements* could deliver exponential convergence for problems with solutions that are analytic up to boundary, i.e. in a larger domain enclosing the domain of computations. The problem is that, except for problems defined on unit cubes with periodic boundary conditions, such regularity does not occur in practice. The solutions to practical elliptic problems may experience singularities at corners and material interfaces. True, Babuška and Vogelius [120] demonstrated that, in presence of singularities, the

p-convergence rates were twice of those for the h-convergence (in terms of the number of d.o.f.). At the same time, however, the p-method produces fully populated matrices, and it was not clear at all that it would pay off to invest into a p-adaptive code.

We were not new to adaptive methods. During my previous stay in Texas, Tinsley Oden and myself, with two of Oden's students - Fanis Strouboulis and Philippe Devloo, had produced already a significant number of results (and codes) on h- p- or r-adaptive methods (r stands for *relocation* of nodes, and r-adaptivity referred to moving mesh strategies)). I had reproduced most of our results on adaptive methods and a-posteriori error estimation in my habilitation thesis [36] and continued working on adaptive methods in Poland. Nevertheless, we had never tried to put together two kinds of adaptivity in one code. In fact, I should have said, I had not tried. The first, limited ($p = 1, 2, 3$) version of an hp-adaptive code was put together by Philippe Devloo in 1997, shortly before he had left Texas. Philippe continued working on the subject and developed later a remarkable, general purpose hp code [62].

In my quest for the new code, I was aided by Waldek Rachowicz - my colleague and collaborator from Cracow, who by then was already for one year at Texas, studying for a Ph.D. degree under Tinsley's supervision. It took us almost 7 months to put together the first 2D hp-code, and then another 8 months to build a 3D version. The results were summarized in the series of three papers [54, 97, 105]. Besides Tinsley, Waldek and myself, the team included also Tim Westerman and Olivier Hardy. The bulk of coding was done by Waldek Rachowicz who implemented the very first version of *constrained approximation*, and also had invented and implemented the first, fully automatic hp-adaptive strategies, see also [102]. The codes were rather limited, they were based on quads (2D) and hexahedra (3D), supported only *isotropic* h- and p-refinements, geometry was handled using only quadratic shape functions (subparametric elements), and the initial mesh was assumed to be a subset of a structured grid. The codes were developed on IBM AT PC's.

The 3D code was later a starting point for a general hp kernel PHLEX[1], supporting both isotropic and anisotropic mesh refinements for general unstructured 3D meshes consisting of hexahedra, prisms and tetrahedra, developed at COMCO (Computational Mechanics Company) in Austin. The 2D and 3D hp codes developed at TICOM, laid down a foundation for an extensive research on hp methods led by Tinsley Oden on, among other topics, compressible flow [53, 55], incompressible flow [96], and transient acoustics [110].

Almost immediately upon my return to Cracow in 1990, I set to develop a new version of the codes. Motivated with two classes of applications: modeling of vibrations of elastic structures immersed in an acoustical fluid, and supersonic flow around an aircraft, I decided to focus this time on triangular elements in 2D and prismatic elements in 3D. The 2D code supported a fully unstructured initial grid (h-refinements produce always locally structured grids) but the 3D code was only '$2\frac{1}{2}$- dimensional', the initial grid was obtained by combining an unstructured 2D triangular

[1] PHLEX is a trademark of Altair Engineering

grid with a structured grid in the third direction. The design assumptions will look less restrictive if you know that the codes were developed on Sony workstations with 12 MB memory only. The codes supported, however, anisotropic refinements and isoparametric elements. We also built the first version of our Geometrical Modeling Package (GMP) to enable mesh generation, and automatic geometry updates during mesh refinements. In my work at Cracow, I was helped by my friend - Andrzej Karafiat, and two of my students - Krzysztof Banaś and Andrzej Bajer. The work was done partially at Cracow and partially at Austin, during our Summer visits.

In 1993, I moved back to Austin and continued working on the acoustics code, with the help of two new students - Yao Chang Chang and Klaus Gerdes. The second versions of the hp codes died in the end of 1995, with the termination of Navy project. By then I was joined by Andrzej Bajer and Timothy Walsh who persuaded me to switch to Fortran 90. We wrote a new version of the 2D code, this time for both triangles and quads, and implemented a third version of *constrained approximation* [46, 61]. The 2D code laid down a foundation for Timothy's Ph.D. work on modeling of the human auditory system using adaptive hp boundary elements, and Andrzej's dissertation on modeling dynamics of planetary gears. In 1999 we completed my third implementation of a 3D code, based on isoparametric hexahedral elements and supporting arbitrary anisotropic refinements [44].

In the course of NPACI (National Partnership for Computational Infrastructure) project we attempted to parallelize the code and we miserably failed - the complexity of the workstation version was overwhelming. But it was the failed attempt of parallelization and an experience within the NPACI community that helped me to understand my previous mistakes, and design a completely new hp data structure. In collaboration with Waldek Rachowicz and help of my student - David Pardo, we completed our most recent (and my fourth) implementation of the 3D code [57]. The code has the same functionality as its last predecessor but, due to the new data structure design, the complexity level went significantly down. It was also for the very first time that I rewrote the 2D code [40], and the 1D code presented in this monograph, *after* having completed the 3D code. The 2D code supports both triangles and quads. All three codes share *identical* data structures which allows for studying the whole methodology from the easiest - 1D case, through moderate difficulty of the 2D implementation, to the challenge of the 3D coding.

Existing hp finite element codes. Several commercial codes claiming hp-adaptivity are in fact, p-adaptive codes that combine an optimal initial mesh design with optimal or even uniform p-refinements. If the mesh is correct, the p-method exhibits a quasi-exponential behavior in the preasymptotic region and delivers an accuracy superior to standard h codes. To this group be-

long commercial codes PolyFEM[2], Pro/MECHANICA [3], and StressCheck [4], the code developed by Barna Szabo and his collaborators.

The only commercial code that supports simultaneously h- and p-refinements, and approximation on irregular meshes, I am aware of, is PHLEX [5], the code developed at COMCO, and now owned by Altair Engineering. The PHLEX kernel, enabling hp-refinements, has served as a starting point to develop commercial and research software, among other applications, for metal/polymer extrusion (HyperXtrude), structural analysis (PHLEXsolid), dynamic fracture propagation (PHLEXcrack), flow in porous media, tire dynamics, and recently, FESTA - a structural acoustics code being developed by Dave Burnett at NATO.

A number of 2D and 3D p and hp codes have been developed in research organizations and in Academia. One of the earliest p codes is STRIPE - the code developed by Borje Anderson at the Aeronautical Institute of Sweden. Similarly to Szabo, Anderson combines a judicious mesh design with uniform p-refinements, to achieve a quasi- exponential convergence in the preasymptotic range. Over the years, STRIPE has been a basis for multiple research projects, including aircraft stress/fracture analysis [8, 6, 10], structural stability [67], crack/fracture analysis in metals [24, 9], and in composites [7, 112], as well statistical analysis [94, 68], and micromechanics [15, 11].

I have already mentioned the work of Philippe Devloo [62].

An original 3D hp code, written within the multilevel KASKADE [6] framework was written by Gerhard Zumbusch [125]. The code is based on regular tetrahedral meshes, and it was applied to solve a number of 3D elliptic model problems, and linear elasticity examples with complex geometries and singular solutions.

The main authority in applying the hp methods to Incompressible and Compressible Flow problems is George Karniadakis from Brown University, see [85], and the references therein. Karniadakis' implementation is centered around the idea of generating shape functions for triangles (2D), prisms and tetrahedra (3D), by collapsing the quad or hexahedral element using Duffy's type transformations. Remarkably, the degenerated element preserves polynomial spaces, and the tensor product structure of shape functions, which allows for an efficient integration of element matrices. Karniadakis and his collaborators developed a general code called NEKTAR for deterministic problems (continuous and discontinous Galerkin with Jacobi polynomials used for shape functions), expanded later to a stochastic simulation code - sNEKTAR, based on generalized polynomial chaos expansions. The code also gave rise to a separate program for microfluidics - μFLOW, see [84], and the references therein.

[2] PolyFEM is a trademark of LMS International

[3] Pro/MECHANICA is a trademark of Parametric Technology Corporation

[4] StressCheck is a trademark of Engineering Software Research & Development, Inc., 10845 Olive Blvd. Suite 170, St. Louis, Missouri 63141

[5] PHLEX is a trademark of Altair Engineering

[6] Developed at Konrad Zuse Centrum in Berlin by a team led by Prof. Peter Deuflhard

The recent dissertation of Düster is focused essentially on the p- method, with applications to nonlinear analysis of thin-walled structures [66]. He also has further developed the original $hp-d$ method of Rank [107], enhancing 2D computations with an overlapping local hp mesh.

The finite element framework *Trellis*, developed by Joe Flaherty and Mark Shepard at Renselier, is capable of hp-adaptivity, see e.g. [63, 116, 108]. More recently, Cugnon [35] wrote also a Ph.D. thesis centered around a 3D hp code. A general 3D hp code supporting continuous approximation on irregular meshes was developed by Christian Lage, is in use in the group of Chris Schwab at TH Zurich, being further developed by Philipp Frauenfelder [69].

hp Edge elements and de Rham diagram. In Summer of 1996 I started to study electromagnetics and a year later, Leon Vardapetyan and I came up with a generalization of Nedelec's edge elements [92, 93] to the case of variable order of approximation [60, 119]. Peter Monk introduced us to the concept of de Rham diagram, and we were able to construct the whole family of H^1- $H(\mathrm{curl})$- and $H(\mathrm{div})$-conforming hp-elements of variable order opening door for hp-adaptivity for Maxwell equations. Waldek Rachowicz joined us in Texas in 1997, and we began to work on 2D [103], and later 3D [104] hp-codes for electromagnetics. Later, Witek Cecot added to the code an infinite element enabling solution of exterior problems [28, 29]. Upon his return to Cracow, Waldek began collaborating with Adam Zdunek from the Swedish Aeronautical Institute.

The codes were written by modifying our existing hp codes for elliptic problems. Three years ago, we decided to use the C-preprocessor and expanded our newest $2Dhp$ code to include the electromagnetics. This is the version presented in this monograph. The 3D work on automatic hp-adaptivity for Maxwell equations within the new data structure, is still in progress. To my best knowledge, our codes are the only existing finite element codes for Maxwell equations, supporting hp-refinements on irregular meshes.

Recently, Paul Ledger [89] developed a two-dimensional hp code using hybrid quadrilateral and triangular hp edge elements, with applications to 2D Maxwell scattering problems.

Automatic hp-adaptivity. The main goal of research presented in this monograph, has been to work out a *fully automatic* hp-adaptive strategy that would deliver meshes with a minimum number of degrees of freedom (d.o.f.) in the full range of error level, especially in the preasymptotic range.

This is a highly nonlinear and non-convex problem. Ideally, we would love to have a mesh optimality condition expressed in terms of local mesh size h and order of approximation p. After all, algorithms based on optimality conditions are among the most efficient methods in optimization. For h-adaptive methods, such optimality criteria are based on the notion of *mesh density functions*, and lead to the error equidistribution principle [52]. Examples of such efforts can be found recently, e.g. in [95, 101].

As mentioned above, most of the commercial implementations rely on *a-priori* information

about corner and edge singularities [18, 113], or boundary layers, see [113, 91], and begin the solution process with a generation of judiciously constructed initial meshes like the geometrically graded meshes of Babuška, or Shiskin type meshes for handling the boundary layers. Once the mesh is known, uniform p refinements are made. In more sophisticated implementations, adaptive p-refinements are used. If the nature of the singularity is known a priori, these techniques are very effective and difficult to beat.

The situation becomes less clear, if the regularity results are not at hand. An unoptimal initial mesh, followed by p-refinements, may lead to meshes that deliver results worse than than standard h-adaptive methods. Years ago I heard from Prof. Olek Zienkiewicz that, for error levels 1-5 percent (measured in energy norm relative to the norm of the solution), the h-adaptive meshes of quadratic elements are the best, and there is little need for any extra sophistication. Indeed, for many practical problems, the quadratic elements offer probably the best balance between the complexity of coding (one d.o.f. per node, no need for orientation of nodes) and the results they can deliver. Attempts to correct an inefficient initial mesh through h refinements, have led to methods that perform h and p-refinements interchangeably, see e.g. the *Texas Three Step* strategy [100, 99, 50]. The resulting meshes are easy to obtain but, in general, do not lead to optimal results.

A genuine hp strategy was proposed in [5], where the choice between the h and p-refinement is based on monitoring local h-convergence rates. The methodology has recently been applied to solve hyperbolic problems in [79], in context of *discontinuous* hp discretizations.

The strategy presented in this monograph follows closely the original idea presented in [105], where the optimal mesh is obtained by minimizing a local projection error for a reference solution. The method has been recently generalized to multidimensions [58]. The main idea is centered around the notion of projection-based interpolation introduced in [97], and expanded to $H(curl)$ and $H(div)$ spaces in [49, 43]. The projection-based interpolation and the corresponding hp- adaptive strategy based on minimizing the interpolation error, are the leading topics discussed in this monograph.

Scope of this volume. The presented volume summarizes our work on 1D and 2D codes, and automatic hp adaptivity. The work on the second volume, focusing on three-dimensional elliptic and Maxwell problems, is still in progress and hopefully will be completed in a finite time. The first half of the volume is an outgrowth of class notes and code manuals prepared for teaching EM394F - an introductory, graduate level class on Finite Elements taught in the Department of Aerospace Engineering and Engineering Mechanics (ASE/EM) at the University of Texas at Austin. The main challenge in teaching the class is its diversified audience, consisting of graduate students coming from all Engineering departments: Aerospace, Civil, Electrical, Mechanical, Chemical and Petroleum Engineering, Engineering Mechanics, many students from our Computational and Applied Mathematics (CAM) program, and occasionally also students of Computer Science of Mathematics. The

class is also open (with the instructor's consensus) to senior graduate students. It may sound a little bit shocking but I do teach the hp elements from the very beginning without covering first the classical, lower order version of finite elements based on Lagrange shape functions. Earlier attempts to introduce the hierarchical shape functions and adaptivity more gradually, had failed. Majority of the students had trouble switching from the classical setting (e.g. getting used to the notion of geometry d.o.f. that cannot be interpreted as nodal coordinates). I do not claim a 100 percent survival rate, though. Despite my efforts, 10-20 percents of students drop the class in the first month.

When teaching the class, I focus on understanding the basic linear structure of the methodology, coding details, and hands on experience. I do not shy from more advanced mathematics completely, though. The CAM students who typically constitute one third of the audience, and who take this class *after* the EM394F - a one semester course on Functional Analysis that I also teach, are well prepared to handle the beginning mathematical theory of finite elements. It is to this group of students that I address all sections with *italicized titles*, and make frequent references to our text on Functional Analysis [98]. Besides the coding assignments found in this text, I give two exams during the semester. I usually prepare two versions of the exams based on the exercises given in the text, with a more mathematically challenging set of problems for the CAM students.

The one semester course stops usually at the end of Chapter 12 explaining the geometric modeling and mesh generation. Students get to use the adaptivity in the 2D code but there is no time left to teach them the coding details. That part of the book, as well as the exposition on the Maxwell equations is based on a large number of seminars and 'group meetings' with my own graduate students, and has been used in a class only once in Summer 2005 when I taught an introductory course of Finite Element Methods for Maxwell equations. I have maintained the same format, in hope that I may use the book for teaching a second semester continuation of the class on a more regular basis. I do not anticipate the same style for the second volume which will be addressed to a more advanced audience.

And now a few words about the scope. The first part is devoted to studying the hp elements for the standard 1D model elliptic problem. We develop the variational formulation and, after discussing the general Galerkin method, we explain the construction of FE basis functions. This part ends with the first assignment, 'a dry run', where students are asked to perform all typical FE operations on paper. We next introduce $1Dhp$ - our 1D code, discuss the coding, convergence, adaptivity and, in the end, the automatic hp-adaptivity. Two new assignments follow. Reading this part of the book has to be done with a simultaneous study of the code. We have done a considerable effort to follow a rigorous coding protocol and equip the code with extensive, in-code comments. The hands on experience is absolutely crucial in understanding my comments and explanations. We finish the first part of the book with a study of a 1D wave propagation problem. Much of the 1D presentation may seem to be over-sophisticated as many introduced concepts anticipate the 2D implementation and only in the 2D (and especially 3D) are fully justified.

In part two, we focus on 2D elliptic problems. We follow the same order of presentation as in 1D, focusing first on regular meshes. We delegate a separate chapter to geometric modeling and mesh generation based on the projection-based interpolation. Finally, we discuss the adaptivity. We include in the second part a discussion of two representative model problems that go slightly beyond standard examples - a 3D axisymmetric antenna problem for Maxwell equations - an example of complex-valued, indefinite problem; and the 2D elasticity - as an example of an elliptic system. Finally, we conclude Part 2 of the book with a presentation on infinite elements - one of possible tools to solve exterior boundary-value problems.

In the third part we focus on 2D time-harmonic Maxwell equations. As most of my audience has had a little exposure to Maxwell equations, I begin with a short derivation of those. Next we explain the construction of the hp edge elements and the fundamental de Rham diagram for the whole family of the hp-discretizations. We then discuss the differences between the elliptic and Maxwell versions of the 2D code, including automatic hp-adaptivity. Finally, we conclude our presentation with the discussion of 2D exterior (radiation and scattering) problems and present sample solutions using coupled hp finite/infinite elements.

Acknowledgments. This work would never have been done without the guidance of my life time mentor and friend - Prof. J. Tinsley Oden. I started to read his books and papers when I began to work on my Ph.D. dissertation, and the possibility of working with him was the main motivation of my first coming to Texas in 1980. Tinsley introduced me to adaptivity and hp-elements and we had worked very closely together on the subject for more than a decade. His continuous support and friendship are my personal treasures.

I have been studying Prof. Ivo Babuška's papers, ever since I was an undergraduate student. He was the first to introduce the ideas of adaptivity, *a-posteriori* error estimation and, of course, the hp-adaptivity. Ever since Ivo moved to Austin in 1994, I had a privilege of invading his office an infinite number of times asking for a discussion or advice. I will never stop learning from him for which I am very grateful.

Half of my research on hp methods was done with Waldek Rachowicz. Without his ingenuity, patience, and care for details (that I miss), most of our results would not have been possible. Much of the presented work was done in collaboration with several of my colleagues: Andrzej Karafiat, Mark Ainsworth, Peter Monk, Witek Cecot, Philippe Devloo, Adam Zdunek, Pavel Solin and Maciek Paszyński (last two were my postdoctoral students).

I have been blessed with excellent graduate students: Krzysztof Banaš, Yao Chang Chang, Klaus Gerdes, Leon Vardapetyan, Timothy Walsh, Andrzej Bajer, David Pardo, Dong (Cynthia) Xue and M.S. students: Michele Pal, Chang-wan Kim, Ute Gauger, Irina Mardanova, Yufeng Zhang and Chad Larson. My recent and present graduate students: David Pardo, Dong Xue, and Jason Kurtz share the credit for all the results accomplished in the last few years.

Over the years, my work on hp methods has been supported by a number of agencies: Office of Naval Research, Polish Komitet Badań Naukowych, National Science Foundation and Air Force Office of Scientific Research. Special thanks go to Dr. Arje Nachman of AFOSR for his continuous support of my work on Maxwell equations.

The book is dedicated to my wife - Stanislawa Demkowicz. Without her love and support, I would have accomplished nothing.

Contents

I ONE-DIMENSIONAL PROBLEMS **17**

1 1D Model Elliptic Problem **19**

1.1 A Two-Point Boundary Value Problem 19

1.2 Algebraic structure of the variational formulation 23

1.3 Equivalence with a minimization problem 25

1.4 Sobolev space $H^1(0, l)$ 27

1.5 *Well posedness of the variational BVP* 31

1.6 Examples from mechanics and physics 33

1.6.1 Elastic bar 33

1.6.2 Heat conduction 37

1.6.3 Vibrations of an elastic bar 38

2 Galerkin Method **43**

2.1 Finite dimensional approximation of the VBVP 43

2.2 Elementary convergence analysis 46

2.3 Comments 48

3 1D hp Finite Element Method **51**

3.1 One-dimensional hp discretization 51

3.2 Assembling element matrices into global matrices 55

3.3 Computing the element matrices 58

3.4 Accounting for the Dirichlet BC 61

3.5 A summary 62

3.6 Assignment 1: A dry run 62

4 *1Dhp* Code **65**

4.1 Setting up the $1Dhp$ code 65

4.2 Fundamentals 67

4.3 Graphics 68

4.4 Element routine 72

4.5 Assignment 2: Writing your own processor 74

5 Mesh Refinements in 1D **79**

5.1 The h-extension operator. Constrained approximation coefficients 79

5.2 Projection based interpolation in 1D 81

5.3 Supporting mesh refinements 84

5.4 Data structure supporting routines 84

5.4.1 Natural order of elements 85

5.4.2 Reconstructing element nodal connectivities 86

5.5 Programming bells and whistles 87

5.5.1 Interfacing with the frontal solver 87

5.5.2 Adaptive integration 89

5.5.3 Choice of shape functions 90

5.6 Interpolation error estimates 90

5.7 Convergence 94

5.8 Assignment 3: Studying convergence 97

5.9 *Definition of a finite element* 99

6 Automatic hp-Adaptivity in 1D **101**

6.1 The hp algorithm 101

6.2 Supporting the optimal mesh selection 104

6.3 Exponential convergence. Comparing with h-adaptivity 107

6.4 Discussion of the hp algorithm 109

6.5 Algebraic complexity and reliability of the algorithm 110

6.5.1 Error estimation. 111

7 Wave Propagation Problems **119**

7.1 *Convergence analysis for non coercive problems* 119
7.2 *Wave propagation problems* 123
7.3 Asymptotic optimality of the Galerkin method 127
7.4 Dispersion error analysis 129

II TWO-DIMENSIONAL ELLIPTIC PROBLEMS 135

8 2D Elliptic Boundary-Value Problem 137
8.1 Classical formulation 137
8.2 Variational (weak) formulation 141
8.3 Algebraic structure of the variational formulation 144
8.4 Equivalence with a minimization problem 145
8.5 Examples from Mechanics and Physics 146

9 Sobolev Spaces 149
9.1 Sobolev space $H^1(\Omega)$ 149
9.2 *Sobolev spaces of an arbitrary order* 151
9.3 *Trace Theorem* 153
9.4 *Embedding Theorems* 153
9.5 *Well posedness of the variational BVP* 153

10 2D hp Finite Element Method on Regular Meshes 155
10.1 Quadrilateral master element 156
10.2 Triangular master element 158
10.3 Parametric element 160
10.4 Finite element space. Construction of basis functions 161
10.5 Calculation of element matrices 164
10.6 Modified element. Imposing Dirichlet boundary conditions 168
10.7 Postprocessing. Local access to element d.o.f. 171
10.8 Projection-based interpolation 172

11 $2Dhp$ Code 177

11.1 Getting started 177
11.2 Data structure in FORTRAN 90 179
11.3 Fundamentals 180
11.4 The element routine 181
11.5 Modified element. Imposing Dirichlet boundary conditions. 182
11.6 Assignment 4: Assembling of global matrices 183

12 Geometric Modeling and Mesh Generation 187

12.1 Manifold representation 187
12.2 Construction of compatible parametrizations 189
12.3 Implicit parametrization of a rectangle 193
12.4 Initial mesh generation 195

13 The hp Finite Element Method on h-Refined Meshes 199

13.1 Introduction. The h-refinements 199
13.2 1-Irregular Mesh Refinement Algorithm 200
13.3 Data structure in Fortran 90 - continued. 204
13.4 Constrained approximation for C^0 discretizations 207
13.5 Reconstructing element nodal connectivities. 210
13.6 Determining neighbors for mid-edge nodes 212
13.7 Additional comments 213

14 Automatic hp-Adaptivity in 2D 217

14.1 The main idea 217
14.2 Discussion of one step of the algorithm 218
14.3 Example: L-shape domain problem. 221
14.4 Additional remarks 223

15 Examples of Applications 225

15.1 Linear elasticity 225
15.2 An axisymmetric Maxwell problem 231

16 Exterior Boundary-Value Problems 239

16.1 Variational formulation. Infinite element discretization 240
16.2 Selection of IE radial shape functions 245
16.3 Implementaion 246
16.4 Calculation of Echo Area 249
16.5 Numerical experiments 250
16.6 Comments 257

III TWO-DIMENSIONAL MAXWELL PROBLEMS 261

17 Two-Dimensional Maxwell Equations 263
17.1 Introduction to Maxwell's equations 264
17.2 Variational formulation 267
17.2.1 Non-dimensionalization of Maxwell's equations 271

18 Edge Elements and de Rham Diagram 277
18.1 Exact sequences 278
18.1.1 Nédélec's Triangular Element of the Second Type 279
18.1.2 Nédélec's Rectangular Element of the First Type 280
18.1.3 Nédélec's Triangular Elements of the First Type 281
18.1.4 Parametric elements 283
18.2 Projection-based interpolation 285
18.3 De Rham diagram 289
18.4 Shape functions 291

19 2D Maxwell Code 297
19.1 Directories. Data structure 298
19.2 The element routine 299
19.3 Constrained approximation. Modified element 302
19.4 Setting up a Maxwell problem 305
19.4.1 Example: scattering of a plane wave on a screen 305

20 hp-Adaptivity for Maxwell Equations 311

20.1 Projection-based interpolation revisited 312
20.2 The hp mesh optimization algorithm 315
20.3 Example: The screen problem 320

21 Exterior Maxwell Boundary-Value Problems **327**
21.1 Variational formulation 328
21.2 Infinite Element discretization in 3D 329
21.3 Infinite Element discretization in 2D 333
21.4 Stability 334
21.5 Implementation 336
21.6 Numerical experiments 338

Part I

ONE-DIMENSIONAL PROBLEMS

Chapter 1

1D Model Elliptic Problem

It is customary to start any presentation on finite elements with one-dimensional elliptic problems. In this opening chapter, we will study a model two-point elliptic boundary-value problem for a linear differential equation of second order. We shall discuss the equivalence of classical and variational formulations, foundations of the Galerkin method, and then follow in the next chapters with a development of a 1D hp code. Much of the introduced formalism will seem to be over-sophisticated as it is motivated with the anticipated development of related ideas in two and three space dimensions.

1.1 A Two-Point Boundary Value Problem

Classical formulation. We wish to find a solution $u(x), x \in [0, l]$, of a second order, linear differential equation,

$$-(a(x)u(x)')' + b(x)u'(x) + c(x)u(x) = f(x), \quad x \in (0, l), \tag{1.1.1}$$

accompanied at each of the endpoints $x = 0$ or $x = l$ with one of three possible boundary conditions:

- Dirichlet boundary condition (BC),

$$u(0) = u_0 \quad or \quad u(l) = u_l, \tag{1.1.2}$$

- Neumann BC,

$$-a(0)u'(0) = \gamma_0 \quad or \quad a(l)u'(l) = \gamma_l, \tag{1.1.3}$$

- Cauchy BC,

$$-a(0)u'(0) + \beta_0 u(0) = \gamma_0 \quad or \quad a(l)u'(l) + \beta_l u(l) = \gamma_l. \tag{1.1.4}$$

The Neumann boundary condition is just a special case of Cauchy boundary condition with constant $\beta = 0$.

The data to the problem consist of

- the *material data:* coefficients of the differential operator $a(x), b(x), c(x)$, coefficients of the Cauchy boundary operator β_0, β_l, and
- the *load data:* source term $f(x)$ and boundary data $u_0, u_l, \gamma_0, \gamma_l$.

For the sake of this presentation, we shall select the Dirichlet boundary condition at $x = 0$ and Cauchy boundary condition at $x = l$,

$$u(0) = u_0, \quad a(l)u'(l) + \beta u(l) = \gamma \, .$$

The discussion of the remaining eight different combinations of BC's follows (almost) identical steps.

Typical regularity assumptions for the data in the classical formulation are:

$$a \in C^1[0, l], \quad b, c, f \in C(0, l) \, .$$

Under a few additional assumptions (e.g. coefficient $a(x)$ should not degenerate, $a(x) > a_{min} > 0$), the classical Ordinary Differential Equations (ODE) theory implies existence (and possibly uniqueness) of a solution $u \in C^2(0, l) \cap C^1[0, l]$.

Interface problem. In practice, the material constants are only piecewise regular, frequently just piecewise constant. At the discontinuity points second derivatives do not exist (in the classical sense), the classical formulation loses its validity, and has to be replaced with an interface problem formulation. For instance, if

$$a(x) = \begin{cases} a_1(x), & 0 \le x \le x_0 \\ a_2(x), & x_0 \le x \le l \end{cases} \, ,$$

we anticipate also a two-branch solution,

$$u(x) = \begin{cases} u_1(x), & 0 \le x \le x_0 \\ u_2(x), & x_0 \le x \le l \end{cases} \, ,$$

and formulate the problem as follows.

- The original equation, to be solved in the two subdomains:

$$-(a_1(x)u_1(x)')' + b_1(x)u_1'(x) + c_1(x)u_1(x) = f_1(x) \quad x \in (0, x_0) \, ,$$
$$-(a_2(x)u_2(x)')' + b_2(x)u_2'(x) + c_2(x)u_2(x) = f_2(x) \quad x \in (x_0, l) \, .$$

- The original boundary conditions:

$$u_1(0) = u_0, \quad a(l)u'(l) + \beta u(l) = \gamma \, .$$

- The interface conditions at x_0:

$$u_1(x_0) = u_2(x_0), \quad a_1(x_0)u_1'(x_0) = a_2(x_0)u_2'(x_0) \, .$$

The interface conditions are encountered in practical applications. Due to a physical interpretation, the product au', will be called the *flux*. The second interface condition requires a continuity of the flux and, in presence of discontinuous material coefficient $a(x)$, implies a corresponding discontinuity of the first derivative. Consequently, at the interface point, the second derivative does not exist.

The interface problem can be generalized to an arbitrary number of interfaces.

Weak (variational) formulation of the problem. Let u be now a classical solution of the original problem. We employ an arbitrary *test function* $v(x)$, multiply equation (1.1.1) pointwise by $v(x)$, and integrate over interval $(0, l)$ to obtain,

$$\int_0^l -(au')'v \, dx + \int_0^l bu'v \, dx + \int_0^l cuv \, dx = \int_0^l fv \, dx \, .$$

Integrating the first term by parts, we get:

$$\int_0^l -(au')'v \, dx = \int_0^l au'v' \, dx - a(l)u'(l)v(l) + a(0)u'(0)v(0) \, .$$

We use now a different strategy at the two endpoints. At $x = l$, we utilize the Cauchy boundary condition and replace the flux term with

$$-a(l)u'(l)v(l) = \beta u(l)v(l) - \gamma v(l) \, .$$

At $x = 0$, as no information about the flux is available, we choose *to not test* the equation by assuming an extra condition for the test function: $v(0) = 0$. Keeping the terms involving the unknown solution on the left, and moving the known data to the right, we arrive at the formulation:

$$\begin{cases} u(0) = u_0 \\ \displaystyle\int_0^l \left\{ au'v' + bu'v + cuv \right\} \, dx + \beta u(l)v(l) = \int_0^l fv \, dx + \gamma v(l) \\ \quad \text{for every test function } v, v(0) = 0 \, . \end{cases} \tag{1.1.5}$$

This is the *Variational (Weak) Formulation* of the BVP (VBVP). Notice that we have simply rewritten the Dirichlet BC.

The interface problem yields exactly the same formulation. Multiplying the original equation with v_1 in $(0, x_0)$, and with v_2 in (x_0, l), integrating over the two subintervals, and summing up the contributions, we get:

$$\begin{aligned}
&\int_0^{x_0} -(a_1 u_1')' v_1 \, dx + \int_{x_0}^{l} -(a_2 u_2')' v_2 \, dx \\
&= \int_0^{x_0} a_1 u_1' v_1' \, dx + \int_{x_0}^{l} a_2 u_2' v_2' \, dx \\
&-a_2(l) u'(l) v_2(l) + \left[a_2(x_0) u_2'(x_0) - a_1(x_0) u_1'(x_0)\right] v(x_0) + a_1(0) u_1'(0) v_1(0) \\
&= \int_0^{l} a u' v' \, dx + \beta u(l) v(l) - \gamma v(l) \, .
\end{aligned}$$

In the above, we have assumed that the test function $v(x)$, similarly to the solution, is continuous at the interface, $v_1(x_0) = v_2(x_0)$. In the last line, we have combined the integrals over the two subintervals into one integral, and dropped the index notation for the branches.

Thus, solutions to both the original and the interface problem, satisfy the same variational identity. We shall show shortly that appropriate regularity assumptions for the solution and the test functions imply automatically global continuity, i.e. there is no need to assume the continuity at the interface explicitly.

Apparently, when constructing the VBVP we have not lost any information about the solution. We will show now that, conversely, any sufficiently regular solution to the variational problem, is also the classical solution. We shall discuss the more general interface problem only.

Let $u(x)$, represented with branches u_1, u_2, be a solution to variational problem (1.1.5). Integrating the weak formulation by parts over each subinterval separately and collecting the subinterval, boundary and interface terms, we obtain:

$$\begin{aligned}
&\int_0^{x_0} \{-(a_1 u_1')' + b_1 u_1' + c_1 u_1 - f_1\} v_1 \, dx + \int_{x_0}^{l} \{-(a_2 u_2')' + b_2 u_2' + c_2 u_2 - f_2\} v_2 \, dx \\
&+\{a_2(l) u_2'(l) + \beta u_2(l) - \gamma\} v_2(l) + \left[-a_2(x_0) u_2'(x_0) + a_1(x_0) u_1'(x_0)\right] v(x_0) = 0 \, .
\end{aligned} \tag{1.1.6}$$

We need to recall now the classical Fourier's lemma.

THEOREM 1

(Fourier's lemma)

Let f be a continuous function defined on $(0, l)$ such that

$$\int_0^{l} f(x) v(x) \, dx = 0 \quad ,$$

for every continuous test function v that vanishes at the endpoints, $v(0) = v(l) = 0$.

Then f must identically vanish, $f = 0$. ∎

Notice that dropping the condition on test functions to vanish on the boundary would make the theorem *weaker*. It is sufficient to test with functions which vanish on the boundary to conclude that tested function f (we assume no BC for f) must be identically equal zero.

We now proceed in two steps.

Recovering differential equations. We first test with test functions v vanishing outside of subinterval $(0, x_0)$, including the interface point. We use Fourier's lemma to *recover the differential equation* in the first subinterval

$$-(a_1(x)u_1(x)')' + b_1(x)u_1'(x) + c_1(x)u_1(x) - f_1(x) = 0 \quad x \in (0, x_0) \, .$$

We use the same argument to recover the differential equation in the second subinterval.

Recovering interface and boundary conditions. Once we know that the integrands corresponding to the two integrals in (1.1.6) are zero, the corresponding integral terms must vanish identically for *all test functions* (not only for the special functions used in the previous step), and (1.1.6) reduces to:

$$\{a_2(l)u_2'(l) + \beta u_2(l) - \gamma\}v_2(l) + \left[-a_2(x_0)u_2'(x_0) + a_1(x_0)u_1'(x_0)\right] v(x_0) = 0 \, .$$

Testing with test function v such that $v(x_0) = 1, v(l) = 0$, we *recover the interface condition*, and testing with test function v such that $v(x_0) = 0, v(l) = 1$, we *recover the Cauchy BC*.

In our proof of equivalence we have tacitly assumed that the variational solution is sufficiently regular, so we can integrate back by parts and introduce the second derivatives.

We say that the *two formulations are equivalent up to the regularity of the solution*.

1.2 Algebraic structure of the variational formulation

It is convenient to identify more precisely the algebraic structure of the weak formulation. The test functions form a *vector space*, the so-called *space of test functions*,

$$V = \{v(x) \, : \, v(0) = 0\} \tag{1.2.7}$$

To be fully precise, we must specify the regularity of the test functions yet, to be discussed in a moment.

The right hand side of the VBVP is identified as a *linear functional* $l(v)$ of test function $v \in V$, i.e.

$$l(\alpha_1 v_1 + \alpha_2 v_2) = \alpha_1 l(v_1) + \alpha_2 l(v_2) \, .$$

Similarly, the left-hand side of VBVP is identified as a *bilinear functional* $b(u,v)$ of solution u and test function v. This means that, with fixed u, the left-hand side is linear in v and, with fixed v, is linear in u.

Next we introduce the notion of the *lift*. A function $\tilde{u_0}$ is a *lift of Dirichlet data* u_0, if the restriction of $\tilde{u_0}$ to Dirichlet boundary $x = 0$ coincides with data u_0. In other words, the lift is simply an extension of u_0 to the whole domain. The simplest lift can be constructed by extending boundary data u_0 with a linear function,

$$\tilde{u_0}(x) = u_0(1 - \frac{x}{l}) \, .$$

Once we have introduced lift $\tilde{u_0}$, we can subtract it from solution u and the difference will vanish at $x = 0$, i.e. it will belong to the space of test functions. In other words, the solution must be the sum of the lift and a test function. The collection of such sums is identified as the *algebraic sum* of lift $\tilde{u_0}$ and space V, and called an *affine space*,

$$\tilde{u_0} + V = \{\tilde{u_0} + v \ : v \in V\} \, .$$

The final algebraic structure of the variational formulation can be expressed in a very concise form known as the *abstract variational formulation*,

$$\begin{cases} u \in \tilde{u_0} + V \\ b(u,v) = l(v), \quad \forall v \in V. \end{cases} \tag{1.2.8}$$

In the particular case of *homogeneous Dirichlet BC's* i.e., when $u_0 = 0$, we can use the zero function as a lift, and the abstract VBVP takes a simpler form,

$$\begin{cases} u \in V \\ b(u,v) = l(v), \quad \forall v \in V. \end{cases}$$

The case of non-homogeneous Dirichlet BC's can be characterized in yet another way. Once we have found a particular function $\tilde{u_0}$ that satisfies the non-homogeneous Dirichlet data, we make simply the substitution $u = \tilde{u_0} + w$ where $w \in V$ satisfies the homogeneous Dirichlet boundary conditions, and set to determine the perturbation w. The corresponding abstract formulation is then as follows.

$$\begin{cases} w \in V \\ b(w,v) = l(v) - b(\tilde{u_0}, v), \quad \forall v \in V \, . \end{cases} \tag{1.2.9}$$

Thus, solution of the non-homogeneous case reduces to the homogeneous one, provided we can find the lift and modify next the right-hand side according to the formula above. Note that, with $\tilde{u_0}$ fixed, the right-hand side of (8.3.10) defines a *modified* linear form,

$$l^{mod}(v) = l(v) - b(\tilde{u_0}, v) \, . \tag{1.2.10}$$

1.3 Equivalence with a minimization problem

The abstract VBVP is more than just a convenient way of communication. First of all, many other concrete BVP's have precisely the same algebraic structure and can be expressed using the same formalism. In fact, all classical theories in stationary (statics) linear mechanics lead to it, to mention a few: linear elasticity, various beam, plates and shell theories.

The abstract VBVP allows also for studying an underlying minimum principle corresponding to the classical Lagrange Theorem in mechanics. Let V be a subspace of some larger vector space X, u_0 an element of that space, and $l(v), b(u,v)$ some linear and bilinear forms defined on space X.

We begin by introducing a *quadratic functional*,

$$J(u) = \frac{1}{2}b(u,u) - l(u) \,,$$

frequently called the *functional of total potential energy*. We are now interested in minimizing the functional over the affine space $u_0 + V$,

$$\begin{cases} u \in u_0 + V \\ J(u) \to \min \end{cases} \tag{1.3.11}$$

Let u be a minimizer of functional J. Take an arbitrary test function v and a number ϵ. Then vector $u + \epsilon v$ is an element of affine space $u_0 + V$ (explain, why?), and we can introduce an auxiliary quadratic function of variable ϵ,

$$\begin{aligned} \phi(\epsilon) &= J(u + \epsilon v) \\ &= \tfrac{1}{2}b(u + \epsilon v, u + \epsilon v) - l(u + \epsilon v) \\ &= \tfrac{1}{2}b(v,v)\epsilon^2 + (\tfrac{1}{2}(b(u,v) + b(v,u)) - l(v))\epsilon + (\tfrac{1}{2}b(u,u) - l(u)) \,. \end{aligned}$$

In deriving the formula above, we have used linearity of l and bilinearity of b. If functional J attains its minimum at u, then quadratic polynomial $\phi(\epsilon)$ must attain its minimum at $\epsilon = 0$. This leads to the condition:

$$\frac{d\phi}{d\epsilon}(0) = \frac{1}{2}(b(u,v) + b(u,v)) - l(v) = 0 \,.$$

If we assume the symmetry of bilinear form,

$$b(u,v) = b(v,u) \quad \forall u, v \in X \,,$$

we obtain

$$b(u,v) - l(v) = 0 \quad \forall v \in V \,.$$

Thus, for the case of a symmetric bilinear form, the (abstract) VBVP forms a *necessary condition* for the existence of minimizer u. To derive a sufficient condition, let us assume that $u \in u_0 + V$

satisfies the abstract VBVP, and let $w \in u_0 + V$ be an arbitrary element. Then $w = u + (w - u)$ with $v := w - u \in V$, and

$$\begin{aligned} J(w) - J(u) &= J(u+v) - J(u) \\ &= \tfrac{1}{2}b(u+v, u+v) - l(u+v) - \tfrac{1}{2}b(u,u) + l(u) \\ &= \tfrac{1}{2}b(v,v) + \{b(u,v) - l(v)\} \\ &= \tfrac{1}{2}b(v,v) \end{aligned}$$

Thus J attains a (strict) minimum at u if

$$b(v,v) > 0 \quad \forall v \in V \,,$$

i.e. bilinear form $b(u,v)$ is *positive definite* on space V.

THEOREM 2

(Equivalence of minimization and variational problems)
If bilinear form $b(u,v)$ is symmetric and positive-definite on space of test functions V, then the variational and minimization problems are equivalent, i.e. $u \in u_0 + V$ solves (8.3.9) iff u solves (1.3.11)). ∎

How do the abstract conditions on symmetry and positive-definiteness of form $b(u,v)$ translate into concrete assumptions on data for our model 1D elliptic problem?

The condition on symmetry is easy to answer. The second and zero order terms are symmetric. The first order term cannot be symmetric unless it is simply zero. Thus symmetry is equivalent with the assumption that coefficient b vanishes.

The condition on positive definiteness is a little bit more complicated. We shall assume that the problem is *(uniformly, strongly) elliptic*, i.e. the leading coefficient $a(x)$ has a lower positive bound,

$$a(x) \geq a_{min} > 0 \,. \tag{1.3.12}$$

What matters is the fact that $a(x)$ is bounded uniformly away from zero. If it were negative, we could simply multiply the whole equation by -1 factor, and redefine the coefficients to satisfy the ellipticity condition. If we additionally assume that zero order coefficient $c(x) \geq 0$, and Cauchy BC coefficient $\beta \geq 0$, the resulting bilinear form is *positive semidefinite*,

$$b(u,u) = \int_0^l \{a(x)(u')^2 + c(x)u^2\}\, dx + \beta u(l)^2 \geq 0 \,.$$

If $b(u,u) = 0$, since all contributing terms are nonnegative, they all must vanish. In particular,

$$a(x)(u')^2 = 0, \quad x \in [0,l] \,,$$

and the ellipticity condition implies that $u' = 0$, i.e. u is equal to a constant C. There are three ways now that we can force the constant C to be zero.

- We can assume that $c(x)$ is positive over at least a subinterval of non zero measure. Vanishing of the term:
$$\int_0^l c(x)u^2\,dx = C^2 \int_0^l c(x)\,dx = 0\,,$$
implies that $C = 0$.

- Similarly, we can assume that $\beta > 0$. Then $\beta u(l)^2 = \beta C^2 = 0$ implies that $C = 0$ as well.

- The bilinear form need to be positive definite over the space of test functions only. But the test functions must vanish at $x = 0$ due to the Dirichlet BC. Thus the Dirichlet BC itself implies that $v(0) = C = 0$.

It is easy to see that only in the case of $c(x) = 0$, and Neumann BC's on both ends of the interval, we cannot guarantee that the constant vanishes. We shall return to the case of pure Neumann BC's later on.

Finally, we emphasize that the discussed conditions to guarantee positive definiteness of form $b(u, v)$ are only *sufficient* and not *necessary*. It is possible, for instance, to have a slightly negative zero order coefficient $c(x)$ and have the positive definiteness. It is also possible to talk about the positive definiteness for nonsymmetric forms.

1.4 Sobolev space $H^1(0, l)$

Variational formulations involve Lebesgue integrals and Lebesgue integrable functions. Any deeper study on the subject must start there. *Lebesgue Dominated Convergence Theorem, Fatou's lemma, Fubini'z Theorem, Change of Variables Theorem, Lebesgue Approximation Sums*, and the notion of properties satisfied *almost everywhere*, i.e. everywhere except for subsets of (Lebesgue) measure zero, are just examples of the most commonly used tools. Then come spaces $L^p(\Omega), 1 \le p \le \infty$, and the delicate understanding why the L^p- functions are really not functions but only *equivalence classes* of such, and why we cannot speak about e.g. boundary values of L^p-functions, see [98].

Distributional derivatives. The space of infinitely differentiable functions on $(0, l)$ with supports in $(0, l)$, denoted $C_0^\infty(0, l)$, is known as the *space of test functions*, and it provides a starting point for the *theory of distributions*. Note that these test functions are different from the test functions used in the variational formulations as they vanish (with all their derivatives) on the interval boundary and, therefore, cannot be used to 'test' boundary terms. A function w is called the *distributional*

derivative of function u if

$$\int_0^l u\phi'\,dx = -\int_0^l w\phi\,dx, \quad \forall \phi \in C_0^\infty(0,l)\,.$$

Notice that, for classical derivatives, the definition above reduces to the integration by parts formula. Typically we assume that both function u and its distributional derivative are L^p functions with the same exponent p.

In order to understand the difference between the classical and distributional derivatives, partition interval $(0,l)$ into two subintervals $(0,x_0)$ and (x_0,l), and consider, like for the interface problem, a function $u(x)$ specified by two different branches,

$$u(x) = \begin{cases} u_1(x), & x \in (0,x_0) \\ u_2(x), & x \in (x_0,l)\,. \end{cases},$$

We assume that $u_1 \in C^1[0,x_0]$ and $u_2 \in C^1[x_0,l]$. Let now $\phi \in C_0^\infty(0,l)$ be an arbitrary test function. We have,

$$\begin{aligned} \int_0^l u\phi'\,dx &= \int_0^{x_0} u_1\phi'\,dx + \int_{x_0}^l u_2\phi'\,dx \\ &= -\int_0^{x_0} u_1'\phi\,dx + u_1(x_0)\phi(x_0) - \int_{x_0}^l u_2'\phi\,dx - u_2(x_0)\phi(x_0)\,. \end{aligned}$$

If we introduce now a function $w(x)$

$$w(x) = \begin{cases} u_1'(x), & x \in (0,x_0) \\ u_2'(x), & x \in (x_0,l) \end{cases},$$

we can rewrite our result in the single integral form,

$$\int_0^l u\phi'\,dx = -\int_0^l w\phi\,dx + [u(x_0)]\phi(x_0)\,. \tag{1.4.13}$$

Here $[u(x_0)]$ denotes the (possible) jump of function u at interface x_0, $[u(x_0)] = u_2(x_0) - u_1(x_0)$. If function u is continuous at x_0, the second term is gone, and function w is the distributional derivative of u. The fact that function w has not been defined at interface x_0, does not matter; Lebesgue integral is insensitive to changes of the integrand on subsets of measure zero, and any countable set of points *is of measure zero*. In other words, it is sufficient to define the distributional derivative only up to a subset of measure zero. Thus a function that consists even of an infinite (but countable) number of C^1 branches may not be differentiable at the interface points but, as long as it is globally continuous, it will be differentiable in the distributional sense.

If function u is discontinuous at x_0, the situation is more difficult. The second term cannot be written in terms of an integral and the equality has to be interpreted in terms of *distributions*. Distributions are defined as *linear functionals* defined on the space of test functions $C_0^\infty(0,l)$. A

precise definition involves setting up a topology for both the test functions and distributions. We mention here only that we distinguish between *regular* and *irregular distributions*. A distribution is regular, if it is generated by a function, i.e. it is of a form,

$$\phi \to L_f(\phi) := \int_0^l f\phi \, dx \, .$$

Regularity of generating function f is dictated by integrability, it is assumed that f is *locally Lebesgue integrable*. Every function $f \in L^p(0,l)$ belongs to this class. Any distribution that is not regular, is called *irregular*. The most important example of an irregular distribution is the *Dirac delta functional*,

$$\phi \to \delta_{x_0}(\phi) := \phi(x_0) \, . \tag{1.4.14}$$

A mathematically incorrect but, nevertheless, popular alternative notation is :

$$\phi \to \int_0^l \delta(x-x_0)\phi(x) \, dx := \phi(x_0) \, .$$

The integral sign here does not signify the Lebesgue integral, and the delta symbol does not signify a function. Formula (1.4.13) can then be rewritten in a concise form,

$$u'(x) = w(x) + [u(x_0)]\delta(x-x_0) \, .$$

A mathematically precise statement is,

$$\frac{d}{dx} L_u = L_w + \delta_{x_0} \, ,$$

where L_u denotes the regular distribution generated by function u. For the *Heaviside function*,

$$u(x) = \begin{cases} 0 & x \in (0, x_0) \\ 1 & x \in (x_0, l) \, , \end{cases}$$

the distributional derivative is equal to the Dirac delta.

For more information on distributions consult [114].

Finite energy solutions. For problems admitting the minimization principle, it is natural that we shall restrict ourselves only to a class of functions that produce a finite energy,

$$X = \{u \, : \int_0^l a(u')^2 + cu^2 \, dx + \beta u(l)^2 < \infty\} \, . \tag{1.4.15}$$

The energy defines a natural (energy) product in X,

$$(u,v)_E = \int_0^l (u'v' + cuv) \, dx + \beta u(l)v(l) \, , \tag{1.4.16}$$

with the corresponding energy norm,

$$\|u\|_E = ((u,u)_E)^{\frac{1}{2}} \ . \tag{1.4.17}$$

Actually, we have the right to use the names of *inner product* and *norm* only if the bilinear form is positive definite on the whole space X, i.e. without resorting to the Dirichlet boundary condition. Otherwise, energy defines only a *semiproduct* and a *seminorm*. When restricted to the space of test functions, though, the seminorm becomes a norm, recall the discussion on the positive definiteness.

Sobolev space. It is convenient to introduce a general functional space that may serve for a large class of elliptic problems, and that is not tied to specific material data or boundary conditions.

This leads to the definition of the *Sobolev space of order one*,

$$H^1(0,l) = \{u \ : \ \int_0^l \left(|u'|^2 + |u|^2\right) \ dx < \infty\} \ ,$$

with the corresponding norm,

$$\|u\|_{H^1(0,l)} = \|u\|_1 = \left(\int_0^l \left(|u'|^2 + |u|^2\right) \ d\boldsymbol{x}\right)^{\frac{1}{2}} \ .$$

We shall also use the corresponding first order seminorm,

$$|u|_{H^1(0,l)} = |u|_1 = \left(\int_0^l |u'|^2 \ dx\right)^{\frac{1}{2}} \ ,$$

and the L^2-norm,

$$|u|_{L^2(0,l)} = |u|_0 = \left(\int_0^l |u|^2 \ d\boldsymbol{x}\right)^{\frac{1}{2}} \ .$$

With the Sobolev space in place, we can make now the variational formulation precise. The space of test functions V is a subspace of the Sobolev space consisting of functions that vanish at $x = 0$,

$$V = \{v \in H^1(0,l) \ : v(0) = 0\} \ .$$

We assume that the lift of Dirichlet data $\tilde{u}_0$ belongs to the Sobolev space (a finite energy extension) as well, $\tilde{u}_0 \in H^1(0,l)$. This makes the affine space $\tilde{u}_0 + V$ a subset of the Sobolev space,

$$\tilde{u}_0 + V \subset H^1(0,l) \ .$$

The algebraic formulation remains intact,

$$\begin{cases} u \in \tilde{u}_0 + V \\ b(u,v) = l(v), \quad \forall v \in V \ . \end{cases}$$

1.5 *Well posedness of the variational BVP*

Having introduced the Sobolev space and proper functional setting, we shall comment now shortly on the well-posedness of the VBVP, i.e. demonstrate that, under appropriate assumptions, the problem has a unique solution that depends continuously on the load data. This section is intended for mathematically advanced readers.

THEOREM 3

(Lax-Milgram Theorem)

Let X be a Hilbert space with a subspace V, and let $b(u,v)$ denote a bilinear form defined on $X \times X$ which is continuous, i.e. there exists $M > 0$ such that

$$|b(u,v)| \leq M\|u\|\|v\|, \quad \forall u, v, \in X \ ,$$

and it is V-coercive, i.e. there exists $\alpha > 0$ such that

$$\alpha\|v\|^2 \leq b(v,v), \quad \forall v \in V \ .$$

Let $l(v)$ be an arbitrary linear and continuous functional defined on V, i.e.

$$|l(v)| \leq C\|v\|, \quad \forall v \in V \ .$$

Finally, let $\tilde{u}_0 \in X$ be an arbitrary element of X. Then, the abstract VBVP,

$$\left\{ \begin{array}{l} u \in \tilde{u}_0 + V \\ b(u,v) = l(v), \quad \forall v \in V \ , \end{array} \right.$$

has a unique solution u such that

$$\|u\| \leq \|\tilde{u}_0\|(1 + \frac{M}{\alpha}) + \frac{C}{\alpha}$$

Here $\|\cdot\|$ denotes the norm in space X. ∎

Proof: See [98], page 583.

Notice that we have not assumed the symmetry of the bilinear form. In the symmetric case, the Lax-Milgram Theorem reduces to the Riesz Theorem. As coercivity implies positive definiteness, the VBVP is then equivalent to the minimization problem.

Assuming that material coefficients are bounded,

$$|a(x) \leq a_{max}, |b(x)| \leq b_{max}, |c(x)| \leq c_{max},$$

we use the Cauchy-Schwartz inequality ([98],page ??) to show the continuity of the integral part of the bilinear form.

$$|\int_0^l \{au'v' + bu'v + cuv\}\, dx| \leq (a_{max} + b_{max} + c_{max})\|u\|_{H^1(0,l)}\|v\|_{H^1(0,l)} \ .$$

The continuity of the boundary term follows from the embedding of space $H^1(0,l)$ into $C[0,l]$ ([], page ??, see Exercise ?? for an elementary argument.), which implies that functions from $H^1(0,l)$ must be globally continuous[1], and there exists $c>0$ such that

$$|u(x)| \leq c\|u\|_{H^1(0,l)}, \quad \forall u \in H^1(0,l), \quad \forall x \in [0,l]\,.$$

Consequently,

$$|\beta u(l)v(l)| \leq |\beta| c^2 \|u\|_{H^1(0,l)}\|v\|_{H^1(0,l)}\,,$$

with the final estimate,

$$|b(u,v)| \leq (a_{max}+b_{max}+c_{max}+|\beta|c^2)\|u\|_{H^1(0,l)}\|v\|_{H^1(0,l)}\,.$$

The coercivity condition is related to ellipticity condition (1.3.12), and discussion follows very much the same lines as for the positive definitness of the bilinear form. In fact all three scenarios discussed in section 1.3 guarantee not only the positive definiteness but the coercivity as well. With the Dirichlet BC in place, the coercivity over the space of test functions follows from the following inequality.

THEOREM 4
(Elementary Poincare Inequality)
There exists a constant $C_P>0$ such that

$$C_P\|v\|_{L^2(0,l)} \leq |v|_{H^1(0,l)}\,,$$

for every $v \in H^1(0,l)$ such that $v(0)=0$. ∎

Proof:

$$\begin{aligned}\int_0^l (v(x))^2\,dx &= \int_0^l (\int_0^x v'(\xi)\,d\xi)^2\,dx \leq \int_0^l \left(\int_0^x d\xi\right)\left(\int_0^x (v')^2\,d\xi\right)\,dx \\ &\leq \int_0^l (v')^2\,d\xi \int_0^l x\,dx \leq \frac{l^2}{2}|v|^2_{H^1(0,l)}\,.\end{aligned}$$

∎

Using now the ellipticity condition (1.3.12), and the Poincare Inequality, we have,

$$b(v,v) \geq a_{min}|v|^2_{H^1(0,l)} \geq \frac{1}{2}a_{min}\min\{C_P,1\}\|v\|^2_{H^1(0,l)}\,.$$

[1]This explains why we do not have to state explicitly continuity of the solution for the interface problem, if we look for a solution in $H^1(0,l)$

The analysis of the case with a positive Cauchy BC constant $\beta > 0$ follows the lines of the proof of the Poincare Inequality. We have,

$$\begin{aligned}\int_0^l (v(x))^2\,dx &= \int_0^l \left(v(l) - \int_x^l v'(\xi)\,d\xi\right)^2 dx \le \int_0^l 2(v(l))^2 + 2(\int_x^l v'(\xi)\,d\xi)^2\,dx \\ &\le 2l(v(l))^2 + 2l^2|v|^2_{H^1(0,l)} \le C\left((v(l))^2 + |v|^2_{H^1(0,l)}\right),\end{aligned}$$

where $C = \min\{2l, 2l^2\}$. From this follows the coercivity, as

$$b(v,v) \ge a_{min}|v|^2_{H^1(0,l)}\,,$$

and, consequently,

$$b(v,v) \ge \min\{a_{min},\beta\}\left(|v|^2_{H^1(0,l)} + (v(l))^2\right) \ge C^{-1}\min\{a_{min},\beta\}\|v\|^2_{L^2(0,l)}\,,$$

with C being the constant from the previous inequality. Summing up the two inequalities above, we get the coercivity condition. Notice that in this case, the bilinear form is coercive over the whole space $H^1(0,l)$, and no kinematic condition is needed.

Finally, in the case of coefficient $c(x)$ possessing a positive lower bound,

$$c(x) \ge c_{min} > 0\,,$$

we simply have,

$$b(v,v) \ge \min\{a_{min}, c_{min}\}\|v\|^2_{H^1(0,l)}\,.$$

Thus, in all three discussed cases, the coercivity condition is satisfied, and the Lax-Milgram Theorem can be used to conclude the well posedness of our model problem. The discussed sufficient conditions are by no means optimal, and they are intended only to help to build an elementary intuition for the coercive case, before we start computing. We illuminate a few more subtle points in the following exercises. The tricky case of pure Neumann BC's will be discussed later.

1.6 Examples from mechanics and physics

In this section we review some standard examples from physics and mechanics leading to the two-point boundary-value problem discussed in this chapter.

1.6.1 Elastic bar

Consider an elastic bar displayed in Fig. 1.1. The bar consists of two segments of length l_1 and l_2, made of two different materials with Young's module E_1, E_2, and densities ρ_1, ρ_2, respectively. We also assume that the corresponding cross-sectional areas A_1, A_2 may be different. The bar is fixed

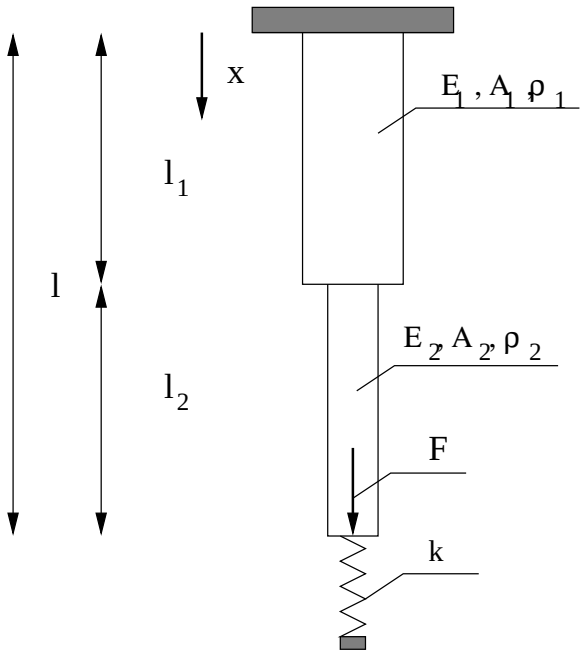


Figure 1.1: An elastic bar

on the top, and supported additionally with an elastic spring of stiffness k on the bottom. The bar is loaded with its own weight, and an additional concentrated force F on the bottom. We introduce a coordinate x with origin at the top of the bar, and consider the following functions of x.

- $u(x)$ is the (vertical) *displacement* of a section located at level x.
- $\epsilon = u'$ is the *strain*.
- $\sigma = Eu'$ is the *stress*.
- $N = A\sigma = EAu'$ is the *axial force*, coefficient EA is called the (longitudal) *stiffness*.

Consider now three different sections of the bar shown in Fig. 1.2. The first section corresponds to $x \in (0, l_1)$ or $x \in (l_1, l_2)$. As long as x comes from the *interior* of any of the two segments, for sufficiently small Δx, the whole cut off section $[x, x + \Delta]$ will be contained entirely in one bar segment only.

The bar is in equilibrium, and so is each subsection of it. The sum of forces acting on the section has to be zero,

$$N + \Delta N - N \underbrace{A\Delta x}_{\Delta V} \rho g = 0 \,. \tag{1.6.18}$$

Here N and $N + \Delta N$ are the axial forces on both sides of the section, ΔV is the volume of the section, and g stands for the acceleration due to gravity. The notation assumed for the axial force

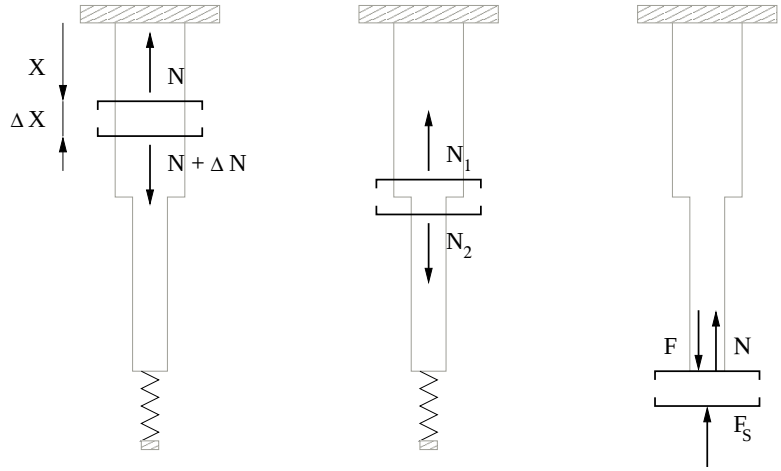


Figure 1.2: Elastic bar example. Derivation of equilibrium equation, interface and boundary conditions

corresponding to cross-section $x + \Delta x$, reflects already a limiting process. If we let the section shrink to a point, $\Delta x \to 0, \Delta N \to 0$, and the equilibrium equation will be automatically satisfied in the limit,

$$N - N = 0 \, .$$

Please consult on the use of the *local system of coordinates* for describing internal forces in bars and beams, [21], pp. 294.

Dividing equation (1.6.18) by Δx, we obtain,

$$\frac{\Delta N}{\Delta x} = -A\rho g \, .$$

Passing to the limit with $\Delta x \to 0$, we arrive at the bar *equilibrium equation*,

$$\frac{dN}{dx} = -A\rho G \quad \Rightarrow \quad -\frac{d}{dx}\left(\underbrace{EA}_{a}\frac{du}{dx}\right) = \underbrace{A\rho g}_{f} \, .$$

The equilibrium equation has to be satified for every $x \in (0, l_1)$ and $x \in (l_1, l_2)$. Stiffness EA is the coefficient a in our general setting, and the source term f corresponds to the weight loading.

We now turn to the second section displayed in Fig. 1.2, The section is described by interval (x_1, x_2) with $x_1 < l_1$ and $x_2 > l_1$. We again start with the equilibrium equation,

$$N_2 - N_1 - A\Delta x \rho g = 0 \, .$$

Passing with $x_1, x_2 \to l$, we get the same equality of the axial forces as before, which implies the interface condition,

$$N_1 = N_2 \quad \Rightarrow \quad E_1 A_1 \frac{du_1}{dx}(l_1) = E_2 A_2 \frac{du_2}{dx}(l_1) \, .$$

Obviously, the other interface condition,

$$u_1(l_1) = u_2(l_1) \, ,$$

expresses the continuity of the displacement. Finally, working with the last section shown in Fig. 1.2, we obtain,

$$\underbrace{-F_s}_{\text{spring force}} + \underbrace{F}_{\text{concentrated force}} - N + A\Delta x \rho g = 0 \, .$$

Recalling the constitutive relation for a linear spring, $F_s = k\Delta l$, and relating the compression Δl to the displacement of the end-point of the bar, $\Delta l = u(l)$, we get $F_s = ku(l)$. This leads to the Cauchy boundary condition,

$$N + ku(l) = F \quad \Rightarrow \quad EA\frac{du}{dx}(l) + ku(l) = F \, .$$

Finally, we observe that the bar is fixed at $x = 0$,

$$u(0) = 0 \, .$$

which leads to the *homogeneous* Dirichlet boundary condition at $x = 0$.

The problem is symmetric, material coefficients are positive, $a = EA > 0, b = c = 0, \beta = k > 0$, so the problem admits the equivalent Lagrange principle of minimum potential energy. The total potential energy functional,

$$J(u) = \frac{1}{2}\left(\int_0^l (EAu')^2 \, dx + ku(l)^2\right) - \left(\int_0^l A\rho g u \, dx + Pu(l)\right) \, ,$$

includes the strain elastic energy accumulated in the bar, and in the spring, and the potential energy corresponding to the weight and the concentrated force. The potential energy corresponding to the load may also be interpreted as the *work of external forces* on displacement field $u(x)$.

The variational formulation,

$$\begin{cases} u(0) = 0 \\ \displaystyle\int_0^l EAu'v' \, dx + ku(l)v(l) = \int_0^1 A\rho g v \, dx + Pv(l) \\ \quad \text{for every virtual displacement } v, \; v(0) = 0 \end{cases}$$

is the *Principle of Virtual Work*:

> *The work of internal forces corresponding to the actual displacement $u(x)$ on a virtual displacement $v(x)$, must be equal to the work of the external forces on the same virtual displacement, for any kinematically admissible virtual displacement $v(x)$: $v(0) = 0$.*

The elementary bar theory may be developed in a more systematic way as an approximation to the three-dimensional elasticity, see e.g. [70].

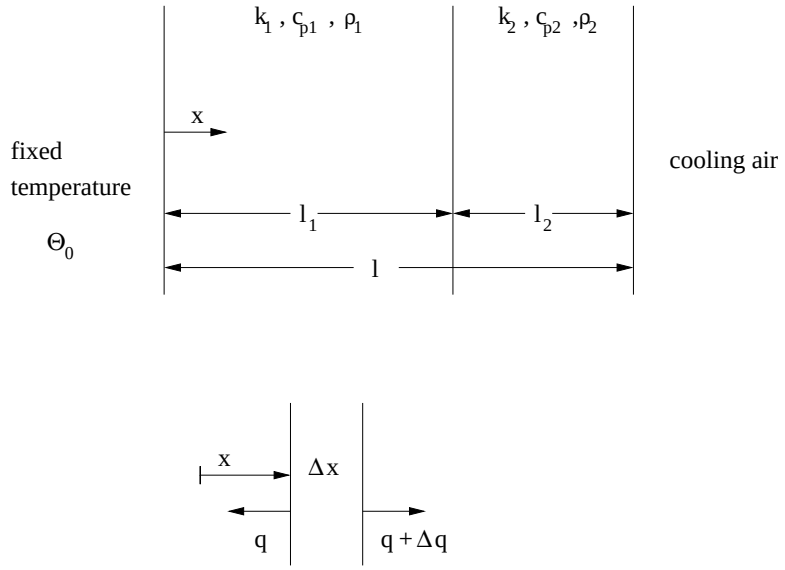


Figure 1.3: Heat conduction in a slab

1.6.2 Heat conduction

Consider a wall consisting of two slabs shown in Fig. 1.3. The two layers are made of different materials with different thermal conductivities k_1, k_2, mass specific heat coefficients c_{p_1}, c_{p_2}, and densities ρ_1, ρ_2. The left boundary is exposed to a constant temperature θ_0, the right boundary is being cooled with a moving fluid, say air. The slabs are of thickness l_1 and l_2. We introduce a coordinate x with the origin on the left boundary, and consider the following functions of x.

- $\theta(x)$ is the temperature distribution in the wall. We assume that the problem is one-dimensional, i.e. the temperature depends only upon the single coordinate x.
- $q = -k\theta'$ is the heat flux (per unit crossectional area), the relation between the flux and the temperature gradient being the *Fourier law of heat conduction.*

As for the bar problem, we consider an arbitrary section $(x, x + \Delta x)$ with a unit crossectional area. The energy balance (First Law of Thermodynamics) principle says that the the time rate with which the total energy in the section changes must be in balance with its flow through the boundary,

$$\frac{d}{dt}\int_x^{x+\Delta x} c_p\rho\theta \, dx + q + \Delta q - q = 0 \, .$$

In presence of an additional *heat source* within the segment (a heating pipe in your wall ?), represented with the rate q^* at which the heat is generated *per unit volume*, we add an additional term to the balance equation,

$$\frac{d}{dt}\int_x^{x+\Delta x} c_p\rho\theta \, dx + q + \Delta q - q = \int_x^{x+\Delta x} q^* \, dx \, .$$

First of all, we assume that the temperature $\theta(x)$ is continuous. As for the bar problem, by shrinking with $\Delta x \to 0$, we learn, that the heat flux must be continuous as well, including the interface. Using the *Mean-Value Theorem* for integrals, we can replace both integrals in the balance equation with the product of the value at some intermediate point and the interval length,

$$\frac{d}{dt} c_p \rho \theta(x_1) \Delta x + \Delta q = q^*(x_2) \Delta x \ .$$

Upon dividing by Δx and passing to the limit with $\Delta x \to 0$ (both intermediate points x_1, x_2 will converge to point x), we get the energy balance equation in the differential form,

$$c_p \rho \frac{d\theta}{dt} + \frac{dq}{dx} = q^* \ .$$

(We have assumed that c_p, ρ are constant within each of the slabs). In the case of a *stationary heat conduction*, the temperature is independent of time, and the equation reduces to,

$$\frac{dq}{dx} = q^* \quad \Rightarrow \quad -\frac{d}{dx}\left(k \frac{d\theta}{dx} \right) = q^* \ .$$

Finally, the cooling on the right end is described by the empirical *Newton's Law of Cooling*,

$$q = h(\theta - \theta_f) \ ,$$

where θ_f is a prescibed temperature of the cooling (in Texas, heating) air, and h is the *heat transfer coefficient* representing in a lumped form a very complicated exchange of energy between the solid and the fluid, being a function of many factors such as the composition of the fluid, the hydronamics of the fluid motion, the precise geometry of the boundary, etc. This leads to the Cauchy boundary condition,

$$k \frac{d\theta}{dx} + h\theta = h\theta_f \ .$$

The problem fits perfectly into our general framework, material data are $a = k, b = c = 0, \beta = h$, and the load data are specified by source term $f = q^*$ and boundary terms θ_0, $\gamma = h\theta_f$. The problem is again symmetric and positive-definite. For more information on modeling of heat transfer, consult [31].

1.6.3 Vibrations of an elastic bar

We return to our elastic bar problem. Or better off, think in terms of a slab of an elastic material with negligible Poisson ratio, see Fig. 1.4. The bar, or slab, is made of a material with Young's modulus E and density ρ. Both constants may vary locally, i.e. we may have a composite or inhomogeneous material. The left end of the slab is subjected to prescibed vibrations,

$$u(0,t) = \Re(u_0 e^{i\omega t}) \ ,$$

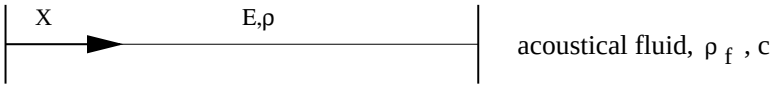


Figure 1.4: A vibrating elastic bar interacting with an acoustical fluid

where ω is a prescribed frequency of vibrations. Displacement $u(x,t)$ of the bar has to satisfy the equation of motion,

$$\rho\ddot{u} - (Eu\prime)' = 0\,.$$

To simplify the problem, we assume that no forces are present. We are interested in determining the time-harmonic solution,

$$u(x,t) = \Re(e^{i\omega t}u(x))\,,$$

where complex-valued $u(x)$ is called the *phasor* of the displacement field. This leads to the *Helmholtz equation* for the (phasor) displacement $u(x)$,

$$-(Eu\prime)' - \rho\omega^2 u = 0\,.$$

In general, the solution to the transient problem consists of a transient term and the time-harmonic (also called steady-state) term. In presence of any dissipation mechanism, the transient term will die out, and the solution will be dominated by the steady-state form.

In presence of material discontinuities, as usual, we have to divide the bar into slabs corresponding to different materials, and state explicitly the interface conditions.

On the right, the slab interacts with an acoustical fluid with density ρ_f and velocity of sound in fluid c_f. Assuming a one dimensional behavior of the fluid as well, the equations of the linear acoustics read as follows,

$$\begin{cases} \dot{p} + \rho_f c_f^2 v' = 0 \\ \rho_f \dot{v} + p' = 0\,. \end{cases}$$

Here v is the velocity of the fluid, and p denotes the acoustical pressure. Assuming time-harmonic vibrations, we get,

$$\begin{cases} i\omega p + \rho_f c_f^2 v' = 0 \\ \rho_f i\omega v + p' = 0\,. \end{cases}$$

This leads to a single Helmholtz equation, in terms of either the pressure or the velocity,

$$c_f^2 v'' + \omega^2 v = 0.$$

We can solve it explicitly to get,

$$v(x) = Ae^{-\frac{\omega}{c_f}x} + Be^{\frac{\omega}{c_f}x}\,,$$

with two unknown constants A, B. The first term represents a wave going to the right, from the bar to infinity, the second term represents a wave going from the infinity to the left, and it has no physical meaning. We discard it to get,

$$v(x) = Ae^{-\frac{\omega}{c_f}x} .$$

This, upon determining pressure p, allows for determining the *impedance of the fluid*,

$$\frac{p}{v} = \rho_f c_f .$$

The physical interface conditions between the elastic slab and the acoustical fluid call for the continuity of velocities and internal forces,

$$i\omega u = v, \quad Eu' = p .$$

Using the known impedance for the fluid, we get the corresponding *impedance condition for the solid*,

$$Eu' = p = \rho_f c_f v = i\omega \rho_f c_f u .$$

The final formulation of our problem calls for solving the Helmholtz equation, accompanied with Dirichlet BC at $x = 0$ and Cauchy (impedance) BC at $x = l$,

$$\begin{cases} -(Eu\prime)' - \rho\omega^2 u = 0 \\ u(0) = u_0 \\ Eu'(l) - i\omega\rho_f c_f u(l) = 0 \end{cases}$$

The problem is symmetric but not longer positive definite. We seek a *complex valued* solution, and the impedance constant $\beta = -i\omega\rho_f c_f$ is imaginary. The problem can be easily generalized to the case of a viscoelastic material [38], in which case the elastic constant E also becomes complex-valued. The problem is driven by the Dirichlet condition (the only load term). The impedance condition models transfer of the energy from the body into the fluid (one way only), a phenomenon known as the *radiation damping*.

Exercises

Exercise 1.1 Consider the following, 1D boundary-value problem with periodic boundary conditions. Derive the variational formulation and demonstrate the equivalence with the classical formulation.

$$\begin{cases} -(a(x)u'(x))' + b(x)u'(x) + c(x)u = f(x) & \text{in } (0,1) \\ u(0) = u(1); \quad u'(0) = u'(l) \end{cases}$$

Assumptions: $a(0) = a(1);\ b(0) = b(1);\ c(0) = c(1);\ f(0) = f(1)$
Formulate sufficient conditions for the material data under which the problem admits the minimum energy formulation.

Exercise 1.2 Consider the following, 1D boundary-value problem with an interface condition. Derive the variational formulation and demonstrate the equivalence with the classical formulation.

$$\begin{cases} -(a_1(x)u_1'(x))' + b_1(x)u_1'(x) + c_1(x)u_1 = f_1(x) & \text{in } (0, x_0) \\ -(a_2(x)u_2'(x))' + b_2(x)u_2'(x) + c_2(x)u_2 = f_2(x) & \text{in } (x_0, l) \\ u_1(x_0) = u_2(x_0); \quad a_2(x_0)u_2'(x_0) - a_1(x_0)u_1'(x_0) = s \\ -a_1(0)u_1'(0) + \beta_0 u_1(0) = \gamma_1; \quad u_2(l) = \gamma_l \end{cases}$$

Interpret the contribution from the interface boundary condition as an irregular (Dirac delta) source term $f(x)$. Formulate sufficient conditions for the material data under which the problem admits the minimum energy formulation.

Exercise 1.3 Consider the elastic bar example presented in section 1.6, see Fig. 1.1, loaded with *an additional force* F_1, applied at the interface $x = l_1$, and acting downward. Discuss changes in the classical, variational and minimum energy formulations necessary to accommodate the force.

Exercise 1.4 Let f be a real-valued function defined on open interval $(0, l)$. Closure of the set containing all points x where function f takes a value different from zero,

$$\overline{\{x \in \mathbb{R} \,:\, f(x) \neq 0\}}\,,$$

is called the *support* of the function. Define $f(x)$ as follows,

$$f(x) = \begin{cases} 0 & x \in (0, \frac{1}{3}) \\ x - \frac{1}{2} & x \in (\frac{1}{3}, \frac{2}{3}) \\ \frac{1}{6} & x \in (\frac{2}{3}, 1)\,. \end{cases}$$

What is the support of function f. Is it contained in interval $(0, 1)$?

Exercise 1.5 Compute the *distributional derivative* of the following function.

$$u(x) = \begin{cases} x^2 & x \in (0, 1) \\ 2 - x & x \in (1, 2) \end{cases}$$

Is the derivative a function itself ? What is its value at $x = 1$?

Exercise 1.6 Compute the second derivative of the function from the previous problem. Answer the same questions.

Exercise 1.7 Let

$$u_{avg} = \frac{1}{l}\int_0^l u(x)\,dx$$

be the average of a continuous function u defined on interval $[0,l]$. Explain why function $u(x) - u_{avg}$ must vanish at some point $x_0 \in [0,l]$. Consequently,

$$u(x) - u_{avg} = \int_{x_0}^l u'(\xi)\,d\xi\ .$$

Use the formula to prove that there exists a constant $C > 0$, independent of u such that

$$|u(x)| \leq C\|u\|_{H^1(0,l)}, \quad \forall x \in [0,l]\ .$$

This, plus a density argument, proves that Sobolev space $H^1(0,l)$ is continuously embedded in the space $C[0,l]$ of continuous functions defined on the closed interval $[0,l]$.

Exercise 1.8 Prove *Silvester criterion* for positive definitness of a symmetric matrix: *a matrix* $a_{ij} = a_{ji} \in \mathbb{R},\ i,j,=1,\ldots,n$ *is positive definite, i.e.*

$$\sum_{i,j=1}^n a_{ij}\xi_j\xi_i > 0, \quad \forall \boldsymbol{\xi} \in \mathbb{R}^n, \boldsymbol{\xi} \neq \mathbf{0}\ ,$$

if an only if,

$$\det(a_{ij},\ i,j,=1,\ldots,m) > 0, \quad \forall m = 1,\ldots,n\ .$$

Chapter 2

Galerkin Method

In this chapter we discuss the fundamentals of the Galerkin method: the idea of finite-dimensional approximation, the notion of the best approximation error, and an elementary approximation error analysis.

2.1 Finite dimensional approximation of the VBVP

The space of test functions is a function space and it is infinite- dimensional. In general, it takes an infinite number of functions to approximate the unknown solution. The Galerkin's idea is to approximate it with a finite number of *linearly independent* functions $e_i = e_i(\boldsymbol{x})$ from test space V,

$$u(\boldsymbol{x}) \approx \tilde{u}_0(\boldsymbol{x}) + \sum_{j=1}^{N} u_j e_j(\boldsymbol{x}) \,,$$

where the coefficients u_j are to be determined. Functions e_j are called the *basis functions*, and the unknown coefficients u_j are called the *global degrees of freedom (d.o.f.)*.

The idea is very general, and we will discuss it in context of the abstract VBVP, using argument-less notation. Along with the solution, we use the same basis functions to approximate the test functions,

$$u \approx \sum_{j=1}^{N} u_j e_j \qquad v \approx \sum_{l=1}^{N} v_l e_l \,.$$

We substitute now the approximations into the abstract VBVP,

$$b(\sum_{j=1}^{N} u_j e_j, \sum_{l=1}^{N} v_l e_l) = l(\sum_{l=1}^{N} v_l e_l) \,. \tag{2.1.1}$$

The equation is to be satisfied *for every* test function which, for the finite-dimensional approximation, translates into saying: *for every choice of coefficients* $v_l, l = 1, \ldots, N$. We select $v_l = \delta_{li}$

(Kronecker's delta) to arrive at a system of N algebraic equations,

$$b(u_0 + \sum_{j=1}^{N} u_j e_j, e_i) = l(e_i), \quad i = 1, \ldots, N \, .$$

Conversely, we can multiply the equations above with arbitrary coefficients v_i, sum up wrt i, and use linearity of form l, and form $b(u, v)$ wrt v, to arrive at the previous equation. The two formulations are thus equivalent. Additionally, if we also use the linearity of form $b(u, v)$ in u (we are solving a *linear problem*), we realize that the algebraic equations are linear,

$$b(u_0, e_i) + \sum_{j=1}^{N} b(e_j, e_i) u_i = l(e_i), \quad i = 1, \ldots, N \, .$$

Function u_0 representing a lift of Dirichlet data is known, so it is advisable to move it to the right-hand side,

$$\sum_{j=1}^{N} b(e_j, e_i) u_j = l(e_i) - b(u_0, e_i), \quad i = 1, \ldots, N \, .$$

Matrix $b_{ij} = b(e_j, e_i)$ will be called the *(global) stiffness matrix*, vector $l_i = l(e_i)$ is the *(global) load vector*, and vector $l_i^{mod} = l^{mod}(e_i) = l(e_i) - b(u_0, e_i)$ is the *(global) modified load vector*. In order to solve the approximate problem, we have to compute matrices $\boldsymbol{B} = \{b_{ij}\}, \boldsymbol{l}^{mod} = \{l_i^{mod}\}$, and solve the system of linear equations

$$\sum_{j=1}^{N} b_{ij} u_j = l_i^{mod}, \quad i = 1, \ldots, N \, , \tag{2.1.2}$$

or, using the matrix notation,

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{l}^{mod} \, .$$

Here the unknown d.o.f. u_j have been put into a *vertical* vector.

Summing up, in order to determine the approximate solution, we need to:

- Select the lift of Dirichlet data $u_0 = \tilde{u}_0$,
- Calculate the global stiffness matrix and the modified load vector,
- Solve system (21.1).

We can look at the Galerkin method more precisely using the language of vector spaces. Functions $e_j, j = 1, \ldots, N$ span an N-dimensional subspace V_h of V. Index h is here an abstraction for a maximum element size, and it comes from the language of finite elements. Thus, by definition,

functions e_j indeed form a basis for space V_h. The Galerkin system of equations (2.1.1) can now be written in a more concise form of the *approximate (abstract) variational boundary-value problem*.

$$\begin{cases} u_h \in u_0 + V_h \\ b(u_h, v_h) = l(v_h), \quad \forall v_h \in V_h \, . \end{cases} \tag{2.1.3}$$

In principle, approximate solution u_h depends only upon the space V_h and it is independent of the choice of basis functions e_j, as long as they span the same approximate subspace. In practice, however, the choice of the basis functions affects the conditioning of the final discrete system of equations and, due to the round-off error effects, may significantly influence the actual approximate solution.

When approximating a BVP, we construct a sequence of finite-dimensional spaces V_h, and solve the resulting linear systems for the corresponding approximate solution u_h. In finite elements, these spaces correspond to different meshes that are refined to yield better and better approximations to the original problem. All discussed quantities: basis functions $e_j = e_j^h$, dimension $N = N^h$ of V_h, global d.o.f. $u_j = u_j^h$, stiffness matrix and load vector, depend upon index h.

Petrov-Galerkin method. The discussed version of the Galerkin method is known as the *Bubnov-Galerkin* method. In a more general approach, known as *Petrov-Galerkin* method, test functions are approximated using a different set of basis functions. The basis functions to approximate the solution are called then *trial basis functions*, and those used for the test functions are known as *test basis functions*. Specially selected test basis functions allow for improving stability properties of the Galerkin approximation for problems that lack such a stability to begin with, e.g. convection dominated diffusion. In this monograph, however, we shall stick with the Bubnov-Galerkin approach only.

Internal approximation. Approximate spaces V_h are assumed to be *subspaces* of the infinite dimensional space of test functions V, i.e. basis functions e_j belong to space V. This is called an *internal approximation*, and leads to *conforming finite elements* that are the subject of this monograph. In a more general version, functions e_j may come from outside of space V, we talk then about *external approximations* of BVP's. The concept of internal and external approximations was introduced by Cea [27] in 1964.

Orthogonality relation. Substituting $v = v_h \in V_h$ (we can do this because the approximation is *internal*, i.e. $V_h \subset V$) in (8.3.9) and subtracting from it (2.1.3) we obtain the famous *orthogonality relation*,

$$b(u - u_h, v_h) = 0 \quad \forall v_h \in V_h \, . \tag{2.1.4}$$

The orthogonality relation states that *the approximation error $e_h = u - u_h$ is orthogonal (in the sense of bilinear form $b(u,v)$) to approximate test space V_h.*

The Ritz method. In the case of a symmetric and positive- definite bilinear form $b(u,v)$, solution u to the variational problem can be obtained by solving the corresponding minimization problem,

$$\begin{cases} u \in u_0 + V \\ J(u) = \frac{1}{2} b(u,u) - l(u) \rightarrow \min \ . \end{cases}$$

Given the finite-dimensional approximate space V_h, it is natural to pose the corresponding finite-dimensional approximation minimization problem,

$$\begin{cases} u_h \in u_0 + V_h \\ J(u_h) = \frac{1}{2} b(u_h,u_h) - l(u_h) \rightarrow \min \ . \end{cases}$$

This is precisely the concept of the *Ritz method*. As bilinear form $b(u,v)$ preserves its properties on the finite-dimensional level, the finite-dimensional minimization problem is equivalent with the approximate (abstract) VBVP, and the Ritz solution coincides simply with the Galerkin solution. The two methods are equivalent. The Galerkin procedure is more general, as it applies also to problems that do not admit the minimization principle.

2.2 Elementary convergence analysis

As we refine the mesh (mesh parameter $h \rightarrow 0$), dimension N of the approximate space grows to infinity. We expect the approximation error (function),

$$e_h = u - u_h \ ,$$

to get smaller. We can measure the error function using different norms. Although the choice of the norm is essentially up to us, some of them are naturally induced by the problem and the corresponding variational formulation. These include the energy norm, and the (equivalent with energy norm) Sobolev (semi)norm of the first order.

SPD case. Let u be the solution of the (abstract) VBVP and $w_h \in u_0 + V_h$ be any function (not necessary the approximate solution). In the case of a symmetric, positive-definite bilinear form, we

have the following algebraic identity.

$$\begin{aligned} 2\left(J(w_h) - J(u)\right) &= b(w_h, w_h) - 2l(w_h) - b(u,u) + 2l(u) \\ &= b(w_h, w_h) - b(u,u) + 2l(u - w_h) \\ &= b(w_h, w_h) - b(u,u) + 2b(u, u - w_h) \\ &= b(w_h, w_h) + b(u,u) - 2b(u, w_h) \\ &= b(u - w_h, u - w_h) \\ &= \|u - w_h\|_E^2 \,. \end{aligned} \tag{2.2.5}$$

The (twice) difference between the total potential energy corresponding to an approximation $w_h \in u_0 + V_h$ and *exact solution* u is equal to the approximation error (squared) measured in the energy norm. Minimization of the total potential energy for the approximate solution is *equivalent* to the minimization of the energy norm of the approximation error,

$$2\left(\min_{w_h \in u_0 + V_h} J(w_h) - J(u)\right) = \min_{w_h \in u_0 + V_h} \|u - w_h\|_E^2 \,.$$

Consequently, the approximate solution u_h is *the best solution* in terms of the energy norm,

$$\|u - u_h\|_E^2 = \min_{w_h \in u_0 + V_h} \|u - w_h\|_E^2 \,.$$

The quantity on the right-hand side is known as *the best approximation error* (wrt energy norm), and it is equal to the actual approximation error.

The result can be shown also in a different way using the orthogonality relation,

$$\begin{aligned} \|u - u_h\|_E^2 &= b(u - u_h, u - u_h) \\ &= b(u - u_h, u - w_h) + b(u - u_h, w_h - u_h) \\ &\leq \|u - u_h\|_E \|u - w_h\|_E \,, \end{aligned}$$

where $w_h \in u_0 + V_h$ is an arbitrary function. Notice that $w_h - u_h \in V_h$, so we can use the orthogonality relation. Notice also that a symmetric positive-definite bilinear form defines a scalar product, so we can use the Cauchy-Schwartz inequality (see [98], page 206).

Consequently, due to arbitrariness of w_h,

$$\|u - u_h\|_E \leq \min_{w_h \in u_0 + V_h} \|u - w_h\|_E \,,$$

and, as substitution $w_h = u_h$ is possible, we actually have the equality,

$$\|u - u_h\|_E = \min_{w_h \in u_0 + V_h} \|u - w_h\|_E \,. \tag{2.2.6}$$

A general PD case. In this case form $b(u,v)$ is no longer symmetric but it is still positive-definite on V,

$$b(v,v) > 0 \quad \forall v \in V, \quad v \neq 0\,.$$

The ellipticity condition (see discussion in the previous chapter) allows for assuming the stronger *V-coercivity condition*,

$$b(v,v) \geq \alpha \|v\|^2 \quad \forall v \in V\,. \tag{2.2.7}$$

where $\alpha > 0$ and $\|\cdot\|$ denotes either the energy, or the first order Sobolev (semi)norm. We also can assume that the bilinear form is continuous wrt the same norm,

$$|b(u,v)| \leq M\|v\|\,\|u\|\,. \tag{2.2.8}$$

THEOREM 5
(Cea's lemma)
Let bilinear form $b(u,v)$ be V-coercive and continuous. Let u and u_h denote the exact and the approximate solutions. Then

$$\|u - u_h\| \leq \frac{M}{\alpha} \min_{w_h \in u_0 + V_h} \|u - w_h\| \tag{2.2.9}$$

where M and α are the continuity and coercivity constants. ∎

Proof: We have

$$\begin{aligned} \alpha\|u-u_h\|^2 &\leq b(u-u_h, u-u_h) \\ &= b(u-w_h, u-w_h) + b(w_h-u_h, u-u_h) \\ &= b(u-w_h, u-u_h) \\ &\leq M\|u-w_h\|\,\|u-u_h\|\,. \end{aligned}$$

which implies the result. ∎

The result of Cea states that, in the discussed case, the approximation error is bounded by the best approximation error, premultiplied with a (possibly large but) *mesh-independent* constant. Thus, up to that constant, the Galerkin method still delivers the best approximation.

2.3 Comments

Notation. The notation for the bilinear form and the corresponding stiffness matrix definition suffers from the traditional notational inconsequence. In the definition of stiffness matrix

$$b_{ij} = b(e_j, e_j)\,,$$

the first index corresponds to the test function, and the second to the trial functions. In the bilinear form notation we have it the other way around - first trial and then test function. Some authors, e.g. Tom Hughes [80] have changed the abstract notation accordingly, and use $b(v,u)$ or $b(v;u)$ in place of $b(u,v)$.

Exercises

Exercise 2.1 Demonstrate that the Galerkin optimality result (2.2.6) generalizes to the complex case of a hermitian, positive definite form $b(u,v)$, i.e. for the solution of variational problem (8.3.9), with an arbitrary, antilinear form $l(v)$. Can you formulate an equivalent minimization problem in the complex case?

Chapter 3

1D hp Finite Element Method

In this chapter we introduce fundamentals of hp finite elements in 1D: construction of basis functions suitable for the Galerkin method using element shape functions, calculation of element matrices, assembling element matrices into global matrices, solution of the global system of equations, and the simplest postprocessing.

3.1 One-dimensional hp discretization

The Finite Element Method is a special case of the Galerkin method, and it differs from other versions of the Galerkin method with the way the basis functions are constructed. FE basis functions are constructed as unions of element shape functions, over a few adjacent elements, extended by zero to the rest of the domain. The support of a FE basis functions is always contained in a few elements only. In the h version of the method, where the convergence is attained by increasing the number of elements, this leads to sparse stiffness matrices.

We begin by partitioning the interval $(0, l)$ into N subintervals,

$$0 = x_0 < x_1 < \ldots < x_K < x_{K+1} < \ldots < x_N = l \, .$$

Each of the subintervals $(x_K, x_{K+1}), K = 1, \ldots, N$, will be called a *finite element*, and it will have two parameters associated with it: element length $h_K = x_{K+1} - x_K$, and element local polynomial order of approximation p_K. Convergence of the method will be achieved by either decreasing the element size, $h_K \to 0$, or by increasing the polynomial degree, $p_K \to \infty$. For each element[1] K, we introduce $p_K + 1$ polynomial shape functions that span the space $\mathcal{P}^{p_K}$ of polynomials of order less or equal p_K. To simplify the notation, we shall drop element index K, and talk about order p, with the understanding that p may change locally, from element to element.

[1] We shall use a 'dual' notation where K denotes also the actual subinterval, $K = (x_K, x_{K+1})$

1D master element of order p. Instead of introducing the shape functions on the *physical element* K directly, it is more convenient to consider first a generic, *master element*, define *master element shape functions*, and then use the idea of *parametric element* to define the shape functions for the physical element.

Geometrically, the 1D master element $\hat{K}$ coincides with the unit interval $(0,1)$. The *element space of shape functions* $X(\hat{K})$ is identified as polynomials of order p, i.e.,

$$X(\hat{K}) = \mathcal{P}^p(\hat{K}) \,.$$

One can introduce many particular bases that span polynomials of order p. The simplest choice is offered by the *Peano hierarchical shape functions* defined as follows,

$$\begin{aligned}
&\hat{\chi}_1(\xi) = 1-\xi \\
&\hat{\chi}_2(\xi) = \xi \\
&\hat{\chi}_3(\xi) = (1-\xi)\xi \\
&\hat{\chi}_l(\xi) = (1-\xi)\xi(2\xi-1)^{l-3},\ l = 4,\ldots,p+1 \,.
\end{aligned}$$

The functions admit a simple recursive formula,

$$\begin{aligned}
&\hat{\chi}_1(\xi) = 1-\xi \\
&\hat{\chi}_2(\xi) = \xi \\
&\hat{\chi}_3(\xi) = \hat{\chi}_1(\xi)\hat{\chi}_2(\xi) \\
&\hat{\chi}_l(\xi) = \hat{\chi}_{l-1}(\xi)(\hat{\chi}_2(\xi) - \hat{\chi}_1(\xi)),\ l = 4,\ldots,p+1 \,.
\end{aligned}$$

Note that, except for the first two linear functions, the remaining shape functions vanish at the element endpoints. For that reason they are frequently called the *bubble functions*.

Legendre polynomials. A much better choice for the 1D master element shape functions is the *integrated Legendre polynomials* [117]. Legendre polynomials $P_n(x), n = 0,1,\ldots$ are eigenvectors of the Legendre operator,

$$\frac{d}{dx}\left[(1-x^2)\frac{dP_n}{dx}\right] = n(n+1)P_n, \quad x \in (-1,1) \,.$$

Since the Legendre operator is self-adjoint in $L^2(-1,1)$, by the Sturm-Liouville theory, the Legendre polynomials are orthogonal both in L^2 and energy inner products, and form an L^2-orthogonal basis in $L^2(-1,1)$, i.e. each L^2 function $u(x)$ can be represented in terms of a (generalized) Fourier series,

$$u(x) = \sum_{n=0}^{\infty} u_n P_n(x) \,.$$

For a general function in L^2, the convergence is understood in the L^2 sense only. Coefficients u_n can be obtained by multiplying both sides of the expansion formula (in the sense of the L^2 product) with an arbitrary Legendre polynomial $P_n(x)$. The Legendre polynomials are usually scaled to one at $x = 1$ in which case,

$$\|P_n\|^2_{L^2} = \int_{-1}^{1} P_n^2(x)\, dx = \frac{2}{2n+1}\,.$$

The Legendre polynomials and their derivatives may be evaluated using the following recursion formulas,

$$\begin{aligned} P_0(x) &= 1, \\ P_1(x) &= x, \\ P_n(x) &=, \qquad n = 3, \dots \\ P_n'(x) &=, \qquad n = 3, \dots \\ P_n''(x) &=, \qquad n = 3, \dots \end{aligned}$$

Rescaling the Legendre polynomials to the interval $(0, 1)$,

$$l_n(t) = P_n(x), \quad x = -1 + 2t,$$

we shall define our second set of higher order shape functions as the integrals of the Legendre polynomials,

$$\begin{aligned} \hat{\chi}_1(\xi) &= 1 - \xi \\ \hat{\chi}_2(\xi) &= \xi \\ \hat{\chi}_n(\xi) &= \int_0^{\xi} l_{n-2}(t)\, dt, \quad n = 3, 4, \dots \end{aligned}$$

We also have an alternative representation,

$$\hat{\chi}_{2+j}(\xi) = (1-\xi)\xi \underbrace{\frac{(-1)}{j(j+1)} \frac{dl_j}{d\xi}(\xi)}_{\text{kernel function } \hat{\chi}_j^{kern}(\xi)} \quad , \quad j = 1, \dots\,. \tag{3.1.1}$$

The *kernel functions* $\hat{\chi}_j^{kern}$ will be useful for constructing shape functions for the triangular element. Notice that, contrary to the Peano shape functions, the first bubble function is

$$\hat{\chi}_3(\xi) = -(1-\xi)\xi\,,$$

i.e. the function is negative.

1D parametric element of arbitrary order. We consider now an arbitrary closed interval $K = [x_l, x_r] \subset [0, l]$. Element K is the image of master element $\hat{K}$ through a simple affine map which

may be conveniently defined using master element linear shape functions,

$$\begin{aligned} x_K(\xi) &= x_l\hat{\chi}_1(\xi) + x_r\hat{\chi}_2(\xi) \\ &= x_l(1-\xi) + x_r\xi \\ &= x_l + \xi(x_r - x_l) \\ &= x_l + \xi h_K \,. \end{aligned} \tag{3.1.2}$$

Here $h_K = x_r - x_l$ is the element length. The map is invertible with inverse x_K^{-1}. We define now the *element space of shape functions* $X(K)$ as the space of compositions of inverse x_K^{-1} and functions defined on the master element,

$$\begin{aligned} X(K) &= \{\hat{u} \circ x_K^{-1},\ \hat{u} \in X(\hat{K})\} \\ &= \{u(x) = \hat{u}(\xi) \text{ where } x_K(\xi) = x \text{ and } \hat{u} \in X(\hat{K})\ \} \,. \end{aligned}$$

Consequently, the element *shape functions* are defined as,

$$\chi_k(x) = \hat{\chi}_k(\xi) \text{ where } x_K(\xi) = x, \quad k = 1, \ldots, p+1 \,.$$

Since the inverse of an affine map is itself an affine map, the element shape functions are again polynomials of order p in physical coordinate x, and the corresponding space of element shape functions coincides with the space of polynomials,

$$X(K) = \mathcal{P}^p(K) \,.$$

Note that, for a more general map, the shape functions are no longer polynomials. In the case of the affine map, we speak about the *affine element*. It makes little sense to do it in 1D, but we may insist on using a more general transformation from the master to the physical element. It is convenient then to assume that map x_K is specified using the master element shape functions, i.e. it is also a polynomial of order p,

$$x_K(\xi) = \sum_{j=1}^{p+1} x_{Kj}\hat{\chi}(\xi) \,.$$

In such a case we talk about an *isoparametric element* [2]. Coefficients x_{Kj} will be identified as the *geometry degrees of freedom (g.d.o.f.)*. Note that only the first two have then the interpretation of coordinates of the endpoints of element K.

1D hp finite element space. Let now interval $(0, l)$ be covered with a FE mesh consisting of disjoint elements $K = (x_{K-1}, x_K)$. The element endpoints with coordinates $0 = x_0 < x_1 < \ldots < x_N < x_{N+1} = l$ will be called the *vertex nodes*. We define the 1D hp *finite element space* X_h [3] as

[2] Same shape functions are used to approximate both geometry and solution

[3] One should really use a symbol X_{hp} as the discretization depends upon the element size $h = h_K$ and order of approximation $p = p_K$

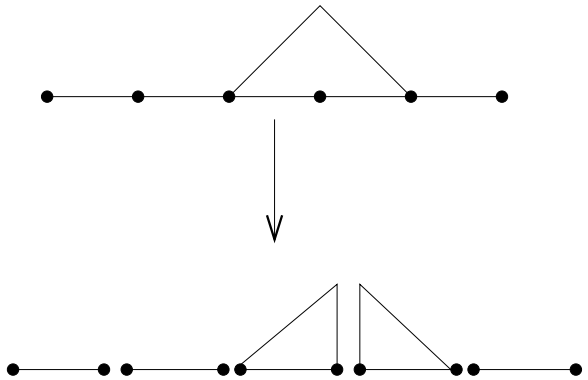


Figure 3.1: Construction of the vertex nodes basis functions

the collection of all functions that are globally continuous, and whose restrictions to element K live in the element space of shape functions,

$$X_h = \{u_h(x) \ : \ u \text{ is continuous and } u|_K \in X(K) \text{ for every element } K \ \} \ .$$

The global basis functions are then classified into two groups:

- vertex nodes basis functions,
- middle nodes *bubble* basis functions.

Basis function corresponding to a vertex node x_K is defined as the union of the two adjacent element shape functions corresponding to the common vertex and zero elsewhere. The construction is illustrated in Fig 3.1. The construction of the middle nodes basis functions is much easier. As the element middle nodes shape functions vanish at the element endpoints, we need only to extend them by the zero function elsewhere. The support of a vertex node basis function extends over the two adjacent elements, whereas for a bubble function it is restricted just to one element.

3.2 Assembling element matrices into global matrices

Let e_i and e_j be two Galerkin basis functions. The corresponding entry in the global stiffness matrix is defined as,

$$B_{ij} = b(e_j, e_i) = \int_0^1 \left\{ a(x) \frac{de_j}{dx} \frac{de_i}{dx} + b(x) \frac{de_j}{dx} e_i + c(x) e_j e_i \right\} \ dx + \beta e_j(l) e_i(l) \ .$$

The support of any FE basis function extends only over two elements for vertex basis function, and just one element for middle node basis functions. This means that the integration in the formula above takes place at most over two elements. In particular, if the supports of basis functions e_i and

e_j do not overlap, the corresponding entry in the global stiffness matrix is simply zero. The FE construction of basis functions produces *sparse* stiffness matrix.

To illustrate the idea of assembling element matrices, let us assume that $e_i = e_j$ is the basis function corresponding to a K-th vertex node shown in Fig. 3.1. The support of the basis function extends over the two neighboring elements only and, therefore, the integral in the formula above can be reduced to the sum of two corresponding element integrals,

$$B_{ii} = b(e_i, e_i) = b_{K-1}(e_i, e_i) + b_K(e_i, e_i) \, ,$$

where we have introduced the *element bilinear form* defined as,

$$b_K(u,v) = \begin{cases} \displaystyle\int_{x_K}^{x_{K+1}} \left\{ a(x) \frac{du}{dx}\frac{dv}{dx} + b(x)\frac{du}{dx} v + c(x) uv \right\} dx + \beta u(l) v(l) & \text{if } x_{K+1} = l \\ \displaystyle\int_{x_K}^{x_{K+1}} \left\{ a(x) \frac{du}{dx}\frac{dv}{dx} + b(x)\frac{du}{dx} v + c(x) uv \right\} dx & \text{otherwise .} \end{cases}$$

The corresponding *element stiffness matrix* is defined now as,

$$B_{K,kl} = b_K(\chi_l, \chi_k) \, .$$

Please notice the systematic notation. Formula for the global stiffness matrix is obtained by replacing solution u and test function v in the bilinear form with basis functions e_j, e_i, the element bilinear form is obtained by restricting the integration to the particular element and, finally, the element stiffness matrix is obtained by using the element trial and test shape functions in place of the solution and the test function. Notice also that the boundary term corresponding to Cauchy BC, contributes only to the very last element in the mesh.

Now, the considered vertex basis function is obtained by gluing together the *second* shape function from element $K-1$ with the *first* shape function from element K. This implies that,

$$B_{ii} = B_{K,22} + B_{K+1,11} \, .$$

We have precisely the same result for the load vector.

$$l_i = l_{K,2} + l_{K+1,1} \, ,$$

where the element linear form is defined as,

$$l_K(v) = \begin{cases} \displaystyle\int_{x_K}^{x_{K+1}} f v \, dx + g v(l) & \text{if } x_{K+1} = l \\ \displaystyle\int_{x_K}^{x_{K+1}} f v \, dx & \text{otherwise ,} \end{cases}$$

and *element load vector* is obtained by substituting for test function v k-th shape function χ_k,

$$l_{K,k} = l_K(\chi_k) \, .$$

The discussed case of a vertex basis function is actually the most complicated one. In the case of two different basis functions e_k, e_l, or middle node basis functions, in 1D, there will be always one contributing element only.

In order to assemble a contribution corresponding to the i-th basis function e_i, we need to know contributing elements K, and the corresponding number of the element shape function $k = k(K, i)$. This is if we decide to use the *recipient strategy*, i.e. we fix our eyes on the receiving i-th basis function and try to identify the corresponding element contributions.

It is probably the most profound characteristics of coding the finite element method that we decide to use the *donor strategy* instead . Given an element K, for each of its shape functions $\chi_{K,k}$, we identify the corresponding global basis function,

$$i = i(K, k) \, ,$$

and assemble the corresponding element matrices into the global matrices. The function above is known as the *d.o.f. connectivities for element* K.

The fundamental assembling procedure looks now as follows.

```
initiate global load vector l and global stiffness matrix B
  with zeros
for each element K in the mesh
  for each element shape function k1
    determine d.o.f.  connectivity i = i_{K,k1}
    l_i = l_i + l_{K,k1}
    for each element shape function k2
      determine d.o.f.  connectivity j = j_{K,k2}
        B_{ij} = B_{ij} + b_{K,k1,k2}
    end of the second loop through element K shape functions
  end of the first loop through element shape functions
end of loop through elements
```

In the case of the hp methods, it is convenient to introduce a *nodal connectivity* information. For each element K, we store the corresponding numbers of the two vertex nodes and the element middle node. Once the nodal connectivities are known, it is easy to use them to reconstruct the d.o.f. connectivities. Given an *ordering of nodes*, and an ordering of d.o.f. for each node, we establish the corresponding ordering of d.o.f. by introducing an array $nbij(nod)$ that, for each node nod, stores the corresponding number of the first d.o.f. associated with the node.

```
initiate global d.o.f.  counter i = 0
for each nod in the mesh
```

```
    store the location of the first d.o.f. for the node:
    nbij(nod) = i + 1
    update the d.o.f. counter: i = i + ndof(nod)
  end of loop through the nodes
```

Here $ndof(nod)$ denotes number of d.o.f. associated with node nod. The element d.o.f. connectivities are then computed using the following algorithm.

```
initiate element d.o.f. counter k = 0
for each node l of element K
determine the nodal connectivity nod = nod(K, l)
  for each d.o.f. j of node nod
    increment d.o.f. counter k = k + 1
    compute the corresponding d.o.f. connectivity:
    i = i(K, k) = nbij(nod) + j − 1
  end of loop through nodal d.o.f.
end of loop through nodes of element K
```

Notice that all the discussed procedures are in fact quite general, and will apply to the multi-dimensional case as well. The logic of storing information for nodes rather than nodal d.o.f. is fundamental for coding hp elements.

3.3 Computing the element matrices

Integrals defining the element stiffness matrix are computed using *Gaussian quadrature*. Given an integrand $f(\xi)$ defined on the master element, $\xi \in (0,1)$, we approximate the corresponding integral as follows,

$$\int_0^1 f(\xi)\, d\xi \approx \sum_{l=1}^{L} x(\xi_l) w_l \, .$$

Points $x_l \in (0,1)$ are the *Gaussian quadrature points* with corresponding weights w_l. With L points and L corresponding weights, the points are selected in such a way, that the approximate formula above becomes exact for polynomials of order $2L-1$. The condition defines $2L$ equations for L unknown quadrature points and L corresponding weights. The equations are nonlinear in ξ_l and linear in w_l. One does not solve, however, the nonlinear system to determine the Gaussian quadrature points. It can be proved (see e.g. [87], page 528) that the Gaussian points of quadrature of order L are roots of Chebyshev polynomials of order L. Once the points are known, one can solve easily the remaining linear system for the weights.

Gaussian quadrature data is not hierarchical. Changing the order of integration implies changing all quadrature points and weights, not adding an extra few. Given an element of order p, we use

quadrature of order $p+1$. This guarantees that, in presence of element-wise constant material data, the element stiffness matrix will be computed exactly (up to machine precision). Indeed, the highest order polynomial contribution comes from the zero order term $\int cuv\ dx$. With constant coefficient c, $u, v \in \mathcal{P}^p$, we need to integrate polynomial of order $2p$, and for that we need $p+1$ Gaussian quadrature points. If the zero order term vanishes, quadrature of order p will be sufficient. For non constant material data, the corresponding integrals will be computed only approximately. This will also apply to integrating the load vector with a general source term $f(x)$.

Frequently, when studying convergence, we deal with very irregular source terms, and then we have to resort to the use of an adaptive quadrature to be discussed later. For most practical problems, however, data *are piecewise constant* and the Gaussian quadrature is sufficient.

The numerical integration is performed always on the master element. This means that we change variables from x to ξ, before we integrate. The procedure is consistent with the fact that physical element shape functions are defined through the master element shape functions. We start with the load vector,

$$\int_{x_L}^{x_R} f(x)\chi_i(x)\ dx = \int_0^1 f(x(\xi))\hat{\chi}_i(\xi)\frac{dx}{d\xi}\ d\xi = \int_0^1 \hat{f}(\xi)\hat{\chi}_i(\xi)h\ d\xi\ ,$$

where $\hat{f}(\xi)$ denote the composition of $x(\xi)$ and $f(x)$, and $h = h_K$ is the element length. Similarly, for the stiffness matrix,

$$\begin{aligned}
&\int_{x_L}^{x_R} \{a(x)\frac{d\chi_j}{dx}\frac{d\chi_i}{dx} + b(x)\frac{d\chi_j}{dx}\chi_i + c(x)\chi_j\chi_i\}\ dx \\
&= \int_0^1 \{a(x(\xi))\frac{d\hat{\chi}_j}{d\xi}\frac{d\hat{\chi}_i}{d\xi}(\frac{d\xi}{dx})^2 + b(x(\xi))\frac{d\hat{\chi}_j}{d\xi}\frac{d\xi}{dx}\hat{\chi}_i + c(x(\xi))\hat{\chi}_j\hat{\chi}_i\}\frac{dx}{d\xi}\ d\xi \\
&= \int_0^1 \{\hat{a}(\xi)\frac{d\hat{\chi}_j}{d\xi}\frac{d\hat{\chi}_i}{d\xi}h^{-2} + \hat{b}(\xi)\frac{d\hat{\chi}_j}{dx}h^{-1}\hat{\chi}_i + \hat{c}(\xi)\hat{\chi}_j\hat{\chi}_i\}h\ d\xi\ ,
\end{aligned}$$

with the same notation for the compositions of $x(\xi)$ and the material coefficients.

Switching to the numerical integration, we get,

$$\begin{aligned}
&\int_{x_L}^{x_R} f(x)\chi_i(x)\ dx \approx \sum_l^L \hat{f}(\xi_l)\hat{\chi}_i(\xi_l)hw_l \\
&\int_{x_L}^{x_R} \{a(x)\frac{d\chi_j}{d\xi}\frac{d\chi_i}{dx} + b(x)\frac{d\chi_j}{dx}v + c(x)uv\}\ dx \\
&\approx \sum_l^L \left\{\hat{a}(\xi_l)\frac{d\hat{\chi}_j}{d\xi}(\xi_l)\frac{d\hat{\chi}_i}{d\xi}(\xi_l)h^{-2} + \hat{b}(\xi_l)\frac{d\hat{\chi}_j}{dx}(\xi_l)h^{-1}\hat{\chi}_i(\xi_l) + \hat{c}(\xi_l)\hat{\chi}_j(\xi_l)\hat{\chi}_i(\xi_l)\right\} h\ w_l\ .
\end{aligned} \tag{3.3.3}$$

We are now ready to present the algorithm for calculating the element matrices.

Step 0: `Prepare element data that do not vary with integration points`

get order for the element p
get vertex nodes coordinates x_L, x_R
Step 1: Compute element integrals contributions
initiate element load vector $\boldsymbol{l}$ and element stiffness matrix $\boldsymbol{B}$
  with zeros
for each integration point $\boldsymbol{\xi}_l$
  compute values of master element shape functions and their
    derivatives wrt master element coordinates, $\hat{\chi}_k(\xi), d\hat{\chi}_k/d\xi$
  use formula (3.1.2) to compute the corresponding physical
    coordinate x_l of the integration point, and its derivative
    wrt master element coordinate $dx/d\xi$
  get material data a, b, c and load data f at the point
  for each element shape function i
    use (3.3.3) to accumulate for the load vector:
    $l_i = l_i + \hat{f} * \hat{\chi}_i * w_l$
    for each element shape function j
      use (3.3.3) to accumulate for the stiffness matrix:
      $B_{ij} = B_{ij} + \left\{ \hat{a} \frac{d\hat{\chi}_j}{d\xi} \frac{d\hat{\chi}_i}{d\xi} h^{-2} + \hat{b} \frac{d\hat{\chi}_j}{dx} h^{-1} \hat{\chi}_i + \hat{c} \hat{\chi}_j \hat{\chi}_i \right\} h\, w_l$
    end of the second loop through element shape functions
  end of the first loop through element shape functions
end of loop through integration points
Step 2: Compute boundary contributions
if the element is adjacent to the Cauchy boundary
  evaluate element shape functions $\hat{\chi}_j$ at $\xi = 1$
  for each element shape function i
    accumulate for the load vector:
    $l_i = l_i + \gamma \hat{\chi}_i$
    for each element shape function j
      accumulate for the stiffness matrix:
      $B_{ij} = B_{ij} + \beta \hat{\chi}_j \hat{\chi}_i$
    end of second loop through shape functions
  end of first loop through shape functions
endif

Notice that in the last part both loops through the shape functions reduce to the second vertex shape function only - this is the only shape function that does not vanish at $\xi = 1$, i.e. $x = l$. Our algorithm anticipates the 2D case where there will be more shape functions contributing to

the boundary term. In the case of the Neumann condition, there is no contribution to the stiffness matrix, and the second loop through the shape functions is dropped. An analogous algorithm holds for $x = 0$ endpoint.

3.4 Accounting for the Dirichlet BC

Incorporation of Dirichlet BC's is based on the modified variational formulation (8.3.10). The modifications can be done either at the element level, i.e. before assembling the global matrices, or at the global level, after the assembling. We prefer the first solution because it gives us more flexibility to interface with various linear solvers.

The first step is to construct a lift of the Dirichlet BC data. This can be done in many different ways but the simplest solution is to use the first element shape function premultiplied with constant u_0,

$$\tilde{u_0}(x) = u_0 e_1(x) = u_0 \chi_1(x) \, .$$

Here $\chi_1(x)$ denotes the (first) left vertex shape function for the (first) element adjacent to the boundary which coincides (up to the zero extension) with the (first) basis function $e_1(x)$.

Once we have selected the lift, we need to modify the load vector. This is done for the first element only. Other elements are not affected by the Dirichlet condition. We have,

$$\begin{aligned} l_{K,j}^{mod} &= l_K^{mod}(\chi_j) \\ &= l_K(\chi_j) - b_K(\tilde{u_0}, \chi_j) \\ &= l_K(\chi_j) - u_0 b_K(\chi_1, \chi_j) \\ &= l_{K,j} - u_0 B_{K,j1}, \quad j = 2, p+1 \, . \end{aligned}$$

Here $K = 1$ is the first element.

Notice that the load vector modification has to be done starting with the second shape function. We do not test with the first shape function, and the corresponding contribution is also missing from the FE approximation of function w in the modified variational formulation (8.3.10). In practice, it is convenient to keep the same logic for all elements, no matter whether they are, or they are not adjacent to the Dirichlet boundary. This means that we have to extend the modified (first) element matrices in such a way that we can 'trick' the solver of linear system to return the Dirichlet data at the first vertex node. This is done in the following way,

$$B_{K,ij} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & B_{K,22} & B_{K,23} & \dots & B_{K,2n} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & B_{K,n2} & B_{K,n3} & \dots & B_{K,nn} \end{pmatrix} \qquad l_{K,j} = \begin{pmatrix} u_0 \\ l_{K,2} - u_0 B_{K,21} \\ \vdots \\ l_{K,n} - u_0 B_{K,n1} \end{pmatrix} \, .$$

Here $K = 1$, and $n = p + 1$ denotes the number of d.o.f. for the first element.

Notice that the first equation is fully decoupled from the remaining equations. Also, if the original stiffness matrix is symmetric (the problem is symmetric), the modified matrix remains symmetric as well.

3.5 A summary

A classical FE code consists of three main parts: a *preprocessor*, *processor* and a *postprocessor*.

In the preprocesor stage, we set up the problem to solve: read in geometry and material data, initiate data structure arrays, generate the mesh, i.e. compute and store geometry d.o.f. (nodal coordinates in 1D).

The processing stage consists of forming global matrices and solving the resulting system of linear equations. The global matrices are assembled from element matrices. We loop through all elements in the mesh, compute element matrices, modify them to account for Dirichlet BC, and use element d.o.f. connectivities to assemble them into the global matrices. We solve then the global system of equations using a direct, or an iterative linear equations solver. We store the computed solution in data structure arrays.

In the final, postprocessing stage, we output the solution either in a graphical form or by computing quantities of interest involving the solution and its derivatives. In mechanical terms those could be displacements, strains, stresses, etc. When postprocessing the solution, we use the same strategy as for the computation of element matrices. Instead of inquiring for the solution at a point specified in terms of its physical coordinate, we specify the corresponding element K and master element coordinate ξ of the point. Value of the solution at the point is then evaluated using the element solution d.o.f. and the element shape functions,

$$u(x_K(\xi)) = \sum_{k}^{p+1} u_{K,k}\hat{\chi}_k(\xi)\,. \tag{3.5.4}$$

Here $u_{K,k}$ denote element solution d.o.f. stored in data structure arrays that are recovered using element d.o.f. connectivities.

3.6 Assignment 1: A dry run

Before we start talking about the actual code, we recommend first going through the FE computations on paper. This may look like an unnecessary burden but it is very essential. We will try to make it both general enough, and very specific at the same time.

We want to solve the following interface problem.

$$\begin{cases} -(u')' + u' = f_1(x) & x \in (0, \frac{1}{2}) \\ -(2u')' + u = f_2(x) & x \in (\frac{1}{2}, 1) \\ u(0) = u_0, \quad 2u'(1) + u(1) = \gamma_1 & \\ u(\frac{1}{2}-) = u(\frac{1}{2}+), \quad u'(\frac{1}{2}-) = 2u'(\frac{1}{2}+) \, . & \end{cases}$$

We have used $u(\frac{1}{2}-), u(\frac{1}{2}+)$ notation to denote the left- and right-side limits. Notice that we have specified the material data but we have not specified the load data.

Step 1: Verify that the problem falls into the class of problems being discussed, i.e. that the coefficients in the differential equations and the boundary conditions are consistent. Write down the variational formulation for the problem.

Instead of setting up an ad hoc load data and solving first the problem analytically, we shall use the method of *manufactured solutions*. We simply assume that our exact solution is of the following form,

$$u(x) = \begin{cases} -(x - \frac{1}{2})^2 + 1 & x \in (0, \frac{1}{2}) \\ 1 & x \in (\frac{1}{2}, 1) \, . \end{cases}$$

Check that the function satisfies the interface conditions.

Step 2: Use the differential equation and the boundary conditions to find the load, i.e. $f_1(x), f_2(x), \gamma, u_0$, corresponding to the assumed solution. We shall use now these data to set up a FE approximation.

Step 3: Use a finite element mesh consisting of four uniform elements of length $h = 1/4$, but with varying orders $p_1 = 3, p_2 = 2, p_3 = 1, p_4 = 2$.

Step 4: Compute analytically the element matrices. You may integrate directly in terms of the physical coordinate or follow the discussed change of variables (the first option is simpler if you do it on paper).

Step 5: Modify the first element matrices to account for the Dirichlet BC.

Step 6: Assembly the global matrices. Use a calculator to solve the resulting system of equations.

Step 7: Use the obtained values and a calculator to plot the FE solution at selected values, say five points per element, and compare with the exact (manufactured) solution. Do they coincide? Should they?

Step 8: Given a function that 'lives' in the FE space, can you compute its FE representation, i.e. identify the corresponding values of d.o.f. ? Can you do it for our manufactured solution? Can you think of a systematic way for doing this for *any* solution?

As trivial as this exercise may look for you, it will help you to understand the Fortran imple-

mentation discussed next.

Exercises

Exercise 3.1 A minimum modification of element matrices to accommodate for Dirichlet BC's consists of decoupling the equation corresponding to the Dirichlet d.o.f. from the rest of the system and setting the corresponding load vector value to the boundary condition data. For the case discussed in section 3.4, we have,

$$B_{K,ij} = \begin{pmatrix} 1 & 0 & 0 & \dots, 0 \\ B_{K,21} & B_{K,22} & B_{K,23} & \dots, B_{K,2n} \\ \vdots & \vdots & \vdots & \vdots \\ B_{K,n1} & B_{K,n1} & B_{K,n2} & \dots B_{K,nn} \end{pmatrix} \qquad l_{K,j} = \begin{pmatrix} u_0 \\ l_{K,2} \\ \vdots \\ l_{K,n} \end{pmatrix} .$$

The resulting, modified stiffness matrix, though, is always non-symmetric, even if the original bilinear form is symmetric. Discuss why the two procedures are equivalent.

Exercise 3.2 Demonstrate that the Legendre operator is self-adjoint in $L^2(-1,1)$.

Exercise 3.3 Prove formula (3.1.1). *Hint:* Use the fact that the Legendre polynomials are eigenfunctions of the Legendre operator.

Chapter 4

1Dhp Code

In this chapter, we discuss the fundamentals of our 1D code: data structure of the code, its organization, graphics, element routine, and discuss the second assignment.

4.1 Setting up the $1Dhp$ code

We shall assume that the reader is familiar with fundamentals of LINUX (UNIX) operating system and Fortran 90 language and compilers, see e.g. [90].

Set up a directory, copy the enclosed *tar1Dhp.gz* file into it, decompress it using command 'gunzip tar1Dhp.gz', and untar it using command 'tar - xvf tar1Dhp'. We are ready to compute. Your directory should contain the following subdirectories.

- *adapt_h* - routines executing automatic h-refinements,
- *adapt_hp* - routines executing automatic hp-refinements,
- *blas* - basic linear algebra routines,
- *commons* - system common blocks,
- *constr_util* - constrained approximation utilities,
- *data* - routines specifying the material and load data,
- *datstrs* - data structure routines,
- *elem_util* - element utilities,
- *files* - system files,
- *frontsol* - a frontal solver,

- *gcommons* - graphics common blocks,
- *graph_1D* - the 1D code graphics programs,
- *graph_interf* - routines interfacing Xwindows with the code graphics routines,
- *graph_util* - graphics utilities,
- *laplace* - element routines for the model problem,
- *main* - the main program,
- *meshgen* - initial mesh generation routines,
- *meshmods* - mesh modification routines,
- *module* - system moduli,
- *solver1* - interface with the frontal solver,
- *utilities* - general utilities routines.

Additionally, you will find file *head* with a standard subroutine header, *makefile*, and a directory *documentation*.

The *makefile* Please modify the makefile replacing the *pgf90*[1] compiler with your compiler. Make sure that the paths specifying the access to the X-windows library are correct. Type *make prec* to obtain a real version of the code. Type then the *make r* command. If everything goes well, you should now find file *a.out* in your directory.

Data types. The code uses only double precision reals, and long integers. All data are specified *implicitly* in *commons/syscom.blk* that should be included in all routines operating with real numbers. The code can also be preprocessed to yield a complex version. To get it, set *COMPLEX = 1* in the makefile, type *make prec*, and then *make z* to obtain the complex version of the code. Be sure to 'touch' all the files using commands *touch */*.F* and *touch */*.f* before recompiling the code. The complex version of the code will be used only in Chapter 7. You do not need to worry about this for now.

With a minimal change, the code can be switched to a complex mode. We will discuss it later.

[1] Portland Group Fortran 90

The coding protocol. Please read *documentation/ coding_protocol* that explains the basic rules, we have attempted to follow when writing the code. We shall use the same rules in the 2D codes. There is no global debugging support in the 1D code. Instead, each routine comes with a local debugging print flag *iprint*. By resetting it to one (sometimes a different value, sometimes also more than one printing flag will be used), you may activate the control prints in the routine.

4.2 Fundamentals

In this chapter we discuss the most basic part of the code that allows for computing with a preset initial mesh. The simple 1D mesh generator provided with the code allows for starting with a mesh of N uniform elements of same order $p = 1, \ldots, 9$. The max order of approximation $NPMAX = 9$ is set in *commons/cpmax.blk* and can be changed to a smaller value but, due to design assumptions, cannot exceed $NPMAX = 9$.

Data structure module. We start with module *module/data_structure1D* that contains the core of the code - data structure arrays. We expect the reader to open the discussed routines with a text editor and follow the comments in the code. We will not reproduce the code in this text. Module *module/data_structure1D* contains the definition of three fundamental objects used in the code:

initial mesh element: element connectivities contain two vertex nodes numbers, and the middle node number, element neighbors contain initial mesh left and right neighbors of the element; zero indicates that the element is adjacent to the boundary.

vertex node: boundary condition flag, the node coordinate, father of the node.

middle node: order of the node, father and sons of the node, solution d.o.f.

The info on father and sons will be necessary to support h-refinements. We shall ignore it in this section.

System files, IO. Besides the keyboard, the code uses a number of standard files.

- *files/input:* standard input file, contains data to generate an initial mesh,
- *files/output:* standard output file, for echo prints from the code, and for outputting content of the data structure arrays.
- *files/control:* includes a few global control parameters that can be set up by the user,
- *files/result:* to output convergence history when running the code in an adaptive mode; the results are then used to plot the convergence curve,

- *files/dump:* to output the content of data structure arrays for a future use, e.g. for restarting the code, or extra postprocessing.

Except for the *dump* file, all files are open in *utilities/opfil*. Each file is assigned a number stored in *commons/cinout.blk*.

The main driver. Program *main/main* contains the main driver for the code. Upon opening the system files, and reading the control parameters, the code precomputes a number of arrays necessary for mesh optimization, and generates the initial mesh using data specified by the user in *files/input*. The main menu contains a number of options that we will discuss. We shall focus on a graphical postprocessing first. Next we will discuss the most important part of our second assignment: writing your own processor. The code contains already a more sophisticated processor built around a frontal solver. This will help you to debug your processor - at any time you will have a chance to compare your results with ours. The two critical items in the menu: automatic h- and hp-refinements, will be discussed in the next chapter.

Initial mesh generator. Our preprocessor consists of a single routine - *meshgen/meshgen*. The routine reads initial mesh parameters specified in file *files/input:* anticipated maximum number of vertex and middle nodes, number of elements in the initial mesh, uniform order of approximation for the initial mesh elements, interval length l, and BC's flags:

1: Dirichlet BC,

2: Neumann BC,

3: Cauchy BC.

. The three data structure arrays *ELEMS, NVERS, NODES* containing initial mesh elements, vertex, and middle nodes, discussed earlier, are allocated dynamically in *meshgen/allocds*. Routine *meshgen/deallocds* deallocates the arrays. Finally, we generate the initial mesh, i.e. initiate the data structure arrays with a relevant data. We use *linked lists* to link free entries in arrays *NVERS* and *NODES*, necessary to support future refinements. Notice the use of attributes *NODES(nod)%father* and *NVERS(nod)%father* to store the links. By inserting *call result* anywhere in the code, you can print out (to file *files/output*) a current content of the data structure arrays. This is especially useful when debugging new routines added to the code.

4.3 Graphics

The code contains three graphics routines: graphics driver *graph_1D/graph* from which you can call the actual graphics routine - *graph_1D/graph1D*, and *graph_1D/rates* - a routine to plot the

convergence curve. Beginning with the graphics is a good way to get familiar with the fundamental data structure supporting routines and learn the logic of the code. Driver *graph_1D/graph* initiates a graphics interface with Xwindows, contained in *graph_interf* directory. Open now *graph_1D/graph1D*, please. The routine contains calls to the most important graphics interface routines:

graph_interf/selwin - opens a new window,
graph_interf/drawline - draws a segment of line,
graph_interf/fillpoly - fills a polygon with a color,
graph_interf/symbol - writes out a sequence of characters,
graph_interf/xmousepos - reads the current position of the mouse,
graph_interf/closwind - closes the window.

We shall not discuss the content of these routines, and the interface (written by my old friend - Tadeusz Liszka) with Xwindows written in C (*graph_interf/xlibf.c*). Instead, we will only illustrate how to use them. The graphical interface is initiated in *graph_util/initwin*. The system supports black & white (B/W), grey scale and color graphics, and grey scale and color Postcript echo. Colors are initiated in *graph_util/colorset* with the number of supported colors *NR_COLORS* and corresponding color codes stored in *gcommons/coltab.blk*. Parameter *NR_COLORS* is set to 200. If you work with an older monitor, your graphics card may not support so many colors simultaneously, and the code will crash in routine *graph_util/colorset*. If this happens (you will get an error message from Xwindows), activate the control print in *graph_util/colorset*, check how many colors are supported simultaneously on your screen, modify parameter *NR_COLORS* in *gcommons/coltab.blk*, and recompile the code.

The discussed graphics routine performs the following tasks.

- It draws a color representation of the FE mesh, with different colors corresponding to different orders of approximation. A reference color scale is drawn (vertically) along the right margin.
- It plots the approximate solution, and the exact solution, if available.
- It enables an interactive mesh modification. By clicking the mouse on, under or above the graphical representation of an element, one can break it, or decrease and increase its order.

All graphical operations have to be done within a range specified by *screen coordinates* defined in pixels. Consult routines *graph_util/selwin* selecting a window, and *graph_interf/openwind* with a (hardwired) call to Xwindows interface routine *xgopw*[2] opening the window. If you want to change the size of your window to one that would better fit your computer screen, modify parameters *iwindl,iwindh* in *graph_util/selwin* and the corresponding call to *xgopw* in *graph_interf/openwind*.

Parameters necessary to convert physical coordinates into screen coordinates are stored in *gcommons/gwind.blk*. Their meaning is explained in Fig. 4.1.

[2]The names of all Xwindows interface routines begin with letter 'x'

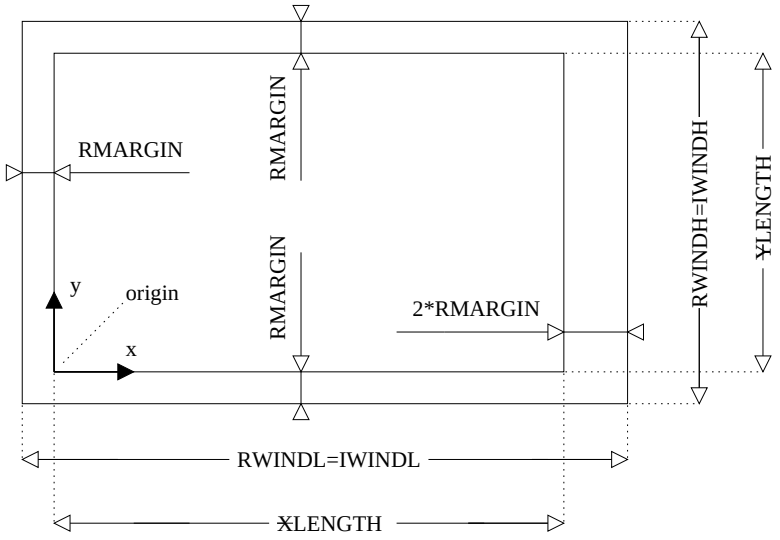


Figure 4.1: Screen coordinates

We will now discuss routine *graph_1D/graph1D* in more detail. We shall proceed in several steps.

Step 1: Determine the extreme values of the approximate and exact (if available) solutions to display. The values will be used to scale the graph and convert physical coordinates into the screen coordinates. We begin by looping through all active elements in the mesh. This is done be executing the following lines.

```
mdle=0
do iel=1,NRELES
  call nelcon(mdle, mdle)
  ...
enddo
```

In a traditional FE code, you will find a more straightforward use of the 'do' loop.

```
do iel=1,NRELES
  ...
enddo
```

In both cases, *iel* is the sequential element number, denumerating active elements in the mesh. Our version, however, anticipates adaptivity. In process of refining or derefining the mesh,

new elements will be added and some existing elements may be deleted. Each element, active or not, has precisely one, unique middle node *mdle*, and we will identify the elements simply with their middle nodes numbers. It looks weird, doesn't it? Indeed, in 1D, the choice does not seem to be natural. You have to wait to learn about the 2D and 3D codes to fully appreciate the logic behind the choice. Routine *datstrs/nelcon* takes you through all active elements in the mesh. We shall discuss it in detail in the next chapter. For a given element *mdle* in the mesh (again, *mdle* is in fact the middle node of the element), *datstrs/nelcon* returns the (middle node of) the next element, identified with the same variable *mdle*. You have to initiate *mdle* with zero and, upon visiting the last active element in the mesh, the routine will return zero as well. For the initial mesh, the routine returns simply the next number, and our construction is equivalent to the usual loop listed above with *iel = mdle*.

We are now in an element *mdle*. We need to access its vertex nodes coordinates and solution d.o.f. For the initial mesh elements, we simply have to reach the data structure arrays. With *mdle = iel*, *ELEMS(iel)%nodes(1:3)* contains the element vertex nodes *nvert1,nvert2* and middle node *mdle* (redundant, we already know it).

The vertex node coordinates can be found then in
NVERS(nvert1)%coord and *NVERS(nvert2)%coord*.

The solution d.o.f. are :
NVERS(nvert1)%zdofs(1),
NVERS(nvert2)%zdofs(1), and
NODES(mdle)%zdofs(1,1:ndof),
where n = *ndof* is equal to the number of d.o.f. associated with the middle node, $n = p - 1$. Order p is stored in *NODES(mdle)%order*.

This is exactly what the three routines *datstrs/nodcor*, *datstrs/solelm* and *datstrs/findap* will do for you, if *mdle* is the initial mesh element - they will simply reach to the data structure arrays, and return the necessary information. The situation will again change for refined meshes, and the discussed coding anticipates that more general information. Thus, instead of reading the data structure arrays yourself, we ask you to simply call these three routines. We shall use the same logic and the same routine names in our 2D and 3D codes.

OK, so now we know the element vertex coordinates, its order of approximation p and its solution d.o.f. We shall sample now the solution at a discrete set of points distributed over the element. When drawing later the solution, we shall simply approximate it with a piecewise linear approximation (this is exactly what the visualization people do) using the solution values at the sampling points. Number of sampling points equals number of segments *nsegm* + 1, and it is preset in the beginning of the routine. Given now the master element coordinate ξ = *xi* of a sampling point, we use (3.1.2) and (3.5.4) to evaluate the corresponding physical coordinate *x* and approximate solution value *zval*, and update the extreme values of the so-

lution. If the exact solution is known, we evaluate it as well (routine *data/exact*) and use the exact solution value to update the global minimum and maximum as well. In this way, we make sure that the graphs of both approximate and exact solutions will fit into the screen.

Step 2: We draw axes of the system of coordinates using the screen coordinates and the p-box, representing orders of approximation (routine *graph_util/dpbox*).

Step 3: We draw the approximate and exact solutions and a graphical representation of the mesh. We repeat exactly the same procedure as in Step 1. Pay attention to the rescaling, converting physical coordinates x and solution values into screen coordinates necessary for calling the graphics drawing routines. This is where we use the extreme values of the solution computed in Step 1, and length l = *XL* of the interval stored in *commons/cdata.blk*.

Step 4: You are given a chance to add two lines of comments to your text to comment on the approximate and exact solution plots. You are also asked to specify the location of the comments in multiples of the margin. As your solutions may take weird forms, it is difficult to anticipate in general where to put those comments.

Step 5: Interactive refinements. By clicking the mouse on the left margin, you may exit now. By clicking the mouse within the box representing an element, you will call routine *mesh-mods/break* that will break the element, i.e. make all necessary changes in the data structure arrays corresponding to this operation. Similarly, by clicking above or below an element box, you may increase or decrease its order (by one only). Do not attempt to study the refinements now. We will systematically discuss it in the next chapter. Make yourself familiar, however, with routine *xmousepos* (from *graph_interf/xlibf.c*.

Step 6: Close the window.

The second graphics routine *graph_1D/rates* reads results of computations (number of d.o.f. and the corresponding error) from file *files/result* and plots the convergence history.

Postscript echo. When opening a window, you will be asked for the name of a postscript file. If you return zero, no postscript file will be produced. Otherwise, the graphics system will produce and exact copy (postcript echo) of what you see on the screen, either a grey scale or color version. This will be useful for writing papers and preparing documents like this book.

4.4 Element routine

Once we have understood the preprocessing (mesh generation) and postprocessing (graphics routines), we are ready to discuss the core routine of any FE code - the element routine. The discussed

version - *laplace/elem* corresponds precisely to our model 1D elliptic problem, a single elliptic equation. We will discuss later how to generalize the routine to the more general case of an elliptic system of 1D equations.

On top of the data structure and control module, we encounter here a new module - *module/tree*. We will use this module to communicate with routine *constr_util/history* that determines history of refinements for an element, a crucial element of our data structure to support h-refinements. We will discuss this routine in a great detail in the next chapter. If you call *constrs/history* for an initial element *mdle*, the results are as follows:

- initial mesh element *NEL* = *mdle*, the generation level *NGEN* = 1;
- arrays containing history of refinements are irrelevant, they are empty;
- *ELEM_NODES* contains the element nodal connectivities, i.e. the vertex and middle nodes numbers.

So, the only useful information that you will find in *module/tree* at this point, is the element nodal connectivities. We could get them directly from *NODES(mdle)%nodes(1:3)*. The whole thing looks way over-sophisticated as the presented logic again anticipates supporting the mesh refinements.

The element routine includes also a new common - *commons/cint.blk* that contains one-dimensional Gauss quadrature data. The data is generated in *elem_util/const* called in the very beginning from the main program.

We now discuss the content of the element routine. The routine consists of three main steps.

Step 0: In this preliminary step, we obtain the order of the element and get its vertex nodes coordinates.

Step 1: Computing integral contributions. Please consult formula (3.3.3) and the algorithm discussed there. The first loop is always over the integration points. We select the order of quadrature according to the element order of approximation, N = *nint* = $p + 1$. If material data are constant within the element, the element matrices will be integrated exactly. Assume that we are at a specific integration point l. The master coordinate of the point ξ_l = *xi* is obtained from *commons/cint.blk*. Given the vertex nodes coordinates, we compute first the element shape functions $\hat{\chi}_k(\xi)$ = *vshap(k)*, and their derivatives wrt master element coordinate $\frac{d\hat{\chi}_k}{d\xi}$ = *dvshap(k)*. We use formula (3.1.2) to compute physical coordinate x_l = *x* of the point, and the jacobian $\frac{dx}{d\xi}$ = *dxdxi*, equal to the element length h. The jacobian is in fact constant, independent of the integration point, and it could be computed outside the loop. We compute the inverse jacobian $\frac{d\xi}{dx}$ = *dxidx*. Given the physical coordinate we evaluate the corresponding material and load data at that point, delivered by *data/getmat* and *data/getf*. We also compute the final *quadrature weight* equal to the product of the jacobian and Gaussian weight for

the master element. The next two loops over the element shape functions are crucial. The first corresponds to test functions, the second one to the element trial functions. Within the first loop we accumulate for the integral contributions to the element load vector, within the second loop we accumulate for the integral contributions to the element stiffness matrix.

If the adaptive integration flag is up, we call routine *laplace/elem_load* that uses an adaptive integration to make sure that the element load vector is computed exactly. We shall discuss the routine elsewhere.

Step 2: Accounting for BC. If the element is adjacent to the Neumann and Cauchy boundary, the extra boundary terms have to be added. Please consult the algorithm discussed in section 3.3. If the element is adjacent to the Dirichlet boundary, the element matrices have to be modified following the discussion in section 3.4. You should be able to follow these few lines of code yourself. The boundary conditions flags are recovered from *NVERS(nvert1)%bcond* and *NVERS(nvert2)%bcond*. This is the only place in this routine where we use the nodal connectivities, i.e. vertex nodes numbers *nvert1*, *nvert2*.

4.5 Assignment 2: Writing your own processor

The global FE system of equations has to be solved using a linear equations solver, either *direct* or *iterative*. In the 1D code, we will use direct solvers only.

The code comes with a frontal solver [] and a processor that provides an interface with the linear equations solver. This includes setting up an information for a *prefront* routine, feeding it with element matrices, and receiving the solution to store it in our data structure arrays. We shall discuss it in detail later.

In order to be able to call yourself a FE coder, you have to write your own processor routine. We will interface with the simplest linear solver ever, based on Gaussian elimination with no pivoting, and the global stiffness matrix stored in a full, dense mode (*blas/gausse*). The processor will have a little value for meshes with many elements. The stiffness matrix has many zeros then; storing the matrix and solving the system without accounting for those zeros, results in a very suboptimal performance. Nevertheless, we have to begin somewhere, and the discussed exercise is crucial for understanding finite elements.

Natural order of nodes and nodal d.o.f. We will come back now to the discussion that we started in Section 3.2. The discussed linear solver uses standard denumeration for the unknowns - they are simple denumerated from 1 to N - the problem size. In order to interface with it, we have to introduce such an ordering for our nodal d.o.f. Do not jump to a quick conclusion, going from left to right is not necessarily the simplest solution. First of all, the middle nodes have a variable number

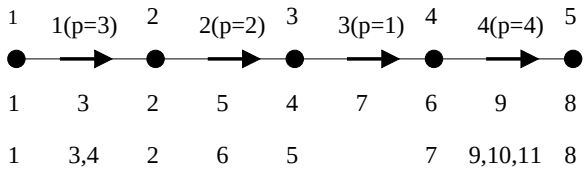


Figure 4.2: Natural order of nodes and nodal d.o.f. The top line lists nodes with their numbers in data structure arrays *NVERS* and *NODES*. Middle nodes are denoted with arrows, and listed with corresponding orders p. The second line shows the natural order of nodes induced by the order of elements. The third line shows the natural order of d.o.f. induced by the natural order of nodes.

of d.o.f. associated with them and you cannot just denumerate the nodes, as in classical Lagrange FE methods where you have just one d.o.f. per node. Secondly, a middle node may be *empty*. If order $p = 1$, there are no d.o.f. corresponding to the middle node. Nevertheless, the node still exists in the data structure. Thirdly, please anticipate 2D and 3D situations. Going 'from left to right' is meaningless there. OK, so we have to be a little bit more sophisticated.

We have already introduced routine *datstrs/nelcon* that takes us through all active elements in the mesh. In the 1D case, this is indeed going from left to right. The routine supports what we will call a *natural order of elements*. This ordering for elements induces a corresponding *natural order of nodes*. The algorithm looks as follows:

```
initiate a counter i = 0
for each element in the mesh
  for each element node nod
    if the node has not been visited yet (nbij(nod)=0)
      assign the number to the node nbij(nod) = i+1
      update the counter:  i = i + 1
    endif
  endfor
endfor
```

In the end, array *nbij(nod)* prescribes for each *active* node a sequential number, starting with 1, up to the total number of active nodes in the mesh. Notice that the introduced ordering for nodes is not 'from left to right', see example in Fig. 4.2. Finally, we can introduce the natural order for nodal d.o.f. We need to modify only slightly our algorithm.

```
initiate a counter i = 0
for each element in the mesh
  for each element node nod
    if the node has not been visited yet (nbij(nod)=0)
```

```
      assign the number to the node nbij(nod) = i+1
      update the counter:  i = i + ndof(nod)
    endif
  endfor
endfor
```

Here *ndof(nod)* is the number of d.o.f. associated with the node. For vertexes, the number is just one, for middle nodes, it is equal to $p-1$, where p is the corresponding order of approximation. If $p=1$, the node is empty, it will still be assigned a number, but the value of the counter will be the same for the next non-empty node.

Notation *nbij* is supposed to remind you you that we are constructing here a *bijection*, i.e. one-to-one mapping between integers and the active d.o.f.

The array *nbij(nod)* supporting the natural order of d.o.f. can now be used to determine element d.o.f. connectivities necessary for the assembling procedure discussed in Section 3.2. Recall that, given and element K, and its nodal connectivities *ELEM_NODES(1:3)*, and the matrix *nbij(nod)*, we determine d.o.f. connectivities $i = i(K,k)$ = *lcon(k)* as follows.

```
initiate element d.o.f.  counter, k = 0
for each vertex node nod
  update the counter, k = k + 1
  set the vertex d.o.f.  connectivity, lcon(k) = nbij(nod)
endor
for the middle node nod
  for j-th d.o.f.  of the node
    update the element d.o.f.  counter, k = k + 1
    set the d.o.f.  connectivity lcon(k) = nbij(nod)+j-1
  endfor
endfor
```

The final structure of your processor will look as follows.

Step 1: Determine array *nbij(nod)* supporting the natural order of nodal d.o.f. Initiate global matrices with zeros.

Step 2: Loop through all elements in the mesh. For each element, determine the corresponding nodal d.o.f. connectivities. Call the element routine to determine the element matrices, and assemble them into the global matrices, using the element d.o.f. connectivities.

Step 3: Call linear solver *blas/gausse* to solve the global system of equations. The solution is to be solved in a global vector.

Step 4: Loop again through all elements in the mesh. For each element, determine the corresponding nodal d.o.f. connectivities, and use them to copy the solution d.o.f. from the global vector to the data structure arrays *NVERS* and *NODES*.

For your convenience, we have already started writing the routine for you. You will find it in *mysolver* directory.

We finish this section with the formal summary of the second Assignment.

Assignment 2. Complete the 1D code. More precisely:

1. Write your own graphics routine displaying the exact and approximate solutions and the mesh. *Hint:* Begin by copying the existing routine and eliminating the interactive mesh refinements. Then add extra features like scales on both x and y axes, description of the exact and approximate solutions, the scale of orders of approximation to the right, etc. You may have to change the overall proportions of the whole picture... Show some initiative in perfecting your routine.

2. Write the interface with the linear solver *blas_pack/gausse*, following the discussion above.

3. Test the solver against the exact solution from the first assignment. You may also use the existing solver for verifying your results. Start with a uniform mesh of linear elements, and use the existing graphics routine to modify the orders. Call the *result* routine to check the content of the data structure.

4. Write a short document describing the new solver and its verification.

Exercises

1. Convert the 1D code to enable solution of more practical problems. Accept the following design assumptions.

   - Initial mesh elements may be of variable length and order of approximation.
   - Material data is constant within each element of the initial mesh.
   - There may be a different interface (jump) condition between any two elements in the initial mesh resulting in an extra Dirac's term for each vertex node.

   Suggested modifications:

   - Modify *module/data_structure1D* to include extra arrays to store the material and load data. Define new objects.

- Use the global control flag *NEXACT* in *module/control* to enable both the existing way of inputing data (through a known exact solution) and the new one.
- Modify routines *data/getmat, data/getf, data/getg, data/getc* accordingly.
- Write a new mesh generator enabling input of an initial mesh data following the design assumptions above.

Think of a series of tests that would verify all the introduced changes. Set up a sample problem and solve it. Write a short document to describe the modifications, verification procedure, and the example.

Chapter 5

Mesh Refinements in 1D

In this chapter we will study the h and p mesh refinements. We begin with the construction of the extension operator corresponding to h-refinement. The idea is fundamental in implementing the *constrained approximation* in 2D and 3D codes. Next we discuss the modification of order p for element middle nodes. This will lead us naturally to the idea of the *projection based interpolation*. We then return to the code and explain how the mesh modifications are realized in practice. We shall conclude the chapter with elementary interpolation error estimates and a discussion of possible types of convergence and adaptivity.

5.1 The h-extension operator. Constrained approximation coefficients

Fig. 5.1 displays a FE solution obtained using a mesh consisting of a single 'big' element of order $p = 8$. When you break the element into two element sons, you will see the same function, but this time supported on the mesh of two 'small' elements of the same order $p = 8$. Except for the

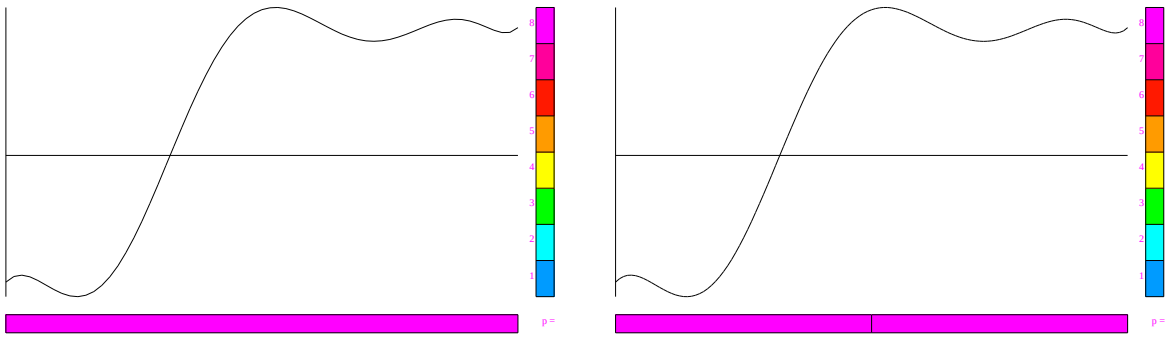


Figure 5.1: Representation of FE solution on a single 'big' (left), and two small (right) elements

graphical representation of the mesh, nothing has changed. The big element mesh representation of the FE solution is,

$$U_h(x) = U_1 e_1^{big}(x) + U_2 e_2^{big}(x) + \sum_{j=1}^{p-1} U_{3,j} e_{3,j}^{big}(x), \quad x \in [0,1] \, .$$

Here $e_i^{big}, i = 1, 2$ denote the left and right vertex nodes basis functions, $e_{3,j}^{big}, j = 1, \ldots, p-1$ are basis functions of the middle node, and $U_i, U_{3,j}$ denote the corresponding d.o.f.

The representation of the solution within the first small element is,

$$u_h(x) = u_1 e_1^{small}(x) + u_2 e_2^{small}(x) + \sum_{j=1}^{p-1} u_{3,j} e_{3,j}^{small}(x), \quad x \in [0, \frac{1}{2}] \, .$$

But, within the first element the basis functions reduce to the first element shape functions. Switching to the shape functions notation, we have,

$$u_h(x) = u_1 \chi_1^{small}(x) + u_2 \chi_2^{small}(x) + \sum_{j=1}^{p-1} u_{3,j} \chi_{3,j}^{small}(x), \quad x \in [0, \frac{1}{2}] \, .$$

In terms of master element shape functions and coordinate, we have,

$$\hat{u}_h(\xi) = u_1 \hat{\chi}_1^{small}(\xi) + u_2 \hat{\chi}_2^{small}(\xi) + \sum_{j=1}^{p-1} u_{3,j} \hat{\chi}_{3,j}^{small}(\xi), \quad \xi \in [0,1] \, ,$$

where $x = \frac{1}{2}\xi$, or $\xi = 2x$.

Operator prescribing for the big element d.o.f. $U_1, U_2, U_{3,j}$, the small element d.o.f. $u_1, u_2, u_{3,j}$ representing the same function, is called *the h-refinement extension operator*. We determine it by collocating both functions at $p+1$ uniformly distributed points within the first small element,

$$\hat{u_h}(\xi_l) = u_h(x_l) = U_h(x_l), \quad l = 0, \ldots, p \, ,$$

with $\xi_l = 1/p, x_l = \frac{1}{2}\xi_l = 1/2p$. More precisely, we have to do it for each choice of big element d.o.f., i.e., we activate one d.o.f. (big element shape function) at a time, and solve the corresponding collocation system of $p+1$ linear equations for the small elements d.o.f. A few simple observations follow.

- $u_1 = U_1$ is the value of the solution at the left vertex node. Consequently, u_1 does not depend upon $U_2, U_{3,j}$.
- u_2 depends upon all the 'big' nodes d.o.f.,

$$u_2 = R_{1,1,1,1} U_1 + R_{2,1,1,1} U_2 + \sum_{j}^{p-1} R_{3,j,1,1} U_{3,j} \, .$$

The four indices in coefficients $R_{i,j,k,l}$ are defined as follows:

i = 'big'node number, in the order: left 'big' vertex node, right 'big' vertex node, 'big' middle node,

j = d.o.f. number for the big node; for $i = 1, 2$ only $j = 1$ has a meaning, for $i = 3$, $j = 1, \ldots, p-1$.

k = 'small' node number, in the order: middle 'small' vertex node, left 'small' middle node, right 'small' middle node.

l = d.o.f. number for the small node.

Notice that $R_{1,1,1,1} = R_{2,1,1,1} = 0.5$ and that $R_{3,j,1,1}$ are zero for even j, as the corresponding shape functions are odd and, therefore, are equal zero at the mid-point.

- $u_{3,k}$ depends only upon $U_{3,j}, j = k, \ldots, p-1$. It is independent of the vertex d.o.f. and d.o.f. corresponding to 'big' element middle node shape functions of lower order,

$$u_{3,k} = \sum_{j=k}^{p-1} R_{3,j,2,k} U_{3,j} \ .$$

In a similar way we define coefficients $R_{3,j,3,k}$ corresponding to the middle node of the second 'small' element. The collocation problems for the two small elements are solved in *constr_util/setcnstr*, and the $R_{3,j,3,k}$ coefficients defining the h-extension operator are stored in array *RRR(1:3,1:NPMAX-1,1:3,1:NPMAX-1)*, in *commons/crrr.blk*.

Coefficients $R_{3,j,3,k}$ will be crucial in implementing the *constrained approximation* in 2D, a technique of enforcing continuity of approximation in presence of *constrained (hanging) nodes*.

5.2 Projection based interpolation in 1D

p-refinements. The coefficients discussed in the previous section, supporting the h-extension, are used for initiating d.o.f. for new nodes that are created during the h-refinement : two new middle nodes and one vertex node. The situation is easier in the case of a p-refinement, i.e. when we decide to increase the order of a middle node, from p^{old} to $p^{new} > p^{old}$. We have to pass then from representation:

$$u_h^{old} = u_1 \chi_1(x) + u_2 \chi_2(x) + \sum_{j=1}^{p^{old}-1} U_{3,j} \chi_{3,j}(x) \ ,$$

to

$$u_h^{new} = u_1 \chi_1(x) + u_2 \chi_2(x) + \sum_{j=1}^{p^{new}-1} U_{3,j} \chi_{3,j}(x) \ .$$

Due to the assumption about the *hierarchical shape functions*, the solution is very simple. We simply keep the old d.o.f. and initiate the new ones, $U_{3,j}, j = p^{old}, \ldots, p^{new} - 1$, with zeros. The corresponding functions will be identical.

p-unrefinements. The situation is more difficult when we decide to decrease the order of approximation for a middle node, i.e. $p^{new} < p^{old}$. Obviously, we cannot then, in general, support the same function; we are bound to loose some information. Nevertheless, we would like to loose as little as possible. This leads to the idea of using a projection. Given u_h^{old}, we determine u_h^{new} by minimizing the distance between the two functions in a norm,

$$\|u_h^{old} - u_h^{new}\| \to \min \ .$$

In the case of a Eucklidean norm, i.e. norm given by a scalar product ([98], page 514), this leads to a *linear system of equations*, otherwise the resulting problem to solve is nonlinear. The Sobolev norm, discussed in the previous chapters, suggests itself. For practical reasons, we also want to do the projection *locally*, over just one element. Finally, the result of our local projections has to be *globally continuous*, we are free to determine d.o.f. corresponding to the middle node, but we cannot change the vertex nodes d.o.f. without affecting the representation in the neighboring elements.

The three conditions: *locality, optimality* and *global continuity* lead naturally to the idea of the *projection based interpolation*. Given a function $U \in H^1(a,b)$, we define its *projection based interpolant* $u^p \in \mathcal{P}^p$, by requesting the following conditions.

$$\begin{cases} u^p \in \mathcal{P}^p \\ u^p(a) = U(a), \quad u^p(b) = U(b) \\ \displaystyle\int_a^b \left((u^p)' - U'\right)^2 \, dx \to \min \, , \end{cases} \tag{5.2.1}$$

or, equivalently (recall Section 1.3),

$$\begin{cases} u^p \in \mathcal{P}^p \\ u^p(a) = U(a), \quad u^p(b) = U(b) \\ \displaystyle\int_a^b \left((u^p)' - U'\right) \phi' \, dx = 0, \quad \forall \phi \in \mathcal{P}^p_{-1}(a,b) \, . \end{cases} \tag{5.2.2}$$

Here $\mathcal{P}^p_{-1}(a,b)$ denotes the polynomials of order p defined on interval (a,b), and vanishing at the endpoints. Recalling the representation of the interpolant in terms of element shape functions,

$$u^p(x) = u_1\chi_1(x) + u_2\chi_2(x) + \sum_j^{p-1} u_{3,j}\chi_{3,j}(x),$$

this is equivalent to a two step operation:

Step 1: Interpolation at vertices

$$u_1 = U(a), \quad u_2 = U(b) \, ,$$

Step2: Projection over the element

$$\sum_{j}^{p-1} a_{ij} u_{3,j} = b_i, \quad i = 1, \ldots, p-1 \, ,$$

where

$$\begin{cases} a_{ij} &= \displaystyle\int_a^b \chi_j' \chi_i' \, dx \\ b_i &= \displaystyle\int_a^b (U' - u_1 \chi_1' - u_2 \chi_2') \chi_i' \, dx \, . \end{cases}$$

Notice that the procedure is recursive, result of the interpolation at the vertices, enters the definition of the load vector for the projection problem. If we represent the interpolant u^p as the sum of the vertex linear contribution u_1^p and the middle node contribution u_2^p, we can rewrite the definition of the projection-based interpolant in yet another form,

$$\begin{cases} u^p = u_1^p + u_2^p \in \mathcal{P}^p \\ u_1^p = U \quad \text{at vertices} \\ (u_2^p, v)_1 = (U - u_1^p, v)_1 \quad \forall v \in \mathcal{P}_{-1}^p \, . \end{cases}$$

Here $(u, v)_1$ denotes the H^1 semi-product.

Implementing projection based interpolation. We have two options. The first one is the two step procedure described above. Upon computing the linear interpolant, we determine the load vector for the projection problem, and then solve the corresponding system of $p-1$ equations. But we may also formulate the interpolation procedure as the solution of a single system of $p+1$ equations, without the necessity of breaking it into the two logical steps. The system looks as follows.

$$\begin{pmatrix} 1 & 0 & 0 & \ldots 0 \\ 0 & 1 & 0 & \ldots 0 \\ a_{31} & a_{32} & a_{33} & \ldots a_{3,p+1} \\ \vdots & \vdots & \vdots & \vdots \\ a_{p+1,1} & a_{p+1,2} & a_{p+1,3} & \ldots a_{p+1,p+1} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{p+1} \end{pmatrix} = \begin{pmatrix} U(a) \\ U(b) \\ b_3 \\ \vdots \\ b_{p+1} \end{pmatrix}$$

Here, the element shape functions have been denumerated with a single index $\chi_j, j = 1, \ldots, p+1$, and

$$a_{ij} = \int_0^1 \chi_i' \chi_j' \, dx, \quad b_i = \int_0^1 U_i' \chi_i' \, .$$

We pay a price, though. The system is not symmetric, contrary to the previous projection problem.

The two possible ways of implementing the projection based interpolation are fully analogous to the two possible ways of implementing Dirichlet BC's, comp. Exercise 3.1.

Initiating d.o.f. for p- and h-unrefinements. We can now describe precisely the initiation of d.o.f. for a p-unrefined element. The d.o.f. for the modified middle node are set to those corresponding to the projection-based interpolant of the previous solution. Exactly the same procedure can be applied when initiating d.o.f. for the middle node of an h-unrefined element. The only difference will be in the representation of the interpolated function U which , in this case, is supported over two elements.

5.3 Supporting mesh refi nements

We discuss now shortly the mesh modification routines. Routine *meshmods/break*, breaking an element, collects only information about the element coordinates and solution d.o.f. and calls *meshmods/break_mdle* - routine breaking the element middle node. Indeed, the entire process of breaking an element is reduced to just this - breaking its middle node.

Breaking a middle node means generating its three sons. The first two are the middle nodes of element sons, the third is the new vertex node shared by the the two new elements. The h-extension operator coefficients are used to initiate the new nodes d.o.f. Generation of the new nodes involves initiation of two new entries in data structure array *NODES* and one new entry in *NVERS*, allocating d.o.f. for the middle nodes, and resetting pointers to the first free nodes. You should be able to follow the explanations in the routine.

Routine *meshmods/nodmod* is much simpler. We only need to modify the number of d.o.f. associated with the node, and initiate the new d.o.f. Implementation of the projection-based interpolation is left as an exercise.

Finally, *meshmods/cluster_mdle* performs reverse operations to those in *meshmods/break_mdle*. The three nodal sons are deleted. Initiation of middle node d.o.f. should be done using the interpolation, and has been left for an exercise (Exercise 5.3).

5.4 Data structure supporting routines

In process of creating new or deleting old nodes - the middle node sons, we update the tree information for nodes. When breaking a middle node *mdle*, we store its three sons numbers - the two new middle nodes: *mdle1*,*mdle2*, and the new vertex node: *vert* in *NODES(mdle)%sons*. We also store *mdle* in *NODES(mdle1)%father*, *NODES(mdle2)%father*, and *NVERS(nvert)%father*. This two way information is crucial for supporting fundamental logical operations for a FE mesh, despite the very minimal changes made in the data structure during the mesh refinement. As the middle nodes are identified with elements, the *family information* for the middle nodes allows for supporting the usual tree structure for the elements, see Fig. 5.2. By storing the extra information about/for the vertex sons, we avoid completely storing explicitly *any information* on elements resulting from the

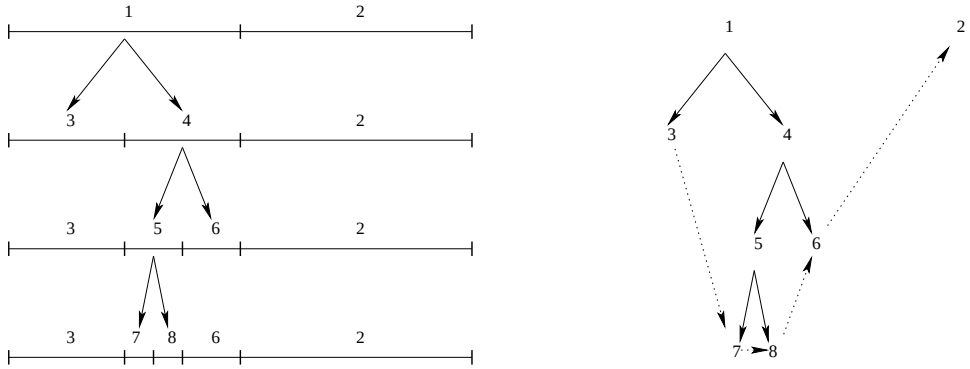


Figure 5.2: Tree structure for middle nodes (elements), and the corresponding natural order of elements for a sequence of h-refined meshes

h-refinements. This is a big departure from earlier versions of our codes, compare [54, 48, 46]. The difference between the old and the new data structures is insignificant in 1D, constitutes an essential improvement in 2D, and it is absolutely essential for the 3D codes.

We shall discuss now two crucial, data structure supporting algorithms.

5.4.1 Natural order of elements

The idea is explained in Fig. 5.2. The ordering for active elements follows the order of elements in the initial mesh and the order of sons within a family. The formal algorithm look as follows.

```
input:  an element middle node number mdle
Step 1:  move horizontally in the tree
if mdle = 0, replace it with the middle node of the first element
  in the initial mesh
if (mdle has a father nfath) then
  find the son number in the family (1 or 2)
  if mdle is the first son then
    replace mdle with its brother
    go to Step 2
  else
    replace mdle with its father
    go to Step 1
  endif
else
  identify the initial mesh element nel corresponing to mdle
  replace mdle with the middle node of the element nel+1 from
```

```
    the initial mesh
  go to Step 2
endif
Step 2:  move vertically
if mdle has sons then
  replace mdle with its first son
  go to Step 2
else
  return
endif
```

Please study the algorithm and routine *datstrs/nelcon* carefully, as it will extend with practically no changes (just more sons in a family) to 2D and 3D. In the 1D case, the algorithm produces 'left to right' ordering.

5.4.2 Reconstructing element nodal connectivities

Please open routine *constr_util/history*. For a given element middle node number *mdle*, the routine returns the entire ancestors history of the element and, in the end, the element nodal connectivities. This is a crucial algorithm that has made possible eliminating elements (except for those in the initial mesh) from the data structure arrays.

The routine communicates with the user through *module/tree*. This is because the module contains a complete information about the h-refinements that have led to the current (middle node of the) element. Having access to the full information allows for a flexible use of it, dependently upon a current need.

The idea of the algorithm is simple. In the first step, we use the tree information to ascend the tree of middle nodes, all the way to the middle node of an initial mesh element. On the way there, we collect the information about the ancestors - their numbers and location (son's number) within their families. In the end we identify number of the initial mesh element. The information is stored in *module/tree*. In the second step, we begin by copying the nodal connectivities for the initial mesh ancestor, from array *ELEMS* in *module/data_structure1D*. We then descend the path along the tree of middle nodes determined in the first step, and use the family information to reconstruct nodal connectivities for all the ancestors. The procedure is recursive, in the end we store only the connectivities for the original element corresponding to *mdle* - the input. The connectivities are returned in *ELEM_NODES*, as discussed earlier.

Before studying the formal algorithm presented below, make sure that you understand the meaning of array *PARENT_NODES* in *constr_util/history*. Given an element *mdle*, and its n-th son; for an

i-th node of the son *PARENT_NODES(1,i,n)* is the element node number for the father of the node, and *PARENT_NODES(2,i,n)* is the son's number of the node.

```
input:  an element middle node number mdle
Step 1:  ascend the tree of middle nodes
if mdle has a father then
  store the son's number for mdle
  replace mdle with its father
  go to Step 1
else
  identify the corresponding initial mesh element and
    copy the element nodal connectivities from ELEMS
endif
Step 2:  descend the tree
if mdle has been refined (has sons) then
  identify proper son and the corresponding son's number n
  for i-th node of the n-th son
    set ip = PARENT_NODES(1,i,n), is = PARENT_NODES(2,i,n)
    the j-th node of the son element is is-th son of ip-th node
      of node mdle
  endfor
  replace mdle with its son, and nodal connectivities of mdle
    with those of the son
else
  return
endif
```

5.5 Programming bells and whistles

In this section, we review shortly a few remaining important components of the code.

5.5.1 Interfacing with the frontal solver

The so-called frontal solver has become a popular choice among direct solvers for finite element codes due to its natural implementation in an *element by element* scheme. In this method, a *front* sweeps through the mesh, one element at a time, assembling the element stiffness matrices and load vectors into the global matrices. The distinction from the standard assembling procedure is that, as soon as all of the contributions for a given d.o.f. have been accumulated, that d.o.f. is elimi-

nated from the system of equations using standard Gaussian operations. Thus, in the frontal solver approach the operations of assembling and elimination occur *simultaneously*. The global stiffness matrix never needs to be fully assembled, and this leads to the significant savings in memory that has given the frontal solver its popularity.

Here we will only describe the interface with the frontal solver, not the solver itself. The interface is constructed via four routines, all located in the *solver1* directory: *solver1/solve1*, *solver1/solin1*, *solver1/solin2*, and *solver1/solout*. *solver1/solve1* is the actual driver for the frontal solver. We will now give an short overview of these routines.

The frontal solution consists of two steps: a *prefront*, and the actual solver performing the Gaussian elimination. Begin, please, by opening *module/frsolmod* which contains all work arrays for the frontal solver. This includes two work arrays for the prefront, two arrays to store the element *destination vectors* returned by the prefront, work arrays to store element matrices, and the actual work space for the solver. Parameter *MFRSOL* prescribes the length of the work array and it should reflect the size of your computer memory. If you specify it too big, in excess of your actual memory, the system will start *paging*, i.e. it will substitute the *virtual memory* (physically residing on the hard disk) for the needed electronic memory. It is more efficient if the solver controls the use of disk files itself. However, if you specify parameter *MFRSOL* too small, the solver will start using the disk memory before this is actually needed. All work arrays in *module/frsolmod* are allocated dynamically, with the memory released to the system after solving the problem.

The prefront requires two arrays on input: *IN* and *IAWORK*. For each element *mdle*, *IN(mdle)* contains the number of nodes for the element, in 1D this is just three, unless the middle node is empty ($p = 1$); we have two nodes then only. *IAWORK* contains a listing of nicknames for the nodes of the element. The nicknames store two integers:

- a unique integer identifier of the node,
- number of d.o.f. associated with the node.

Since vertex and middle nodes are denumerated independently, we have to define the identifier in a different way for the two kinds of nodes. We use simply the node number for vertex nodes, and the node number augmented with *MAXVERS*- maximum number of vertex nodes, for middle nodes. It is not sufficient to use *NRVERS* - the current number of vertex nodes, for the sake of avoiding the duplication of identifiers. In process of mesh refinements/unrefinements, the occupied space in data structure array *NVERS* may loose its contiguity, and some vertex nodes numbers my be greater then *NVERS*.

The node nicknames are therefore defined as follows:

- for a vertex node *nod*, *nickname* = *nod**1000 + 1 ,
- for a middle node of order $p > 1$, *nickname* = (*nod* + *MAXVERS*)*1000 + (p-1) .

The nicknames are stored in the single index array *IAWORK*. With this information, the prefront produces the destination vectors which, for a given element, denote at what stage of the frontal solution each of its nodes can be eliminated. Once this information is constructed, the elimination phase can begin. *solver1/solve1* prepares arrays *IN* and *IAWORK*, calls the prefront routines, and then calls the main elimination routines. Thus, this routine is seen to be the primary driver of the frontal solver.

The other interface routines are simply for auxiliary purposes. The destination vectors are returned by the prefront through *solver1/preout*. We simply store them in a couple of arrays in *module/frsolmod*. For a given element, *solver1/solin1* recovers a listing of the destination vectors from the storage, *solver1/solin2* returns the element stiffness matrix and load vector, and *solver1/solout* takes the solution values returned from the frontal solver and inserts them into the data structure arrays .

5.5.2 Adaptive integration

We arrive at the need for a precise integration twice in the code. We need it for the first time when integrating the element load vector. For practical, engineering problems, this is not an issue. The load is always piecewise constant and Gaussian integration is sufficient. When studying the convergence, however, we use the method of *manufactured solutions* and compute the necessary load data by calling *data/exact* - routine evaluating the exact solution and its derivatives. If the solution is irregular or singular, Gaussian integration is insufficient, and the results may be completely erroneous; a more precise integration is needed.

We also need the adaptive integration when computing the actual FE error, i.e. the H^1-norm of the difference between the exact and FE solutions.

Routines *elem_util/comp_error* and *laplace/elem_load* are practically identical and both use the same methodology of an h-adaptive integration. The order of Gaussian quadrature is fixed according to element order p, $n = p+1$, as discussed earlier. Instead of integrating over the whole element, we split it into subintervals, integrate over each subinterval separately and then add the corresponding contributions. The subintervals are determined adaptively, as needed.

Given an element to integrate, we begin by creating a list of subintervals to integrate with, and place the element on the list.

We then begin by taking the first subinterval from the list (which, in the beginning, is the whole element), and compute the integral twice, first integrating over the whole subinterval and then by splitting the element into two equal subintervals, integrating over the two subintervals separately, and adding up the contributions. Obviously, the second procedure is more accurate. We then compare the two results of integration. If the difference exceeds a tolerance level, we add both subintervals to the list, if the tolerance has been met, we accumulate (with the better value) for the ultimate

integral(s), and remove the subinterval from the list.

We then proceed with the next subinterval on the list, until the list is empty.

We shall not present a formal algorithm here. Please study the two routines *elem_util/comp_error* and *laplace/elem_load*. The integrals over an arbitrary subinterval are computed in *laplace/load_contr* and *laplace/error_contr*, using the standard numerical integration algorithm discussed earlier.

5.5.3 Choice of shape functions

From the mathematical point of view, the approximate solution depends not upon the choice of specific shape functions but the polynomial spaces, i.e. the order of approximation p. We have written our codes in such a way as to be able to compute with different shape functions and, among other things, be able to study the effect of their choice on conditioning and convergence. At present, the code supports two sets of hierarchical shape functions. By setting flag *SHAPE_FLAG* in *module/control*, in *files/control*, you can choose between:

Peano shape functions: *SHAPE_FLAG*=1

Integrated Legendre polynomials: *SHAPE_FLAG*=2

At any point, you may want to add an extra choice to the code by expanding *elem_util/shape1*.

5.6 Interpolation error estimates

We return now to basic mathematical issues. We have learned from Cea's lemma that, for the coercive case, the actual *approximation error* can always be bounded by a mesh independent constant times the *best approximation error*. Thus, it is sufficient to estimate and control the *best approximation error*. By definition, the best approximation error is always bounded by the norm of the difference between the exact solution and any particular choice of a function that 'lives' in the FE space. The choice, we make, is the *projection-based interpolant* of the exact solution $w_{hp} = \Pi_{hp} u$,

$$\inf_{w_{hp} \in \tilde{u_0} + V_{hp}} \|u - u_{hp}\|_{H^1(0,l)} \leq \|u - \Pi_{hp} u\|_{H^1(0,l)} \, .$$

Just in this section, we will use the discretization symbol hp, to remind you that both element size h and order p will affect the interpolant and the FE solution. The quantity on the right is the *global (projection based) interpolation error*. Obviously, the global interpolation error can be bounded by summing up element interpolation errors,

$$\|u - \Pi_{hp} u\|^2_{H^1(0,l)} = \sum_K \|u - \Pi_{hp} u\|^2_{1,K} \, .$$

p-interpolation error estimate. We begin by estimating the error over the master element. It follows from the construction of the projection-based interpolation that the interpolation operator is a continuous map from space $H^1(0,1)$ onto the polynomial space (comp. Exercise 5.4),

$$\|\Pi_p u\|_{H^1(0,1)} \leq C\|u\|_{H^1(0,1)} ,$$

with a constant C *independent* of order p. Notice that the operator preserves polynomials of order p, i.e.

$$\Pi_p u = u \quad \forall u \in \mathcal{P}^p .$$

This leads immediately to estimating the interpolation error by the *best approximation error* (*projection error*),

$$\begin{aligned} \|u - \Pi_p u\| &= \|u - w_p - \Pi_p(u - w_p)\| \\ &\leq (1 + \|\Pi^p\|)\|u - w_p\| \\ &\leq (1 + C) \inf_{w_p \in \mathcal{P}^p} \|u - w_p\| \end{aligned}$$

It is known that (see [113], pp. 75),

$$\inf_{w_p \in \mathcal{P}^p} \|u - w_p\|_{H^1(0,1)} \leq C(r) p^{-r} \|u\|_{H^{1+r}(0,1)} ,$$

with constant $C = C(r)$ dependent upon r but independent of function $u \in H^{1+r}(0,1)$. $H^n(0,1)$ denotes here *Sobolev space of order* n, consisting of L^2-functions whose all derivatives $u^{(i)}$ (in the distributional sense) up to order n are square integrable as well,

$$H^n(0,1) = \{u \in L^2(0,1) \ : u^{(i)} \in L^2(0,1), \quad i = 1, \ldots, n\} .$$

Sobolev space is a Hilbert space with a natural inner product defined as,

$$(u,v)_{H^n(0,1)} = \sum_{i=0}^{n} \int_0^1 \frac{d^i u}{dx^i} \frac{d^i v}{dx^i} \, dx ,$$

and the corresponding Sobolev norm $\|u\|_{H^n(0,1)}$ and seminorm $|u|_{H^n(0,1)}$ defined as follows,

$$\begin{aligned} \|u\|_{H^n(0,1)} &= \left(\sum_{i=0}^{n} \int_0^1 \left(\frac{d^i u}{dx^i} \right)^2 dx \right)^{\frac{1}{2}} \\ |u|_{H^n(0,1)} &= \left(\int_0^1 \left(\frac{d^i u}{dx^i} \right)^2 dx \right)^{\frac{1}{2}} . \end{aligned}$$

Sobolev space of order n is a natural generalization of the Sobolev space of order 1, discussed in the first chapter, and a tool to measure the regularity of solutions.

Putting together the results above, we arrive at the interpolation error estimate,

$$\|u - \Pi_p u\|_{H^1(0,1)} \leq C p^{-r} \|u\|_{H^{1+r}(0,1)} ,$$

with constant $C = C(r)$ depending upon index r, but independent of order p and function $u \in H^{1+r}(0,1)$.

The Sobolev norm of order n on the right hand side is equal to the sum of the seminorms of order i, for $i = 0, 1, \ldots, n$. We can eliminate the first p seminorms from the estimate using the Bramble-Hilbert lemma.

THEOREM 6

(Bramble-Hilbert Lemma)
Let $A : H^{r+1}(0,1) \to H^1(0,1)$ be a linear, continuous operator,

$$\|Au\|_{H^1(0,1)} \leq C\|u\|_{H^{r+1}(0,1)} ,$$

that vanishes for polynomials of order p,

$$Au = 0, \quad \forall u \in \mathcal{P}^p .$$

Then, there exists a $C > 0$, such that,

$$\|Au\|_X \leq C \left(\sum_{i=p+1}^{r} |u|^2_{H^i(0,1)} \right)^{\frac{1}{2}} ,$$

for every $u \in H^{1+r}(0,1)$. ∎

Proof: See, e.g. [32], see also Exercise 5.5. ∎

As the projection-based interpolation operator Π_p reproduces polynomials of order p, the difference $I - \Pi_p$ vanishes on polynomials of order less or equal p, and we can modify our interpolation error estimate to its final form,

$$\|u - \Pi_p u\|_{H^1(0,1)} \leq Cp^{-r} \left(\sum_{i=p+1}^{r} |u|_{H^i(0,1)} \right)^{\frac{1}{2}} , \tag{5.6.3}$$

hp-interpolation error estimate. We use now the standard scaling argument to convert our p-interpolation estimate into an hp-estimate. Let $\hat{u}(\xi)$ be a function defined on the master element, and $u(x)$ the corresponding function defined on the physical element, with the usual affine map,

$$x = x_K + \underbrace{(x_{K+1} - x_K)}_{h_K} \xi, \quad \frac{dx}{d\xi} = h_K .$$

Change of variables leads to the following scaling relations,

$$\begin{aligned} \|u\|^2_{L^2(x_K,x_{K+1})} &= \int_{x_K}^{x_{K+1}} u^2\,dx = h\int_0^1 \hat{u}^2\,d\xi = h|\hat{u}|^2_{L^2(0,1)} \\ |u|^2_{H^1(x_K,x_{K+1})} &= \int_{x_K}^{x_{K+1}} \left(\frac{du}{dx}\right)^2 dx = h^{-1}\int_0^1 \left(\frac{d\hat{u}}{d\xi}\right)^2 d\xi = h^{-1}|\hat{u}|^2_{H^1(0,1)}\,, \end{aligned}$$

where $h = h_K$. More generally,

$$|u|^2_{H^r(x_K,x_{K+1})} = h^{-2r+1}|\hat{u}|^2_{H^r(0,1)}\,.$$

We now denote by Π_{hp} the projection-based interpolation operator defined on a physical element (x_K, x_{K+1}), and by Π_p the projection-based interpolation operator on the master element $(0,1)$. The change of variables theorem implies the following commutativity property (Exercise 5.6),

$$(\Pi_{hp}u)^{\wedge} = \Pi_p\hat{u}\,. \tag{5.6.4}$$

In other words, the interpolation on the physical element is equivalent to first changing the variables and 'moving' function u from the physical to the master element, then interpolating on the master element, and then 'moving' the interpolant back from the master to the physical element. If we identify explicitly the operator corresponding to the change of variables,

$$T : u \to \hat{u}\,,$$

we can display clearly the commutativity property as:

$$T\Pi_{hp} = \Pi_p T, \quad \Pi_{hp} = T^{-1}\Pi_p T\,.$$

We use now the commutativity of the interpolation operators, p-estimate on the master element, and the scaling relations to arrive at the final estimate,

$$\begin{aligned} |(I-\Pi_{hp})u|_{1,K} &= h^{-\frac{1}{2}}|(I-\Pi_p)\hat{u}|_{1,\hat{K}} \\ &\le Ch^{-\frac{1}{2}}p^{-r}\left(\sum_{i=p+1}^{r}|\hat{u}|^2_{i,\hat{K}}\right)^{\frac{1}{2}} \\ &= Ch^{-\frac{1}{2}}p^{-r}\left(\sum_{i=p+1}^{r}h^{2i-1}|u|^2_{i,K}\right)^{\frac{1}{2}} \\ &\le C\frac{h^p}{p^r}\|u\|_{r+1,K}\,. \end{aligned}$$

Here we have switched to the Ciarlet notation [32] for Sobolev (semi)norms, with K and $\hat{K} = (0,1)$ denoting the physical and master elements, respectively.

So far, we have assumed that order p does not exceed the regularity index r, $p \leq r$. If order p exceeds r, then the exponent for the h factor will be limited by the regularity rather than order of approximation. It is customary then to write the estimate in the form,

$$|(I-\Pi_{hp})u|_{1,K} \leq C\frac{h^{\min\{p,r\}}}{p^r}\|u\|_{r+1,K} \ .$$

Finally, we observe that we have no trouble trading the first order seminorm on the left-hand side of the estimate, for the first order norm. Indeed, by the Poincare inequality argument (for functions vanishing at endpoints, the L^2 norm can bounded by the H^1- seminorm), we have,

$$\begin{aligned} |(I-\Pi_{hp})u|_{0,K} &= h^{\frac{1}{2}}|(I-\Pi_p)\hat{u}|_{0,\hat{K}} \\ &\leq h^{\frac{1}{2}}|(I-\Pi_p)\hat{u}|_{1,\hat{K}} \\ &\leq C\frac{h^{p+1}}{p^r}\|u\|_{r+1,K} \ . \end{aligned}$$

That yields the final estimate in the form,

$$\|(I-\Pi_{hp})u\|_{1,K} \leq C\frac{h^{\min\{p,r\}}}{p^r}\|u\|_{r+1,K} \ . \tag{5.6.5}$$

The estimate is optimal for the first order seminorm, but not for the L^2-part of the norm. We will return to this subject later.

5.7 Convergence

In this section we put together Cea's argument and the hp interpolation error estimate to discuss the meaning of convergence rates for various types of refinements.

Uniform h refinements. This is the classical version of finite elements. We start with an initial mesh of elements of uniform order p, and keep on decreasing mesh size h. This can be done in practice either by changing the number of elements in the initial mesh or by breaking all elements in the mesh, i.g. refining the mesh globally in h, and proceeding recursively with such refinements. If we use uniform meshes, or understand by h the maximum element size, we have,

$$\|u-u_h\|_1 \leq Ch^{\min\{p,r\}}\|u\|_{r+1} \ .$$

Exponent $\min\{p,r\}$ specifies the expected rate of convergence. If our estimate is optimal, we expect to see asymptotically an equality, and the estimate can be interpreted on a logarithmic scale,

$$\log\|u-u_h\|_1 \approx c+\min\{p,r\}\log h \ .$$

Thus, on the log-log scale, we should see a straight line, with the slope equal to the rate of convergence. Notice that the rate of convergence is limited by both order of approximation p and the regularity of the solution. Constant c hides the logarithm of all constants used in the presented estimates.

Taking into account the double contributions for vertex nodes, we can say that each element contributes with p d.o.f.. The total number of d.o.f. N is then inversely proportional to the element size,

$$N = p/h \, ,$$

and we can represent the final estimate in terms of the total number of d.o.f. rather than element size,

$$\|u - u_h\|_1 \leq C N^{-\min\{p,r\}} \, .$$

Here we have incorporated the norm of the exact solution into the constant C.

Uniform p refinements. Instead of decreasing the element size, we shall start with an initial mesh (it may consist of just one element only) and proceed by increasing *uniformly* order p in all elements. This leads to the estimate,

$$\|u - u_h\|_1 \leq C p^{-r} \|u\|_{r+1} \, ,$$

or in terms of the total number of d.o.f. N, proportional to p,

$$\|u - u_h\|_1 \leq C N^{-r} \, .$$

In the case of solutions with unlimited accuracy, there is no limit on the rate. Otherwise, the rate is limited by the regularity only.

Adaptive h refinements. Instead of refining uniformly the mesh, we can refine only those elements where the error is large. For that we need an *a posteriori error estimate*, or at least an *error indicator*. Those are usually of the form,

$$\|u - u_h\|_1^2 \leq \sum_K \eta_K^2 \, ,$$

where η_K is an error indicator corresponding to element K. We can refine then only those elements for which the error indicators are large. The simplest strategy would be to determine the maximum element indicator,

$$\eta_{max} = \max_K \eta_K^2 \, ,$$

and then refine all elements K for which:

$$\eta_K \geq \frac{1}{3} \eta_{max} \, .$$

Parameter $1/3$ is selected ad hoc, and it can be replaced with any parameter from interval $(0,1]$. Without going into details (see [19, 113]), we mention only that the h-adaptive refinements (if the error indicator is correct) allow to eliminate the influence of regularity of the solution on the convergence rate. The final error estimate reads as folows.

$$\|u-u_h\|_1 \leq CN^{-p}\,.$$

Notice that, since meshes are not longer (quasi)uniform, it makes no sense to estimate the rate in terms of the element size h. Thus for solutions with limited regularity (e.g. solutions with singularity), the h-adaptivity improves the rate, restoring the rate corresponding to regular solutions. For regular solutions, the h-adaptivity does not improve the rate of convergence, see e.g. [52].

Adaptive p refinements. Instead of breaking elements with large error indicators, we can increase adaptively the order of approximation p. We are not aware of corresponding estimates of convergence rates better than for the uniform p refinements.

Adaptive hp refinements. This is the most flexible kind of refinement. We change adaptively *both* element size h and order p.

In the case of an unlimited regularity, we expect the p-refinemnets to produce exponnetial (superalgebraic) convergence. In the case of a limited regularity, though, all discussed refinement schemes will deliver only *algebraic rates* of convergence.

It is a profound result of Prof. Babuška, his students and collaborators [113] who have demonstrated that, with properly done refinements, we can expect *exponential convergence*,

$$\|u-u_h\|_1 \leq Ce^{-\alpha N}\,, \tag{5.7.6}$$

for *both regular and irregular(singular) solutions*.

It has been the goal of our research in the past 15 years to demonstrate that such a convergence can be realized in a fully automatic mode, without any interference from the user. This will be the subject of the next chapter.

Adaptive p refinements with an optimal initial mesh. With an *a priori* knowledge of location of singularities, and their form, one can come up with an optimal initial mesh which, when combined with adaptive or even uniform p-refinements, will deliver exponential decay of the error. The first important class of problems for which we can construct such meshes, are elliptic problems where corners, edges or material interfaces may produce point and edge (3D) singularities. The construction leads to the so called *geometrically refined meshes*. For a class of *singularly perturbed problems* [91, 113], we have so called optimal Shishkin meshes. In both cases, if we start with such

an optimal mesh, the adaptive or uniform p-refinements will produce a quasi exponential decay of the error. We call it only 'quasi exponential' since, after a number of refinements, the convergence will return to its algebraic rates only. The method was pioneered by Prof. Szabo and his students, see e.g. [117], and it is frequently also called an hp-method.

5.8 Assignment 3: Studying convergence

Before we start studying adaptivity, we want you to get some experience with the various discussed types of convergence, study the influence of the regularity of the solution, and the importance of the assumptions that guarantee the coercivity of the bilinear form.

We shall deal with two different boundary-value problems.

A coercive elliptic problem

$$\begin{cases} -u'' = f(x) \quad x \in (0,1) \\ u(0) = u_0 \\ u'(1) + u(1) = \gamma \end{cases}$$

with *load data* corresponding to one of the two exact solutions:

- *An analytical, but possibly irregular solution* (with high gradients),

  $$u(x) = \arctan(\alpha(x - \frac{1}{\pi})) \, ,$$

  with $\alpha = 20, 60, 120$.

- *A singular solution* (with a singularity at $x = 0$),

  $$u(x) = x^{0.6} \, .$$

A convection dominated diffusion problem

$$\begin{cases} -(\epsilon u')' + u' = 0 \quad x \in (0,1) \\ u(0) = 1 \\ u(1) = 0 \end{cases}$$

And now the actual assignment.

1. For each of the four exact solutions and the 'stable' problem, use the initial mesh generator to vary h or p, and study (uniform) h and p convergence rates. You may want to use the interactive refinements to modify the initial mesh, before you study the p-refinements.

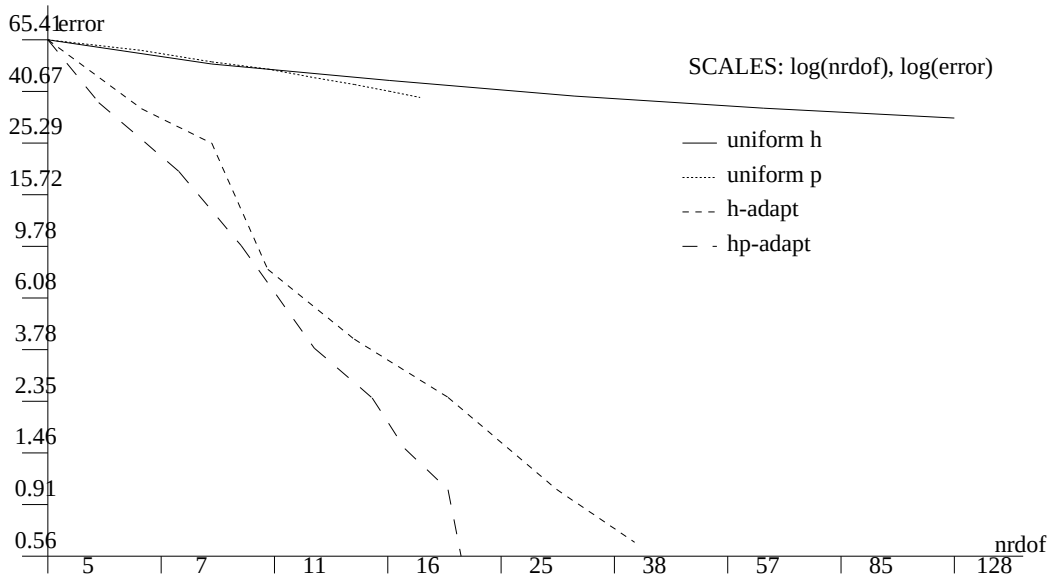


Figure 5.3: Example 1 with $\alpha = 20$. Convergence rates on the log-log scale

Use then the hp-adaptive, and h-adaptive routine provided in the code, to produce optimal hp and h meshes, and compare the convergence rates for uniform h,p, h-adaptive, and hp-adaptive refinements. Use second order elements when studying the h-adaptive convergence (the most common choice in commercial codes). Collect the results in *files/result* and run *graph_1D/rates* to display the corresponding rates. Produce and print the corresponding PS files. Fig. 5.3 shows a sample picture for the first solution and $\alpha = 60$. You should produce at least four different pictures (there are three different values of parameter α in the first solution).

Hint: You may want to utilize the mesh refinements routines and add to the code the possibility of uniform h- and p- refinements, rather than modifying the input file by hand.

2. Based on the collected numerical evidence, discuss the dependence of convergence rates on the regularity of the solution. For the first exact solution, discuss the *preasymptotic* and *asymptotic* behavior of the solution, and its dependence upon parameter α.

3. Determine and code the exact solution to the second problem with ϵ left as a (diffusion) parameter. Solve the problem using a mesh of your choice for value of $\epsilon = 1$. Next, for the same mesh, try to solve the problem for smaller values of $\epsilon = 0.1, 0.01, \ldots$. Try to come up

with your own strategy to cope with the problems for very small values of diffusion parameter ϵ. *Hint:* Use the interactive mesh refinements to generate different meshes.

5.9 *Definition of a finite element*

We conclude this chapter with a short discussion on a more precise definition of a finite element. Our construction of a *conforming finite element* incorporates three components:

Geometry: a domain $K \subset \mathbb{R}^n, n = 1, 2, 3$,

The element space of shape functions: $X(K)$,

Interpolation operator: $\Pi : \mathcal{X}(K) \to X(K)$.

The interpolation operator Π is defined on a large functional space $\mathcal{X}(K)$, containing $X(K)$ which should include solutions to our boundary-value problems. Notice that, contrary to the classical definition of Ciarlet [32], we prefer to think in terms of spaces and interpolation operators rather than degrees-of-freedom and shape functions. The difference between the two definitions is rather insignificant. If we select a basis in the dual $X(K)^*$,

$$\psi_j : \mathcal{X}(K) \to \mathbb{R}, \quad j = 1, \ldots, N = \dim X(K) \, ,$$

and introduce shape functions χ_i as the dual basis,

$$\psi_j(\chi_i) = \delta_{ij} \, ,$$

we have the usual definition of the interpolation operator,

$$(\Pi u)(x) = \sum_j^M \psi_j(u)\chi_j(x) \, .$$

For hp elements, though, it is more natural to think in terms of spaces and the (projection-based) interpolation operators directly. However, if we introduce a specific basis of shape functions,

$$\chi_j \in X(K), \quad j = 1, \ldots, N \, ,$$

we can always identify (see Exercise 5.7) a unique set of degrees- of-freedom functionals ψ_j such that the Ciarlet definition of the interpolation operator will coincide with our original construction. Explicit definition of those functionals are rather awkward, and we prefer to think directly in terms of the interpolation procedure rather than selecting particular d.o.f. functionals. Notice also that our definition does not operate with particular shape functions which is consistent with our way of coding the hp elements.

Exercises

Exercise 5.1 Think of other linear functionals than evaluation at collocation points, to evaluate the h-extension operator coefficients. Can you define the linear functionals in such a way that the resulting system of linear equations decouples (is lower or upper triangular) ?

Exercise 5.2 Discuss why coefficient RRR needs to be defined only for the maximum order $NPMAX$. Would this also be the case if the shape functions were not hierarchical, i.e. classical Lagrange shape functions?

Exercise 5.3 Implement the projection-based interpolation. Consider two cases for a function U to be interpolated over an element K of order p:

case 1: U is a polynomial of a higher order $p^{old} > p$,

case 2: U is a piecewise polynomial defined over two small element sons, with orders p_1, p_2.

Assume in both cases that function U is given in terms of element d.o.f. Use the implemented routine to initiate d.o.f. for p- and h-unrefinements.

Exercise 5.4 Prove that the projection-based interpolation operator is a well defined and continuous operator from space $H^1(a,b)$ into itself.

Exercise 5.5 Prove the Bramble-Hilbert lemma. *Hint:* Consider the quotient space $H^{r+1}/\mathcal{P}^p$, and use the fact that, in a finite dimensional space, any two norms are equivalent.

Exercise 5.6 Prove the commutativity property 5.6.4.

Exercise 5.7 Let $\Pi_p : H^1(0,1) \to \mathcal{P}^p(0,1)$ be the projection- based interpolation operator. Let χ_1, χ_2 denote the linear shape functions, and let $\chi_j, j = 3, \ldots, p+1$ denote any basis for the space $\mathcal{P}^p_{-1}$ of polynomials of order less or equal p, vanishing at the endpoints. Demonstrate that there exists a unique set of linear and continuous functionals $\psi_j : H^1(0,1) \to \mathcal{P}^p(0,1)$ such that,

$$\Pi_p u = \sum_{j=1}^{p+1} \psi_j(u)\chi_j \ .$$

Chapter 6

Automatic hp-Adaptivity in 1D

In this chapter we discuss the one-dimensional version of our hp- algorithm, a scheme that produces automatically a sequence of meshes that deliver not only exponential convergence but also the best meshes in a whole preasymptotic region. We will describe first the main idea of the algorithm, next discuss its practical implementation in the code, and then document our claims with a number of numerical examples. We shall conclude the chapter with a short discussion on the algebraic complexity and efficiency issues.

6.1 The hp algorithm

The algorithm consists of two major components.

Fine grid solution. Given an existing (coarse) mesh, we first refine the mesh in *both* h and p, to obtain a corresponding *fine mesh*. We then solve the problem on the fine mesh and obtain the *fine mesh solution*. In 1D, the fine mesh problem is roughly twice the size of the original problem and we will simply use a direct solver to solve it. In 2D, and especially in 3D, we will resort to an iterative *two mesh solver*, to be discussed later.

Optimal mesh selection. Given the fine mesh solution, we use it to determine optimal mesh refinements of the coarse mesh, by minimizing the *projection based interpolation error* discussed earlier.

The intuition is very simple. We first need to bring additional scales to the solution. Solving on the hp-refined grid provides a *reference solution* with enough of such scales, to make the decision on the optimal mesh refinement possible. The use of a two-grid solver enables the solution of the fine grid problem in many dimensions. One may use other techniques to construct the reference solution. In [105] we used the so called *extraction formulas* to produce a highly accurate postprocessed solution that can be used in place of the fine grid solution. In context of Boundary Elements and integral

equations of the second kind, one might use the Sloan postprocessing [51] to produce the reference solution. The fine grid technique, though, is very general and seems to be the most natural choice in context of finite elements.

The optimal mesh selection technique is related to the algorithm introduced in [105]. It consists of two steps.

Step 1 (local): Determine an optimal refinement for each element of the coarse mesh. We choose between the p-refinement and *competitive* h-refinements that result in the same increase of the number of d.o.f.. More specifically, we assume that the order p may be only increased by one. This implies that orders p_1, p_2 for the h-refined element sons must satisfy the condition:

$$p_1 + p_2 - 1 = p\,, \tag{6.1.1}$$

where p is the original order of the element.

For each of the competing meshes, we solve the local projection problem.

$$\begin{cases} u_{hp}^{new}(a) = u_{h/2,p+1}(a), \quad u_{hp}^{new}(b) = u_{h/2,p+1}(b) \\ |u_{hp}^{new} - u_{h/2,p+1}|_{H^1(a,b)} \rightarrow \min\ . \end{cases}$$

Here (a,b) is the coarse mesh element, $u_{h/2,p+1}$ is the fine mesh solution and u_{hp}^{new} is the result of the projection. For a single (p-refined) element, u_{hp}^{new} is the projection-based interpolant, discussed earlier. For an h-refined element, the projection generalizes the notion of the interpolation. Nevertheless, we will still call it a (generalized) projection-interpolation procedure, comp. also Exercise 6.1.

We also compute the projection-based interpolant u_{hp}^{old} of the fine mesh solution $u_{h/2,p+1}$ for the coarse mesh element, and the corresponding interpolation error.

Once we have computed the interpolants corresponding to all competing h refinements and the p-refinement, we select the winner, i.e. the refinement that produces the smallest interpolation error.

In two cases, however, we cancel the competition. The first case occurs when the p-refinement is selected but we have already reached the maximum order of approximation supported in the code. We force then the h-refinement but determine minimum orders of approximation for the element sons for which the corresponding interpolation error matches the interpolation error for the p-refined element.

The second case takes back the price from h-refinements and it is more tricky. We call it *the case of a missing scale*. It may happen that the optimal h-refinement produces a small (in proportion to the element error) decrease of the local interpolation error,

$$|u_{h/2,p+1} - u_{hp}^{old}|_{H^1}^2 - |u_{h/2,p+1} - u_{hp}^{optimal}|_{H^1}^2 \leq 0.1\,|u_{h/2,p+1} - u_{hp}^{old}|_{H^1}^2\ .$$

Since in the discussed refinement strategy, there is no h-unrefinements, we feel that in such a case we should postpone the decision about breaking the element to the next mesh and we force the

p-refinement (if possible, i.e. the maximum order has not been reached yet). Factor 0.1 reflects the 'smallness' of the missing scale and, up to a certain point, is arbitrary.

Step 2 (global): Determine which elements to refine. The goal is to minimize the global interpolation error (squared):

$$\sum_K \underbrace{|u_{h/2,p+1} - u_{hp}|^2_{1,K}}_{\eta_K} \tag{6.1.2}$$

Once we have identified the optimal way of refining each element, we refine all elements that contribute with at least of one third of the maximum element contribution,

$$\eta_K \geq \frac{1}{3} \max_K \eta_K \, . \tag{6.1.3}$$

Factor $1/3$ above is somehow arbitrary and it reflects an integer version of the method with components of the gradient having been scaled to change between 0 and 1.5. Rounding to the integer values 0,1, we refine all elements for which the gradient component is between .5 and 1.5. Hence the 1/3 factor...

Notice that the refinement criterion is based on the error itself and not the rate with which the interpolation error decreases. We shall discuss this issue in detail in Section 6.4.

Step 3: Enforcing optimal refinements. Once we know which elements to refine and how to refine them, we proceed by unrefining the fine mesh. Elements that have not been selected for refinement at all, are clustered and the original order is restored. Elements that have been selected for a p-refinement, are only clustered. Finally, sons of elements selected for an h-refinement are assigned the optimal order of approximation determined in the first step.

Equivalently, one could dump out the coarse mesh before the mesh optimization starts, determine the optimal refinements, dump in the old mesh, and proceed with optimal refinements of the mesh.

The final, formal hp-algorithm looks as follows.

```
begin with an initial coarse mesh
do until exhausted
      solve the problem on the current mesh
      save data on the current mesh and dumpout the corresponding content
        of the data structure arrays
      refine globally the mesh, first in h, then in p
      solve the problem on the refined mesh, determining new solution u_{h/2,p+1}
      determine the norm of the difference between the two solutions
      if the error is acceptable then STOP
      use u_{h/2,p+1} as reference solution to determine optimal
        refinements of the current mesh
```

```
      dumpin the original mesh data and perform the optimal refinements
enddo
```

We illustrate the procedure with the solution of the 'shock problem' introduced in Assignment 3, with coefficient $\alpha = 60$,

$$u(x) = \mathrm{atan}(60(x - \pi/3)) \; . \tag{6.1.4}$$

The exact solution, along with its FE approximation on the initial mesh of two quadratic elements, is shown in Fig. 6.1. Notice that the FE solution interpolates the exact solution at the vertex nodes (comp. Exercise 6.2). This happens only because of the adaptive integration that we use to integrate the load vector.

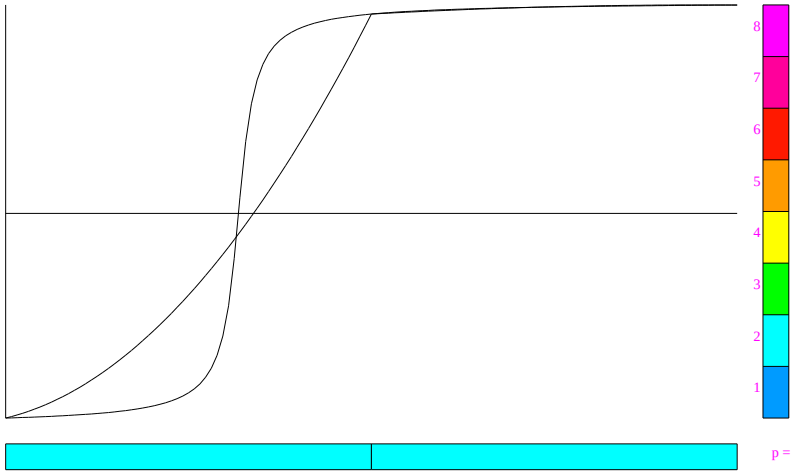


Figure 6.1: 1D shock problem: Initial hp mesh with the corresponding FE solution

We impose the error tolerance $tol = 1.2$ per-cent of the norm of the solution. We use the H^1-seminorm (which happens to be the energy norm for the problem) to compute both the error and the norm.

Figures 6.2 show first 4 consecutive meshes, with the $h/2, p+1$ solution on the left, and optimal meshes on the right. The final mesh with the corresponding solution overlapping exactly the exact solution, is shown in Fig. 6.3.

6.2 Supporting the optimal mesh selection

The automatic hp adaptivity routines are stored in directory *adapt_hp*. There are twelve of them, a total of 1800 lines, quite a lot for a 1D code. Indeed, we consider the complexity of coding to be

the most challenging part to make the hp methods popular...

Routine *adapt_hp/hp_strategy* is the main driver for the hp algorithm, it performs a single mesh optimization step discussed in the previous section. Please open it up.

We begin by storing the information about the current coarse mesh. Our data structure can support only one mesh at a time (it is already complicated enough to support one mesh, isn't ?). The data on the current coarse mesh is stored in *module/ hp_refinements*. A new object - *element_refined* is introduced. It stores a number of attributes for an element in the coarse mesh, including data on the fine mesh solution, optimal refinement flag, optimal order of approximation for the new mesh, etc.

An array consisting of the new objects is allocated dynamically.

Next we solve the problem on the coarse mesh, just a call to the frontal solver.

If the exact solution is known, we proceed with the evaluation of the exact solution norm and the approximation error. We shall use them to report the convergence history.

Next we perform the global hp-refinement, first refining in h and then in p. In that way, middle nodes that are in the coarse mesh, keep their original order of approximation. The fine mesh d.o.f. still supports the coarse grid solution. We shift all d.o.f. from the first to the second position in data structure arrays. The number of solution components must be set to two, *MAXEQNS = 2*. We again call the frontal solver to solve the fine grid problem.

Next we execute *adapt_hp/compute_error*. We use the standard Gaussian quadrature to compute the norm of the difference between the fine and the coarse grid solutions. This is our error estimate and we report it in place of the error, if the exact solution is unknown.

The core of the algorithm resides in routine *adapt_hp/hp_elements* which determines the optimal refinement for each of the coarse mesh elements. We will discuss it separately below. Once we know the optimal refinements, we enforce them using the mesh modification routines discussed in the previous chapter, and we deallocate the array supporting the coarse grid information. We are done.

Determining the optimal refinement for an element. The two core routines to be discussed here are *adapt_hp/project_element* and *adapt_hp/optimize*. Both routines have been written with a minimum interaction with the 1D code data structure. This will enable 'recycling' them in the 2D and 3D codes.

Routine *adapt_hp/project_element* is a 'manager'. It controls a preparation of data for *adapt_hp/optimize*, and receives the corresponding results. Given an element, and the corresponding fine mesh solution, the routine performs one of the two tasks:

- It determines the optimal refinement for the element,

- It determines an h-refinement that matches, in terms of the corresponding projection error, the p-refinement.

. The second action is necessary when we reach the maximum order *NPMAX*, and have to force an h-refinement. In both cases, the preliminary work is identical. The routine determines element stiffness matrices corresponding to the projection based interpolation, for both element sons and the fine grid order $p+1$. As the interpolation is done on the master element, the matrices *are independent of a physical element, and can be precomputed*. The matrices are precomputed in routine *adapt_hp/precompute*, called from the main program, and stored in *module/interpolation*. Thus, instead of integrating the matrices over and over again, we simply copy the necessary portion of them from *module/interpolation*. Notice the form of the matrices corresponding to the 'one shot' interpolation procedure discussed in Section 5.2.

Routine *adapt_hp/optimize* performs three different tasks:

Task 0: It projects (interpolates) the fine grid solution on a smaller FE space, corresponding either to a single element or a two element mesh.

Task 1: Given the interpolation error for the old, coarse mesh element, routine projects the fine grid solution on *all* competitive h-refined meshes, and determines the winner, i.e. the orders p for the element sons that deliver the smallest interpolation error. Recall that an h-refinement is *competitive* if the corresponding number of d.o.f. is equal to that in the p-refined element.

Task 2: Given the interpolation error corresponding to the p-refined element, the routine determines the cheapest (in terms of number of d.o.f.) h-refinement that matches the $p+1$- interpolation error. This is always possible because the meshes are nested.

In each of these cases, the routine starts by computing the element load vectors, and assembling the global matrices corresponding either to the h-refined of unrefined element, and order $p+1$ of the fine mesh. The order of the fine mesh provides an upper limit for the mesh optimization. In the case of the unrefined element, the 'big' element matrices are computed using the h-extension operator coefficients delivered in a convenient, matrix form by routine *adapt_hp/element_ext*.

The routine starts then the iteration process through the meshes to be considered. In the first case, it simple exits the iterations after the first step. The formal algorithm is as follows.

```
begin with an order(s) p prescribed by the calling routine,
do until the specific task is met
  extract the necessary matrices corresponding to the current
    order(s) p
  solve the linear system defining the projection-based interpolant
  compute the corresponding element contributions to the interpolation
```

```
      error
    Task 0: exit
    Task 1: update the smallest error, and the corresponding best
      mesh info
    Task 2: compare the current rate with the rate to be met, and exit
      if done
    increase the order p in one of the elements, according to the error
      distribution
  enddo
```

Please study the presented routines carefully. Much of the presented logic of the implementation will be used in 2D and 3D as well.

6.3 Exponential convergence. Comparing with h-adaptivity

We shall present now results for the two model problems discussed in Section 5.8. We will consider the 'shock problem' for three different values of coefficient $\alpha = 20, 60, 120$, and then consider the problem with the singular solution. We will use the Peano shape functions. As discussed earlier, a FE solution to the 1D Laplace equation is exact at the vertex nodes, and it coincides with the projection based interpolant. Results for more general operators will be presented in the next chapter. For each of the presented cases, we will compare the hp-adaptivity with h-adaptivity using quadratic elements. From our experience, this is the strongest competitor for the hp elements. The h-adaptive algorithm is a 'scaled down' version of the hp algorithm.

```
  begin with an initial coarse mesh
  do until exhausted
    solve the problem on the current mesh
    save data on the current mesh and dumpout the corresponding content
      of the data structure arrays
    refine globally the mesh in h
    solve the problem on the refined mesh, determining new solution u_{h/2}
    determine the norm of the difference between the two solutions
    if the error is acceptable then STOP
    determine the maximum element error η_max = max_K η_K
    dumpin the coarse mesh
    refine each element K for which η_K > 1/3 η_max
  enddo
```

Routines supporting the h-adaptivity can be found in directory *adapt_h*.

The shock problem. In each of the presented cases, we will run the hp-algorithm until the maximum order of approximation is reached. Then we will match the result with h-adaptivity and quadratic elements.

Shock problem. The results for the three values of parameter α are presented in Figures 6.4,6.5, 6.6.

In each of the presented cases, the algorithm delivers exponential convergence with the same rate, equal approximately 1. This is demonstrated by using N^1 scale for the number of d.o.f. N and the logarithmic scale for the error. The meshes deliver a smaller error than h-adaptivity for each number of d.o.f., not only in an asymptotic range. In the last case, $\alpha = 120$, the maximum order is reached comparatively early. Figure 6.7 shows the result for the same case, with a destination error of 0.1 percent. The maximum order of approximation has been reached three times during the refinements.

The singular problem. We have imposed a limit on the number of h-refinement levels (see *module/tree*), *NGEN* = 100. Figure 6.8 presents the final mesh and five zooms on the origin with zooming factors of $2^0, 2^{16}, 2^{32}, 2^{48}, 2^{64}, 2^{80}$. Fig. 6.9 presents the final zoom with factor = 2^{92}, and the convergence rates compared with the corresponding h-adaptive rates. The automatically obtained mesh is known as the *geometrically refined mesh* with refinement factor = $1/2$. The geometrical refinement toward the singularity is accompanied by a gradual ,'linear' increase of order of approximation p away from the singularity. This optimal mesh was constructed analytically by Babuška and Gui in [75]. The corresponding convergence is exponential with error behaving like $e^{\alpha N^{\frac{1}{2}}}$, which has been illustrated by using the algebraic scale on the horizontal axis. The obtained rate confirms the theoretical estimates, see [113], pp. 90. Notice that the hp-strategy consistently outperforms the h-adaptivity.

You may be surprised with two facts: it took 100 refinements to increase the order of approximation to level $p = 4$ only, and after so many (optimal) refinements the final error is still 0.07 percent of the norm. At a 'practical level' of one percent error, the p-refinements do not occur at all, and the h-refinements seem to be more than sufficient. This is not the case. The presented 1D mesh optimization used H^1 norm, to be replaced by weaker $H^{\frac{1}{2}}$ norm in 2D, and L^2 norm in 3D. The order p starts increasing then much earlier.

Finally notice the typical behavior of the algorithm at the singularity point. Even if we start with a linear mesh only, the algorithm raises first from $p = 1$ to $p = 3$. Only then it 'sees' the singularity and breaks the element into a cubic element element at the singularity and a linear element next to it. The situation repeats itself and the algorithm keeps breaking the *key element* with the same optimal h-refinement pattern. Eventually, the error in the key element decreases to the error level in the large elements, and the h-refinement of the key element is accompanied with a p-refinement of

large elements. This is the mechanism of producing the optimal, geometrically graded meshes.

6.4 Discussion of the hp algorithm

In the original algorithm in [105], we had considered the possibility of adding more than one d.o.f. per element. This is possible if we interpolate a given function, or a highly accurate postprocessed solution. In [105], we use Babuška's extraction formulas for such a postprocessing. Eventually we dropped this idea as the use of extraction formulas is possible only for linear elliptic problems with constant coefficients, and their application is very expensive. The idea of using the fine mesh solution is problem-independent, and seems to be more feasible.

Since the fine mesh is obtained by raising the order of approximation by one, it makes sense to consider only a p-refinement in which the order increases by one only. This leads naturally to the idea of the competitive h-refinements. A comparison of interpolation errors corresponding to a p-refined, and an h-refined element (with the original p for the element sons) favors almost always the h-refinement. If we compare the rates instead,

$$\frac{\text{old interpolation error - new interpolation error}}{\text{number of d.o.f. added}} ,$$

the p-refinement is declared to be a winner. With neither of the two strategies, we were able to reproduce the optimal geometrical meshes for singular problems, or obtain exponential convergence. Only once we have allowed for the 'flow' of d.o.f. between the h-refined element sons, we have observed the optimal reported behavior. Notice that the strategy results in optimal meshes that are nested in the h, but not in the p sense.

Selecting which elements to refine, based on the interpolation error rather then the rate of decrease of the error, is necessary.

Fig. 6.10 presents convergence history for the shock problem with parameter $\alpha = 120$, and element (error) indicators replaced with

$$\eta_K = |u_{h/2,p+1} - u_{hp}^{old}|^2_{1,K} - |u_{h/2,p+1} - u_{hp}^{opt}|^2_{1,K} ,$$

where u_{hp}^{old} is the interpolant corresponding to the old mesh, and u_{hp}^{opt} denotes the interpolant corresponding to the optimal refinement. The algorithm produces a prolonged plateau of very slow convergence and looses dramatically to the h-refinements. We understand what has happened if we look at the FE error corresponding to one of the meshes producing the plateau, see Fig. 6.11. The fourth element contributes with the max error (we integrate the derivative to compute the error !), and the corresponding error function in the element, is practically an even function. Increasing order from $p = 2$ to $p = 3$ means adding an odd shape function, which does not result in a decrease of the error. The competitive h-refinements, due to the symmetry of the error, do not win. Obviously, what is needed, is an increase of the order to $p = 4$. The error is still 'seen' by the fine mesh. Enforcing

the refinements of elements with large error solves the problem.

Speaking philosophically, the refinement strategy cannot be based just on a quick profit return only. The error and *not* the rate of its decrease decides where we invest.

Finally, we remind (comp. Exercise 6.2) that, for the 1D Laplace (Poisson) equation, with no integration error, FE solution coincides with the exact solution at the vertex nodes. Consequently, the interpolation error coincides with the FE error. In a general case, see the discussion in the next chapter, the two errors converge to each other asymptotically. Consequently, replacing the interpolation error with the approximation error in the mesh refinement criterion produces (asymptotically) the same meshes.

6.5 Algebraic complexity and reliability of the algorithm

By now, I hope, we have convinced ourselves that the hp-adaptive strategy delivers error that decreases exponentially as a function of problem size N,

$$\|e_h\| \approx Ce^{\alpha N^\beta} ,$$

with $\beta = \frac{1}{2}$ for the singular solution and greater than $\frac{1}{2}$ for solutions 'analytic up to the boundary'. For the model shock problem we observed $\beta = 1$. What about the error vs. cost of the computations? A quick answer is very simple: the cost of the fine grid and coarse grid solutions, as well as the cost on the mesh optimization algorithm increases algebraically in N, while the error decreases exponentially. The exponential function eventually always 'wins', so, *asymptotically*, we shall always see the error decreasing exponentially with cost of computations as well.

The danger is that, with finite resources *and finite precision*, we may never get into that asymptotic region of exponential convergence. First of all, let us emphasize again (we consider this to be a very important point) that the hp-algorithm produces *always* a mesh better than h-adaptivity and, as a function of problem size N, the error seems to behave exponentially from the very beginning, without any 'warm up' preasymptotic behavior. The cost, however, deserves a more systematic analysis.

Cost of the linear solver. For hp meshes, it is convenient to estimate the number of operations in terms of number of elements n and (average) order p. In 1D, the band width is of order p, number of dof of order $N = np$, and the number of operations for the frontal solver version of Gaussian elimination is of order $N \times p^2 = np^3$. The cost of the fine grid problem is of the same order $2n(p+1)^3$. The cost of integration of element matrices for a single element is of order: number of integration points $p+1$ $\times$ number of shape functions $p+1$ squared, a total of order of p^3. Multiplying by number of elements n, we end up with the same order np^3 as for the Gaussian elimination.

Cost of mesh optimization. We have the following major components.

Computation of interpolation matrices. This part is negligible because the interpolation takes place on the master element, element matrices are precomputed, and the cost of assembling the global stiffness matrix corresponding to the two elements mesh of order $p+1$, is of lower order p^2.

Assembling single element matrices. Recall that the single element matrix is computed using the h-extension operator. The cost of the two matrix-matrix multiplies is of order p^3.

Solution of a subsystem of equations. Given a particular distribution of orders p for the element sons, we extract the corresponding matrices with size of order p, and solve the corresponding projection problem. If we use a dense matrix solver, then the cost of a single solve is p^3, and we have to solve such a subsystem of order p times (number of competitive h-refinements). This gives an estimate of order p^4. This problem can be resolved in two ways. The first solution is very specific for the 1D case. If we choose to work with integrated Legendre polynomials, the interpolation matrix is simply *diagonal* and the cost of the solution becomes all together negligible, comp. Exercise 6.1. The second solution explores the fact that the extraction process produces a sequences of nested meshes, we only add new d.o.f. Exploring this fact in the construction of the linear solver allows for bringing the cost of the whole process back to order p^3.

Error calculation. This operation cost is of order = number of integration points $p+1\times$ number of shape functions $p+1$, a total of order p^2, and it may be neglected.

In summary, the cost of the solver and the mesh optimization are of the same order np^3. Solving the fine system of equations costs only slightly more then twice the cost of the coarse system. As we insist on using a direct solve for the coarse problem, attempting to use an iterative solver for the fine problem makes little sense.

6.5.1 Error estimation.

The issue of cost cannot be discussed without relating it to the fundamental problem of *reliability of computations*. In this book, we shall not discuss any *a posteriori error estimation* techniques. For recent, excellent monographs on the subject see [4, 16]. Any (multidimensional) a-posteriori error estimate technique brings into the model additional scales, and compares the current solution with a corresponding improved solution. This is exactly what we do, when we compute the difference between the coarse and fine grid solutions. We invite the reader to rerun all the discussed examples and watch the corresponding effectivity indices,

$$\text{effectivity index} = \frac{\|u_{h/2,p+1} - u_{hp}\|}{\|u - u_h p}\,,$$

or, in terms of percent of energy,

$$\text{effectivity index} = \frac{\frac{\|u_{h/2,p+1}-u_{hp}\|}{\|u_{h/2,p+1}\|}}{\frac{\|u-u_hp}{\|u\|}} \ .$$

Regardless of the definition, the observed effectivity indices are very close to one and, in all cases, they converge asymptotically to one (comp. Exercise 6.3).

The technique (as any other error estimation technique) will work, only if the fine grid 'sees the solution', and it cannot replace the human expertize. We shall return to this point in subsequent chapters.

Exercises

Exercise 6.1 Discuss an optimal implementation of the projection-based interpolation for a single element of order p, and then for a refined element, i.e. a group of two elements with orders p_1, p_2. Use Legendre polynomials to arrive at a diagonal stiffness matrix.

Exercise 6.2 Prove that the FE solution of the 1D Laplace (Poisson) equation,

$$-u'' = f \ ,$$

interpolates the exact solution at the vertex nodes, i.e. $u_h(x_j) = u(x)$ for each vertex node x_j. Consequently, for this particular problem, the projection-based interpolant *coincides* with the FE solution. *Hint:* Construct the Green function,

$$-G'' = \delta(x - x_j) \ ,$$

and use the fact that it can be reproduced with the FE vertex basis functions.

Exercise 6.3 Prove that the 'a-posteriori error estimate' used in the code,

$$|u - u_h|_{H^1(0,l)} \approx |u_{h/2,p+1} - u_h|_{H^1(0,l)} \ ,$$

is *asymptotically exact*, i.e.

$$\frac{|u_{h/2,p+1} - u_h|_{H^1(0,l)}}{|u - u_h|_{H^1(0,l)}} \to 0 \ .$$

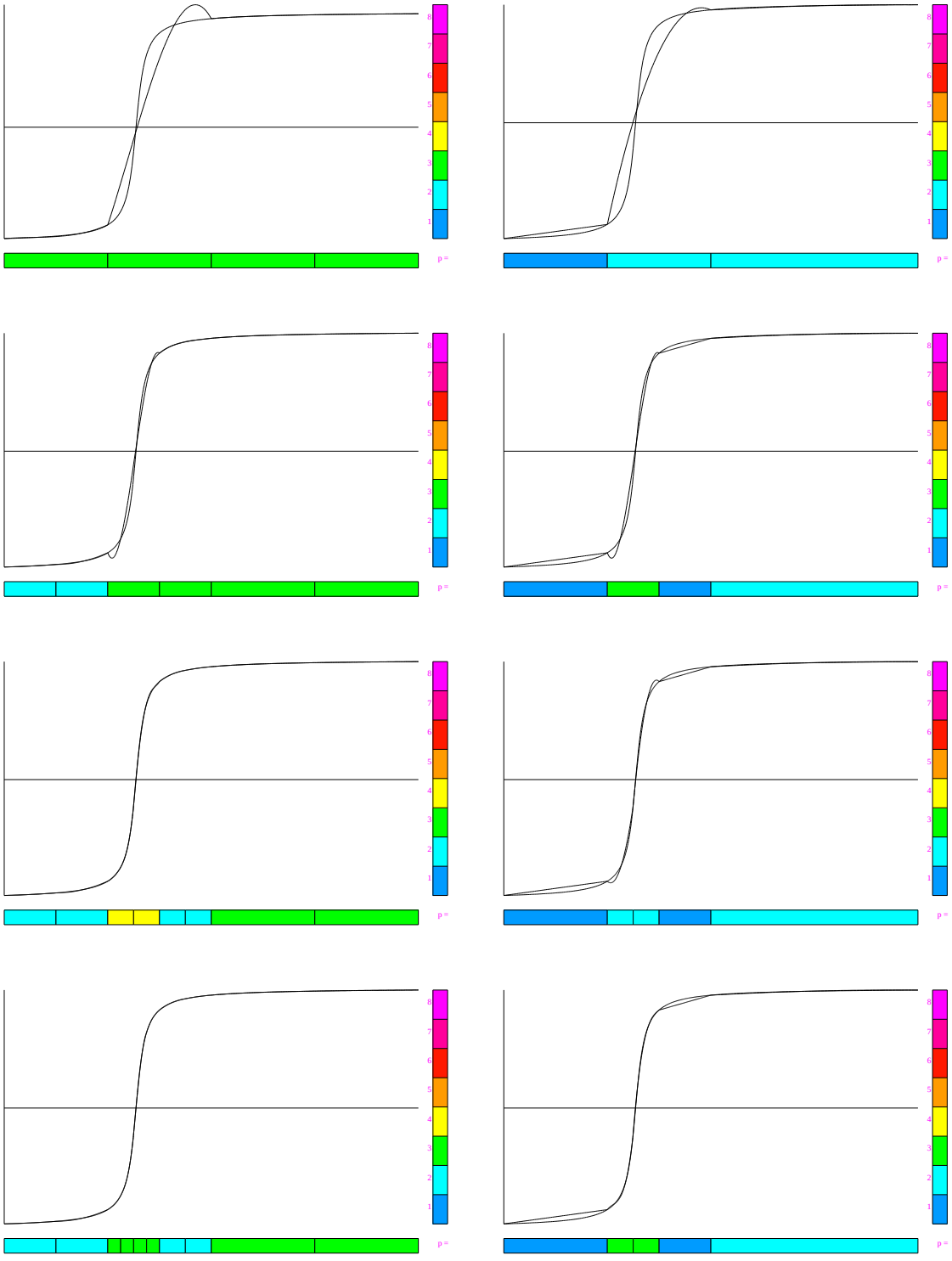


Figure 6.2: Example1: Stages 1-4 of the optimal hp mesh selection. Fine mesh reference solutions on the left, optimal hp interpolations on the right

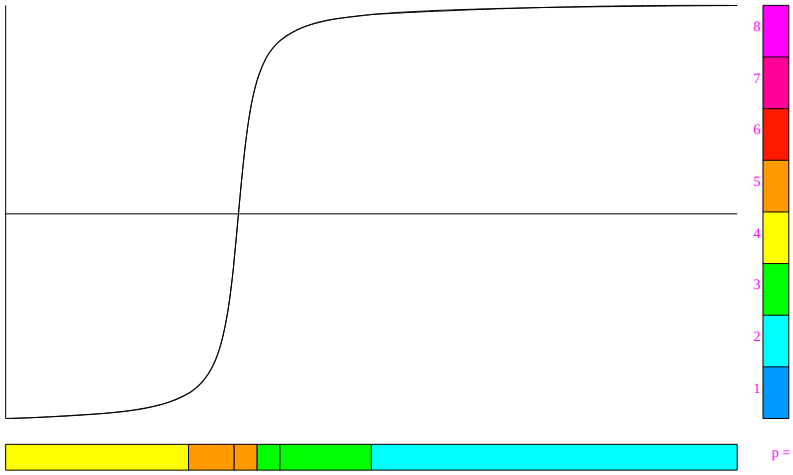


Figure 6.3: 1D shock problem: Final, optimal hp mesh that delivers a FE solution with 1.2% error. Can you distinguish the approximate and exact solutions?

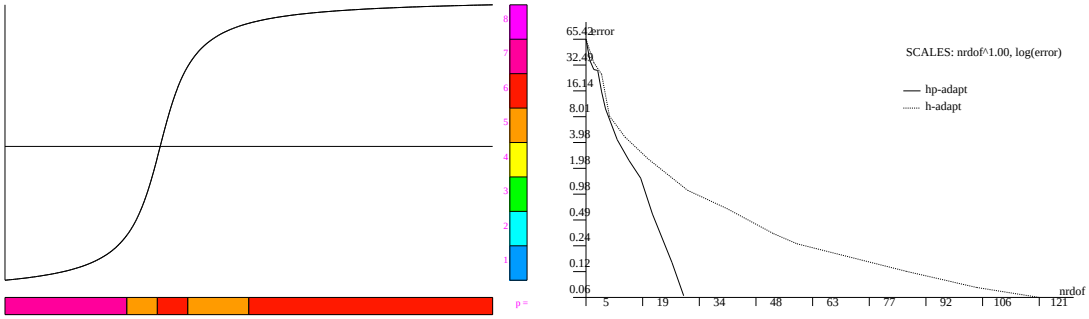


Figure 6.4: 1D shock problem with $\alpha = 20$. Optimal hp mesh and convergence history

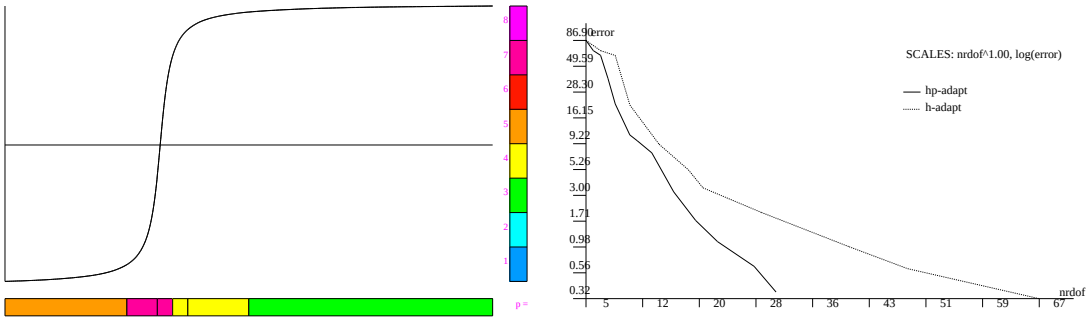


Figure 6.5: 1D shock problem with $\alpha = 60$. Optimal hp mesh and convergence history

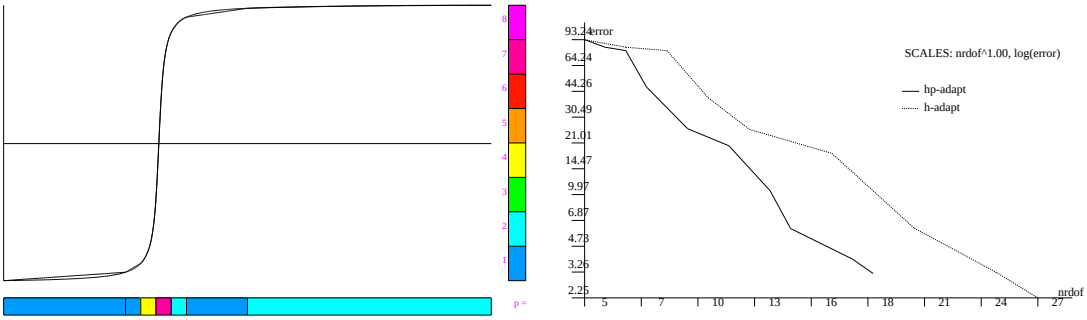


Figure 6.6: 1D shock problem with $\alpha = 120$. Optimal hp mesh and convergence history

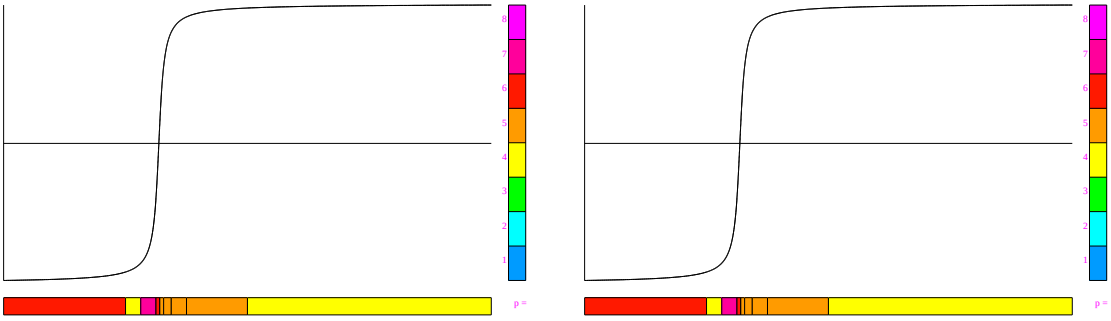


Figure 6.7: 1D shock problem with $\alpha = 120$. Optimal hp mesh and convergence history with a goal of 0.1 percent error

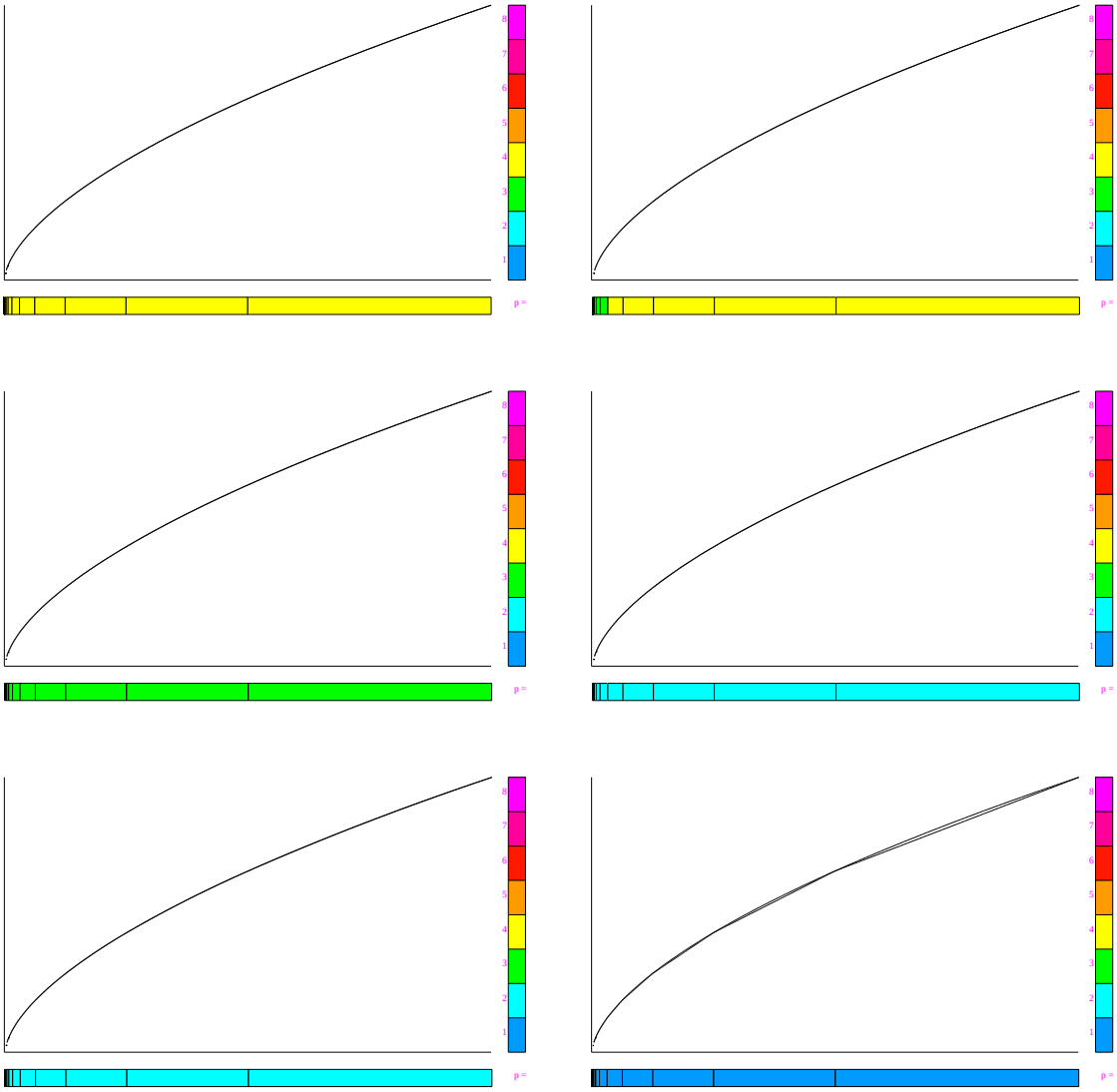


Figure 6.8: Example2: Zooms on the singularity point with zooming factors = $2^0, 2^{16}, 2^{32}, 2^{48}, 2^{64}, 2^{80}$

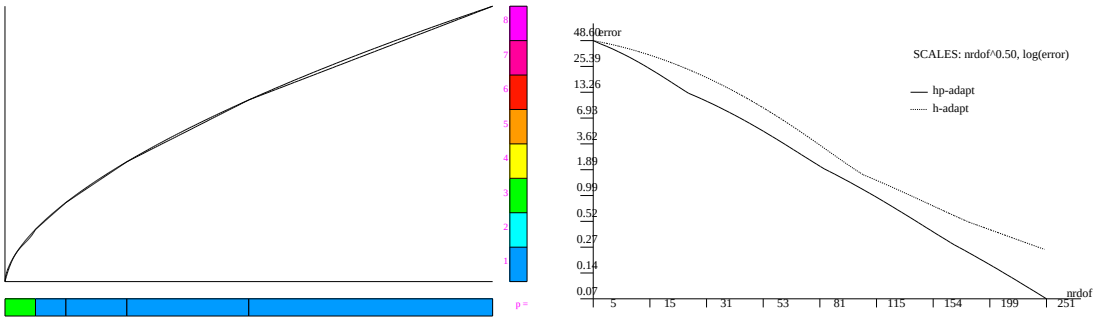


Figure 6.9: Example2: Zoom on the singularity point with zooming factor = 2^{92}, and the convergence history

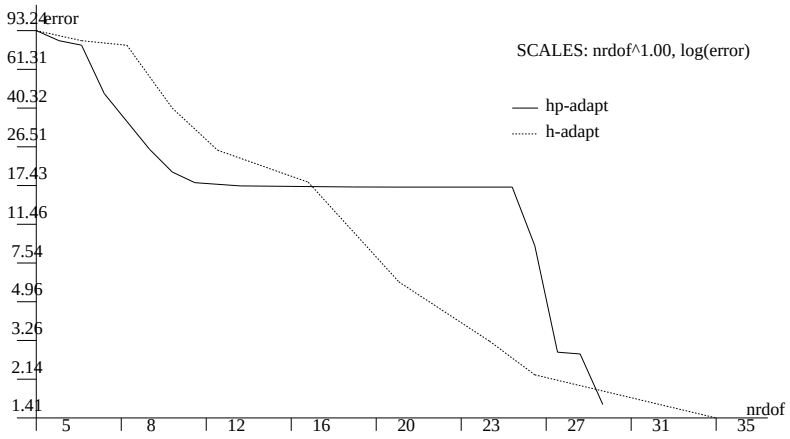


Figure 6.10: 1D shock problem with $\alpha = 120$. Convergence history for the algorithm based on controlling the rates

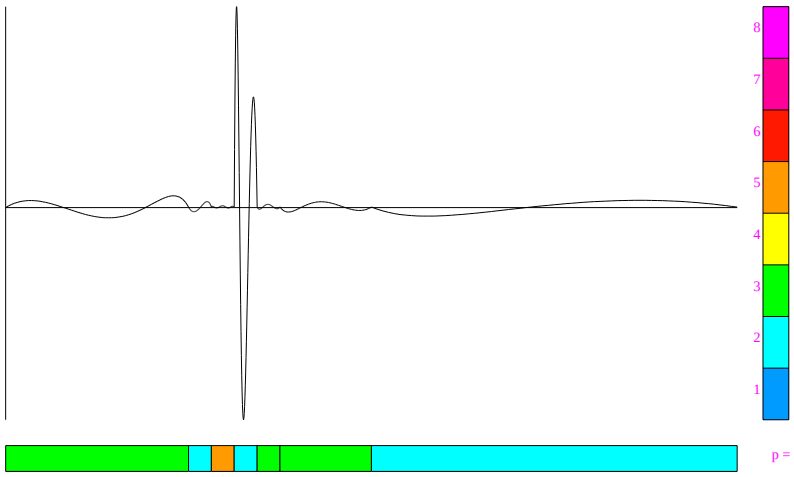


Figure 6.11: 1D shock problem with $\alpha = 220$. FE error function for an intermediate FE mesh

Chapter 7

Wave Propagation Problems

In this chapter we study a representative class of indefinite problems related to wave propagation. As a representative example, we will use the problem of a vibrating elastic bar discussed in Section 1.6.3.

7.1 *Convergence analysis for non coercive problems*

A general, non positive-definite boundary-value problem has to be analyzed with a stronger tool than the Lax-Milgram theorem. We shall use the same algebraic framework as in Section 1.5 with the coercivity assumption replaced by a new condition.

So, consider again the abstract boundary-value problem,

$$\begin{cases} u \in V \\ b(u,v) = l(v), \quad \forall v \in V \, . \end{cases}$$

First of all, let us observe that, in general, we may have test functions v that are b-orthogonal to all $u \in V$,

$$V_0 = \{v \in V \, : \, b(u,v) = 0, \quad \forall u \in V\} \, .$$

The simplest example of such a case is provided by bilinear form,

$$b(u,v) = \int_0^1 uv' \, dx, \quad u, v \in H^1(0,1) \, ,$$

for which the subspace V_0 consists of all constant functions.

Consequently, in the general case, the right-hand side cannot be arbitrary but it must satisfy a *compatibility condition*,

$$l(v) = 0, \quad \forall v \in V_0 \, .$$

We are ready now to formulate a new sufficient condition for the existence and uniqueness of solutions to the abstract variational problem.

THEOREM 7

(Banach Closed Range Theorem)

Let X be a Hilbert space with a subspace V, and let $b(u,v)$ denote a bilinear form defined on $X \times X$ which is continuous, i.e. there exists $M > 0$ such that,

$$|b(u,v)| \leq M\|u\|\|v\|, \quad \forall u, v \in X \,,$$

and it satisfies the inf-sup condition, i.e. there exists a constant $\gamma > 0$ such that,

$$\inf_{u \in V, u \neq 0} \sup_{v \in V, v \neq 0} \frac{|b(u,v)|}{\|u\|\|v\|} \geq \gamma > 0.$$

Let $l(v)$ be a linear and continuous functional defined on V, i.e.

$$|l(v)| \leq C\|v\|, \quad \forall v \in V \,,$$

and $\tilde{u}_0 \in X$ be an element of X, such that the compatibility condition is satisfied,

$$l(v) - b(\tilde{u}_0, v) = 0, \quad \forall v \in V_0 \,.$$

Then, the abstract VBVP,

$$\begin{cases} u \in \tilde{u}_0 + V \\ b(u,v) = l(v), \quad \forall v \in V \,, \end{cases}$$

has a unique solution u such that

$$\|u\| \leq \|\tilde{u}_0\|(1 + \frac{M}{\gamma}) + \frac{C}{\gamma}$$

Here $\|\cdot\|$ denotes the norm in space X. ∎

Proof: See [98], page 583. The theorem does not cover the case of a non unique solution.

We return now to the Galerkin approximation of the variational problem,

$$\begin{cases} u_h \in \tilde{u}_0 + V_h \\ b(u_h, v_h) = l(v_h), \quad \forall v_h \in V_h \,, \end{cases}$$

where $V_h \subset V$ is a sequence of finite-dimensional approximate spaces.

Corresponding to Theorem 7, we have the fundamental convergence result due to Babuška [17].

THEOREM 8

(Ivo Babuška, 1971)

Suppose, the discrete $\inf \sup$ *condition holds,*

$$\inf_{u_h \in V_h, v_h \neq 0} \sup_{v_h \in V_h, v_h \neq 0} \frac{|b(u_h, v_h)|}{\|u_h\| \, \|v_h\|} =: \gamma_h > 0 \,. \tag{7.1.1}$$

Then

- *The approximate problem has a unique solution u_h, and*

$$\|u_h\| \leq \frac{\|l\|}{\gamma_h} \,. \tag{7.1.2}$$

- *We have the following estimate,*

$$\|u - u_h\| \leq \left(1 + \frac{M}{\gamma_h}\right) \inf_{w_h \in \tilde{u}_0 + V_h} \|u - w_h\| \,, \tag{7.1.3}$$

where M is the continuity constant for form $b(v,u)$.

∎

Proof: The stability result follows immediately from the inf-sup condition,

$$\|u_h\| \leq \frac{1}{\gamma_h} \sup_{v_h \in V_h} \frac{|l(v_h)|}{\|v_h\|} \leq \frac{\|l\|}{\gamma_h} \,.$$

Stability implies uniqueness and, as the number of equations is equal to the number of unknowns, uniqueness implies existence. Next, using the discrete $\inf \sup$ condition, we have,

$$\begin{aligned} \gamma_h \|u_h - w_h\| &\leq \sup_{v_h \neq 0} \frac{|b(v_h, u_h - w_h)|}{\|v_h\|} \\ &= \sup_{v_h \neq 0} \frac{|b(v_h, u - w_h)|}{\|v_h\|} \leq M \|u - w_h\| \,, \end{aligned}$$

and, by triangle inequality,

$$\begin{aligned} \|u - u_h\| &\leq \|u - w_h\| + \|w_h - u_h\| \\ &\leq \left(1 + \frac{M}{\gamma_h}\right) \|u - w_h\| \,. \end{aligned}$$

∎

Only if the discrete inf-sup constant is mesh-independent, the approximation error is bounded by (a mesh-independent constant times) the best approximation error. Contrary to the coercive case, the discrete stability is not longer implied by the stability on the continuous level.

Notice finally that, on the discrete level, there is no need to introduce the compatibility condition. Indeed, for a square matrix (discrete operator), the matrix is nonsingular if an only if its transpose is nonsingular as well.

Interpretation with the Projection Operator. Every bilinear form $b(u, v)$ defines a (linear) projection (comp. [98], page 172) from space of test functions V onto its finite dimensional counterpart V_h,

$$\begin{cases} P_h \,:\, V \to V_h \\ b(P_h u, v_h) = b(u, v_h) \quad \forall v_h \in V_h \,. \end{cases} \tag{7.1.4}$$

If the bilinear form defines a scalar product, i.e. it is symmetric and positive-definite, we have an *orthogonal projection*.

Suppose we already know that the exact solution u exists. Due to the relation between linear form and the exact solution, $l(v) = b(u, v)$, the approximate VBVP can be rewritten as:

$$\begin{cases} u_h \in \tilde{u}_0 + V_h \\ b(u_h, v_h) = b(u, v_h) \quad \forall v_h \in V_h \,, \end{cases}$$

or, equivalently,

$$u_h - \tilde{u}_0 = P_h(u - \tilde{u}_0) \,.$$

In the case of homogeneous Dirichlet BC, $u_h = P_h u$. In other words, solving the approximate VBVB is equivalent to projecting the exact solution onto the approximate space, with the modification involving the lift of non-homogeneous Dirichlet data.

The approximation error can now be represented in a concise way as,

$$u - u_h = u - \tilde{u}_0 - (u_h - \tilde{u}_0) = (I - P_h)(u - \tilde{u}_0) \,,$$

where I is the identity operator. Finally, making use of the fact that projection P_h reduces to identity on approximate space V_h, we have,

$$u - u_h = (I - P_h)(u - (\tilde{u}_0 + v_h)) \quad \forall v_h \in V_h \,.$$

This implies that,

$$\|u - u_h\| \leq \|I - P_h\| \inf_{v_h \in V_h} \|u - (\tilde{u}_0 + v_h)\| \,.$$

Obviously,

$$\|I - P_h\| \leq \|I\| + \|P_h\| = 1 + \|P_h\| \,.$$

We say that the approximation is *stable* if we can bound the norm of projection operator P_h with a mesh-independent constant,

$$\|P_h\| \leq C \,.$$

Thus, if the best approximation error converges to zero, stability implies convergence [1].

In the case of a symmetric and positive-definite bilinear form $b(u, v)$, both operators P_h and $I - P_h$ are orthogonal projections (wrt the inner product defined by form $b(u, v)$), and $\|I - P_h\| =$

[1] The common phrase is : *stability and approximability imply convergence*

$\|P_h\| = 1$ which again means that the Galerkin method delivers the *best solution* (in terms of the energy norm).

It is less trivial to show [86] that, in the case of a Hilbert space and an *arbitrary linear projection*, we also have

$$\|I - P_h\| = \|P_h\| ,$$

although the common norm is no longer equal one [2]. This improves the estimate from the original proof of Babuška as,

$$\|P_h\| \leq \frac{M}{\gamma_h} .$$

Note that in the coercive case $\gamma_h = \alpha$.

7.2 *Wave propagation problems*

The example of a vibrating elastic bar discussed in Section 1.6.3 falls into a general class of problems where the bilinear form is obtained by adding to a coercive form a compact perturbation,

$$b(u, v) = a(u, v) + c(u, v) .$$

A sufficiently general framework can be established by introducing a second Hilbert space H, such that there exists a compact map $T : V \to H$, i.e. for every sequence $v_n \in V$, converging *weakly* in V to a limit v_0, Tv_n converges strongly in H to Tv_0. A primary example is provided by Sobolev spaces, space $H^s(0, l)$ being compactly embedded in $H^r(0, l)$, for any $r > s$, see [2], page 144. Any bilinear form $c(u, v)$ defined and continuous on $H \times V$,

$$|c(u, v)| \leq C\|u\|_H\|v\|_V , u \in H, v \in V$$

is then compact.

In the case of our model problem,

$$c(u, v) = -\omega^2 \int_0^l \rho uv \, dx + i\omega\rho_f c_f u(l)v(l) .$$

The integral term is continuous on $L^2(0, l) \times H^1(0, l)$, and the embedding $H^1 \to L^2$ is compact. The second term can be identified as a continuous bilinear form on $\mathbb{C} \times H^1(0, l)$,

$$c(z, v) = i\omega\rho_f c_f zv(l), \quad z \in \mathbb{C}, v \in H^1(0, l) ,$$

and the trace map,

$$H^1(0, 1) \ni u \to u(l) \in \mathbb{C} ,$$

[2] The result does not extend to Banach spaces

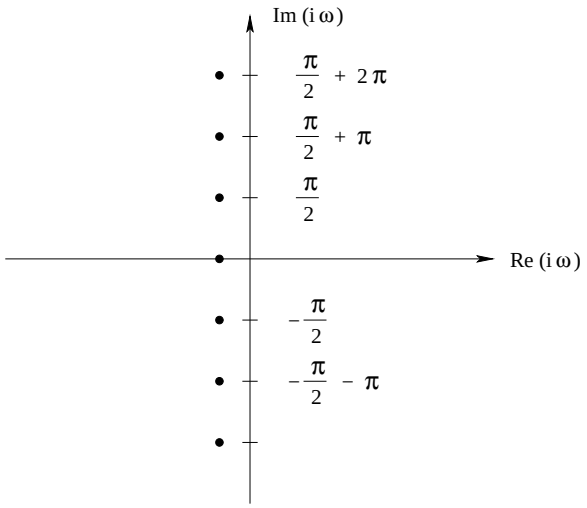


Figure 7.1: Resonant frequencies for an elastic bar interacting with an acoustical medium

is compact as well, comp. Exercise 7.1. Consequently, bilinear form $c(u,v)$ is compact.

For a coercive form $a(u,v)$ and a compact form $c(u,v)$ we can consider an eigenvalue problem,

$$\begin{cases} u \in V, u \neq 0, \ \lambda \in \mathbb{C} \\ b(u,v) = \lambda c(u,v), \quad \forall v \in V \ . \end{cases}$$

One can prove [20, 98] that the problem above admits a sequence of eigenvalues $\lambda_n \to 0$, plus possibly a zero eigenvalue. If -1 does not belong to the sequence, i.e.

$$a(u,v) + c(u,v) = 0, \quad \forall v \in V \ ,$$

does not admit a non trivial solution, then the corresponding bilinear form satisfies automatically the inf-sup condition, see Exercise 7.2. Non trivial solutions to our model problem correspond to frequencies $i\omega$ identified as the *resonant frequencies* (also called scattering frequencies, [88]) illustrated in Fig. 7.1, compare Exercise 7.3, [38]. Due to the transfer of energy from the bar into the acoustical medium (radiation damping), all resonant frequencies have a negative real part. Consequently, our problem admits a unique solution for all values of ω, and the (continuous) inf-sup condition is always satisfied.

A similar reasoning can be applied to the discrete case as well. All discrete resonant frequencies $i\omega_h$ have negative real parts, and the discrete problem is always stable. Moreover, one can prove the following result.

THEOREM 9

Let $b(u,v) = a(u,v) + c(u,v)$ be the sum of a coercive form $a(u,v)$ and a compact perturbation

$c(u,v)$. *Assume that the homogeneous problem* $b(u,v)=0, \forall v \in V$, *admits no non-trivial solution, and denote by* γ *the corresponding inf-sup constant for bilinear form* $b(u,v)$.

Let $V_h \subset V$ *be a sequence of approximating finite-dimensional space, satisfying the* **approximability assumption**,

$$\forall v \in V \exists v_h \in V_h \text{ such that } \|v - v_h\| \to 0 \, .$$

Then, there exists an $h_0 > 0$ *such that for every* $h \leq h_0$, *the discrete problem is stable, i.e.* $\gamma_h > 0$, *and*

$$\gamma_h \to \gamma \, .$$

∎

Proof: See [37].

So, the good news are that, asymptotically in h, we are O.K., i.e. the discrete problem is stable and, therefore, the Galerkin method will deliver an optimal (up to a multiplicative constant) approximation. The bad news are, however, that we do not know the threshold value h_0. The theorem tells us also that we should watch carefully the continuous inf-sup constant. *Things do not get better in the limit !* If the continuous γ is small then we expect the discrete problem (due to the finite precision) to be numerically unstable.

The inf sup constant for our model problem is a function of two factors: frequency ω and ratio of $\rho_f c_f$ to ρc. The problem is more stable if the fluid is 'heavier' and the sound speed in fluid is greater, and the stability constant decreases algebraically with frequency. In other words, for a small radiation damping (and close to a resonant frequency), as well as for large frequencies, *we will not be able to solve the problem* using the Galerkin method.

Aubin-Nitsche duality argument. As we have seen, the compact perturbation can be identified by introducing a weaker norm, and verifying the the perturbation is continuous in the weaker norm. Given a weaker norm (think L^2 norm for problems formulated in H^1), we can ask a natural question about the convergence in that norm. If we relax the norm, do we get a faster convergence ? The following reasoning is known as the *Aubin-Nitsche argument*.

We begin by introducing the *adjoint (dual) problem*,

$$\left\{ \begin{array}{l} v \in H^1(0,l), v(0) = 0 \\ \displaystyle\int_0^l \left\{ a \delta u' v' + b \delta u' v + cuv \right\} \, dx + \beta u(l) v(l) = \int_0^l \delta u \, \psi \, dx \\ \quad \text{for every test function } \delta u \in H^1(0,l), u(0) = 0 \, . \end{array} \right. \tag{7.2.5}$$

We are using still the same bilinear form and space of test functions, but the role of arguments have changed, the second argument signifies the solution, whereas the first argument has become a test

function. The linear functional is specified in terms of the L^2-product of the test function δu, and a prescribed function $\psi \in L^2(0,l)$. Notice that the functional is continuous in the weaker L^2-norm,

$$|\int_0^l \delta u\, \psi\, dx| \leq \|\delta u\|_{L^2(0,l)} \|\psi\|_{L^2(0,l)} .$$

The dual problem can be written in the abstract form,

$$\begin{cases} v \in V \\ b(\delta u, v) = (\delta u, \psi), \quad \forall \delta u \in V , \end{cases}$$

with $(\delta u, \psi)$ denoting the L^2-product.

One can show that (Exercise 7.5), due to the regularity of the right-hand side, the solution v of the dual problem, sought in space $H^1(0,l)$, is actually more regular, $v \in H^2(0,l)$. Conversely, any $v \in H^2(0,l), v(0) = 0, a(l)v'(l) + \beta v(l) = 0$, defines a unique function $\psi \in L^2(0,l)$ such that the v is the solution of the dual problem with the load defined by ψ. In other words, the map,

$$\{v \in H^2(0,l), v(0) = 0, a(l)v'(l) + \beta v(l) = 0\} =: W \ni v \to \psi \in L^2(0,l) , \tag{7.2.6}$$

is a bijection. It is also continuous (Exercise 7.6). It follows from the Banach Open Map Theorem that the inverse of the map must be continuous as well. This is a very strong result. Given function $\psi \in L^2(0,l)$, not only we know that the corresponding solution $v = v(\psi)$ lives in the smaller space W but also that the dependence of v upon ψ (measured in the stronger, H^2-norm) is *uniform*,

$$\|v(\psi)\|_{H^2(0,l)} \leq C \|\psi\|_{L^(0,l)} \quad \forall \psi \in L^2(0,l) .$$

Constant $C > 0$ depends upon the geometry (length l) and material data, but it is independent of load ψ.

We set now ψ to the FE error function, $\psi = u - u_h$. We have,

$$\begin{aligned}
\|u - u_h\|^2_{L^2(0,l)} &= (u - u_h, u - u_h)_{L^2(0,l)} \\
&= b(u - u_h, v) && \text{(definition of the adjoint problem)} \\
&= b(u - u_h, v - \Pi_{hp} v) && \text{(orthogonality relation (2.1.4))} \\
&\leq M \|u - u_h\|_{H^1(0,l)} \|v - \Pi_{hp} v\|_{H^1(0,l)} && \text{(continuity of form } b(u,v)\text{)} \\
&\leq CM \|u - u_h\|_{H^1(0,l)} \frac{h}{p} \|v\|_{H^2(0,l)} && \text{(interpolation error estimate)} \\
&\leq CM \|u - u_h\|_{H^1(0,l)} \frac{h}{p} \|u - u_h\|_{L^2(0,l)} . && \text{(continuous dependence of } v \text{ upon } \psi\text{)}
\end{aligned}$$

Here C is a generic positive constant, and $\Pi_{hp} v$ denotes the (projection-based) interpolant of solution v to the adjoint problem. Consequently,

$$\|u - u_h\|_{L^2(0,l)} \leq C \frac{h}{p} \|u - u_h\|_{H^1(0,l)} . \tag{7.2.7}$$

In the weaker, L^2-norm the convergence is *faster* by the h/p factor.

7.3 Asymptotic optimality of the Galerkin method

Recall that for the case of a symmetric and positive-definite bilinear form, the Galerkin method delivers *the best* approximation in terms of the energy norm,

$$\|u - u_h\|_E = \min_{w_h \in \tilde{u}_0 + V_h} \|u - w_h\|_E \, .$$

Here the energy norm was defined using the symmetric and positive-definite form $b(u,v)$,

$$\|u\|_E^2 = b(u,u) \, .$$

First of all, the result extends immediately to sesquilinear, positive definite forms (Exercise 2.1).

If the bilinear form $b(u,v)$ is obtained by perturbing a sesquilinear, coervive form $a(u,v)$ with a compact perturbation $c(u,v)$, the same result will hold asymptotically,

$$\|u - u_h\|_E \leq C(h) \min_{w_h \in \tilde{u}_0 + V_h} \|u - w_h\|_E, \quad C(h) \to 1, \quad h \to 0 \, .$$

For those readers that have skipped the two previous sections, let us simply say that $c(u,v)$ includes all first and zero integral contributions, as well as the boundary term,

$$c(u,v) = \int_0^l \{bu'v + cuv\} \, dx + \beta u(l) v(l) \, .$$

Thus, for our 1D model problem, as long as $a(x) > a_{min} > 0$ is real and bounded away from zero, the problem fits into the discussed category. The compact perturbation is continuous in weaker norm,

$$|c(u,v)| \leq C \|u\|_{H^1(0,l)} \|v\|_{L^2(0,1)} \, .$$

Constant C depends upon the maximum values of material data $b(x)$ and $c(x)$.

The orthogonality condition (2.1.4) implies that,

$$a(u - u_{,} u - u_h) + c(u - u_h, u - u_h) = a(u - u_{,} u - v_h) + c(u - u_h, u - v_h) \, ,$$

where v_h is the *best approximation* (in energy norm) of exact solution u. At the same time, using the equivalence of the H^1- and energy norms, we have,

$$a(u - u_{,} u - u_h) + c(u - u_h, u - u_h) \geq \|u - u_h\|_E^2 - C \|u - u_h\|_E \|u - u_h\|_{L^2(0,l)} \, ,$$

and,

$$a(u - u_{,} u - v_h) + c(u - u_h, u - v_h) \leq \|u - u_h\|_E \|u - v_h\|_E + C \|u - u_h\|_E \|u - v_h\|_{L^2(0,l)} \, .$$

Consequently,

$$\|u - u_h\|_E - C \|u - u_h\|_{L^2(0,l)} \leq \|u - v_h\|_E + C \|u - v_h\|_{L^2(0,l)} \, .$$

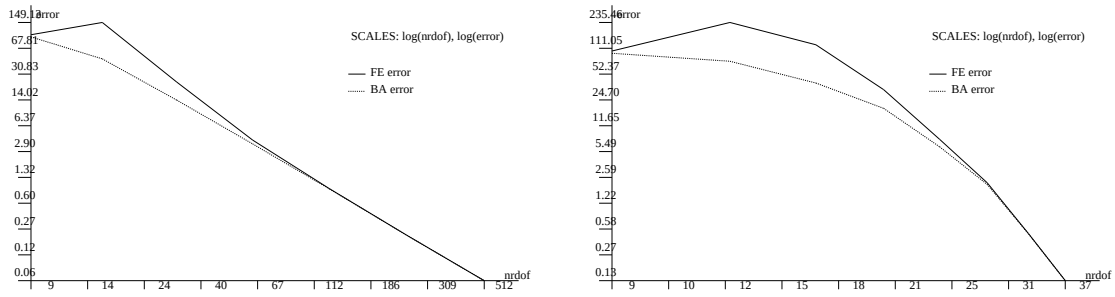


Figure 7.2: Vibrations of an elastic bar. FE and best approximation errors for uniform h- ($p = 2$) and p-refinements.

However, we have learned in the previous section, comp. (7.2.7), that $\|u - u_h\|_{L^2(0,l)}$ converges to zero faster than the error measured in the H^1 or (equivalent) energy norm. In fact,

$$\|u - u_h\|_{L^2(0,l)} \leq C\frac{h}{p}\|u - u_h\|_E \ .$$

The same argument applies to the best approximation error, explain, why? Consequently, the final result reads,

$$\|u - u_h\|_E(1 - C\frac{h}{p}) \leq \|u - v_h\|_E(1 + C\frac{h}{p}) \ .$$

We shall illustrate now the result with the vibrating bar problem. Introducing non-dimensional quantities:

$$x = \frac{x}{l}, \quad u = \frac{u}{l} \ ,$$

we rewrite the problem as,

$$\begin{cases} -u'' - k^2 u = 0 \quad \text{in } (0,1) \\ u(0) = u_0 \\ u'(1) - ikru(1) = 0 \ , \end{cases} \tag{7.3.8}$$

with nondimensional wave number k and ratio r defined as:

$$k = \frac{\omega}{c}l, \quad c^2 = \sqrt{\frac{E}{\rho}}, \quad r = \frac{\rho_f c_f}{\rho c} \ .$$

Figure 7.2 compares the FE error with the best approximation error for uniform h- and p-refinements, for material data $k = 32, r = 0.015$. In both cases, the FE error experiences a delay but eventually it catches up with the best approximation error and, starting at an error level of about 1 percent, the two errors are practically equal.

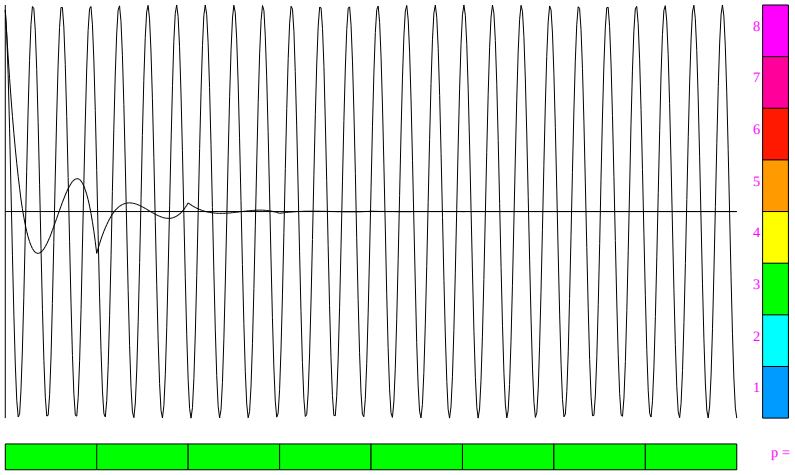


Figure 7.3: Vibrations of an elastic bar, $k = 160$. Exact solution, initial fine mesh and the corresponding FE solution

7.4 Dispersion error analysis

It is of an utmost importance for wave propagation problems to start the mesh optimization process with an initial coarse mesh such that, for the corresponding fine mesh, the Galerkin method delivers a solution close to the best approximation. This necessity is illustrated running the hp-adaptive algorithm for the vibrating bar problem, for the case of a large wave number, $k = 160, r = 0.015$. The initial fine mesh, with the corresponding FE solution is presented in Fig. 7.3. It is clear that the mesh is nowhere close enough to capture features of the exact solution. No wonder then that the convergence results presented in Fig. 7.4 are completely chaotic in its first stage, and the exponential convergence is observed only after the fine mesh has become 'fine enough'. The final coarse mesh, shown in Fig. 7.5, reflects very much the chaotic refinements phase, and displays a rather random distribution of element size and order. It is clear thus that the optimization process should start only once we are confident that the fine mesh delivers a solution close to the optimal, best approximation result.

Figure 7.6 demonstrates one more time the behavior of the FE error, compared with the best approximation (BA) error, for the case of uniform h-refinements, and a large wave number. Three phases in the convergence process are clearly visible. In the first, preasymptotic, phase, both the FE and BA errors stay at the level of 100 percent. This reflects the simple fact that, with too few d.o.f., it is simply impossible to reproduce the wave. In the second phase, the BA error behaves in the way predicted by the interpolation error estimates, for the uniform h-refinements, we observe the rate of convergence equal to the order of approximation, in our case $p = 2$. The actual FE error,

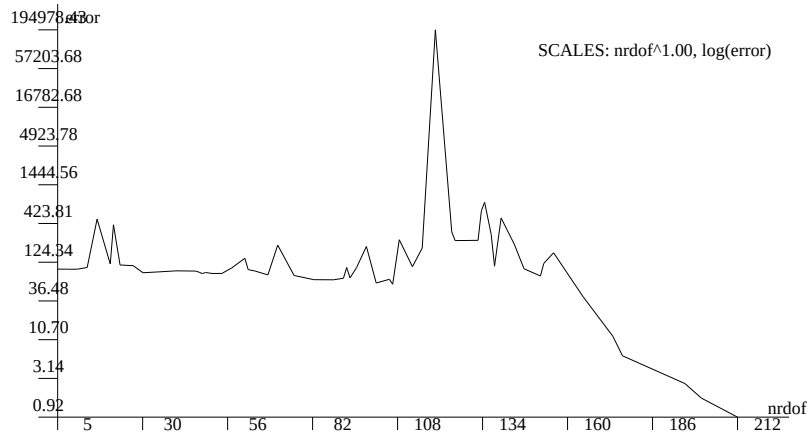


Figure 7.4: Vibrations of an elastic bar, $k = 160$. Convergence performance for the hp-algorithm

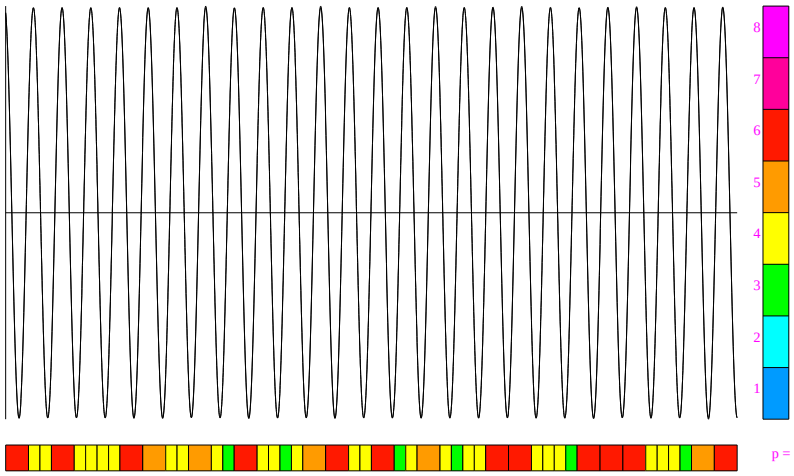


Figure 7.5: Vibrations of an elastic bar, $k = 160$. Final mesh

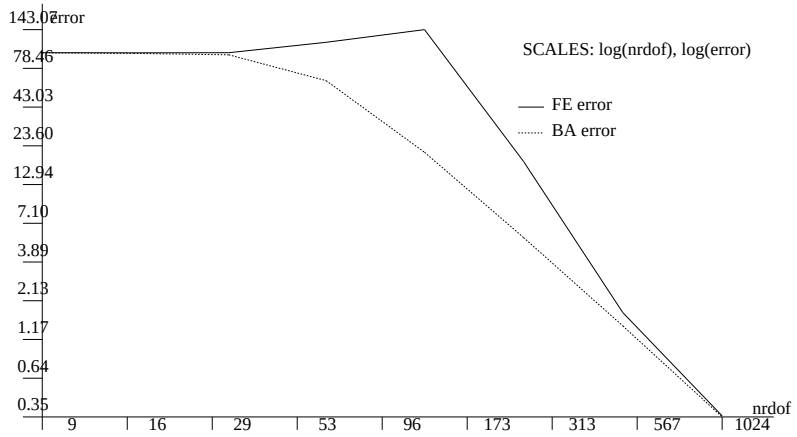


Figure 7.6: Vibrations of an elastic bar, $k = 160$. Uniform h-convergence. FE error compared with best approximation error

however, experiences a delay in entering the region of asymptotic behavior, in fact, it initially grows over the 100 percent level [3], and only after a while it starts catching up with the best approximation error. The 'catching up' behavior produces a misleading effect of a higher convergence rate that predicted by the interpolation theory. Finally, in the third, fully asymptotic phase, starting at the level of about 1 percent error, the FE and BA errors are the same. Ihlenburg and Babuška (see [81], and the literature therein). have coined a name of a *pollution error* for the second phase and the difference between the FE an BA errors,

$$\text{FE error = BA error + pollution error,}$$

and investigated the behavior of the pollution error as a function of (non-dimensional) wave number k, (uniform) element size h and order of approximation p. It turns out that the leading term in the pollution error coincides with that in the classical *dispersion error* analysis.

In the dispersion analysis, given a uniform mesh with element size h and order p, we are concerned with reproducing on the mesh a discrete equivalent of the wave $u = e^{ikx}$. More precisely, if $x_j = jh$, we seek a solution to the discrete system of equations in the form,

$$u_h(x_j) = e^{ik_{hp}x_j} = e^{ik_{hp}jh} ,$$

with a discrete, mesh dependent wave number k_{hp}. The difference between the continuous and discrete wave numbers is called the *dispersion error*,

$$\text{dispersion error } = k - k_{hp} .$$

[3] That initial growth of the error seems to be more severe with higher order elements

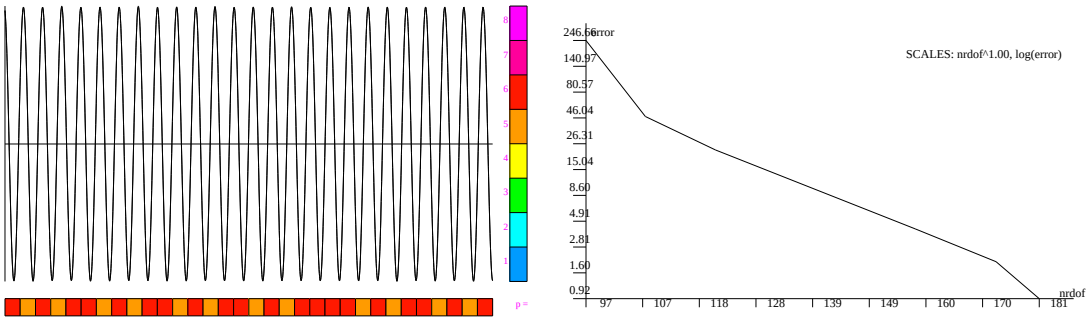


Figure 7.7: Vibrations of an elastic bar, $k = 160$. Optimal hp mesh and the convergence history, starting with an initial mesh determined by the dispersion error estimate

Ihlenburg's result confirms the conventional wisdom that one should not trust a mesh that cannot reproduce the wave with a sufficiently small dispersion error. The dispersion (pollution) error is represented by a rather complicated formula [81],

$$|k - k_{hp}| \approx k \left(\frac{ekh}{4p} \right)^{2p} p^{-\frac{1}{2}} . \tag{7.4.9}$$

We use it to determine the initial mesh. More precisely, we assume that the geometry of the mesh is given, and determine a minimum order of approximation p such that the dispersion error for the *fine mesh* is smaller than a prescribed tolerance *tol*. For instance, with *tol* = 0.05, and an initial mesh of 30 uniform elements, the corresponding order for the initial mesh is set to $p = 4$. The hp-algorithm produces then practically uniform p-refinements. The final mesh and the corresponding convergence history are shown in Fig. 7.7 Obviously, in the case of the 1D problem, the entire mesh adaptivity is really very academic. Once the initial mesh is determined using the dispersion error analysis, it is sufficient to use uniform p-refinements. In the multidimensional case, though, the mesh has to resolve not only the wave form of the solution but singularities coming from geometrical irregularities and material interfaces as well.

The initial (fine) mesh dispersion error is controlled by either raising p or breaking the elements. Since the pollution error decreases faster for the p-method[4], we give a preference to higher order p and use the h-refinement only if the maximum order p is reached.

Exercises

Exercise 7.1 Prove that any continuous map from a Hilbert space V into a finite-dimensional space H is compact.

[4]The pollution error seems to be a function of the overall error level, and for smooth wave solutions, the error decreases faster for p-refi nements, compare also discussion in [81]

Exercise 7.2 Variational problem,

$$\begin{cases} u \in V \\ a(u,v) + c(u,v) = l(v), \quad \forall v \in V \ , \end{cases}$$

can be expressed in terms of linear operators as,

$$(A + C)u = l \ ,$$

where $A, C \ : \ V \to V'$ are the linear operators corresponding to forms $a(u,v),\ c(u,v)$, respectively. Apply A^{-1} to both sides of the equation to show that the problem satisfies the *Fredholm Alternative*, see [98], page 507.

Exercise 7.3 Derive the following formula for the resonant frequencies of problem (7.3.8).

$$2ik_n = \mathrm{Ln}\left(-\frac{1+r}{1-r}\right) + 2n\pi \ .$$

Exercise 7.4 Reformulate the vibrating bar problem in such a way that the resonant frequencies can be interpreted as eigenvalues of a linear operator. *Hint:* Introduce new variables and replace the single equation with a system.

Exercise 7.5 Prove that the solution of the adjoint problem (7.2.5) lies in space $H^2(0,l)$. *Hint:* Interpret the differential equation in the distributional sense.

Exercise 7.6 Show that map (7.2.6) is continuous.

Exercise 7.7 In the case of a viscoelastic material, and time-harmonic vibrations, see [38], the Young constant E is complex-valued. The corresponding sesquilinear form $a(u,v)$ representing the leading term,

$$a(u,v) = \int_0^l Eu'\bar{v}' \, dx \ ,$$

is not longer hermitian. Do you expect the Galerkin method still to deliver the (asymptotically) optimal result? In what norm ?

Part II

TWO-DIMENSIONAL ELLIPTIC PROBLEMS

Chapter 8

2D Elliptic Boundary-Value Problem

We begin by formulating the problem of interest: an elliptic boundary-value problem (BVP) in $\mathbb{R}^2$. We shall follow the same same steps as in 1D, discuss the classical and variational formulation, identify the algebraic structure of the problem, and discuss the equivalence with a minimization problem.

8.1 Classical formulation

We begin with geometry. Let Ω be a bounded *domain* in $\mathbb{R}^2$. Domain is a set that is both *open* and *connected*. A set is open if each of its points can be enclosed with a (possibly small) ball, contained in the set. Intuitively speaking, open sets have "no boundary", like a peeled potato. A set is connected, if each two of its points can be connected with a curve enclosed in the set. It comes in "one piece". Finally, a set is bounded, if it can be enclosed in a (sufficiently large) ball. The case of an *unbounded domain* is more difficult, we shall not discuss it for now. We shall split boundary $\Gamma = \partial\Omega$ of the domain into three disjoint parts (not necessary connected, each part may have several segments) Γ_D, Γ_N, and Γ_C, on which *Dirichlet*, *Neumann* and *Cauchy* boundary conditions will be imposed.

We now set up in domain Ω the following *boundary-value problem*, in short BVP. Find $u(\boldsymbol{x}), \boldsymbol{x} \in$

$\bar{\Omega}$ (closure of Ω, i.e. the domain plus its boundary), such that

$$\left\{\begin{array}{rl} -\sum_{i=1}^{2}\frac{\partial}{\partial x_i}\left(\sum_{j=1}^{2}a_{ij}(\boldsymbol{x})\frac{\partial u}{\partial x_j}\right)+\sum_{j=1}^{2}b_j(\boldsymbol{x})\frac{\partial u}{\partial x_j}+c(\boldsymbol{x})u &= f(\boldsymbol{x}) \qquad \text{in } \Omega \\ u &= u_D(\boldsymbol{x}) \qquad \text{on } \Gamma_D \\ \sum_{i=1}^{2}\sum_{j=1}^{2}a_{ij}(\boldsymbol{x})\frac{\partial u}{\partial x_j}n_i &= g(\boldsymbol{x}) \qquad \text{on } \Gamma_N \\ \sum_{i=1}^{2}\sum_{j=1}^{2}a_{ij}(\boldsymbol{x})\frac{\partial u}{\partial x_j}n_i+\beta(\boldsymbol{x})u &= g(\boldsymbol{x}) \qquad \text{on } \Gamma_C\,. \end{array}\right. \tag{8.1.1}$$

The left-hand side of the differential equation defines a differential operator Lu. Functions $a_{ij}(\boldsymbol{x}), b_j(\boldsymbol{x}), c(\boldsymbol{x})$ are the *coefficients of the differential operator*, and they are defined in the whole domain Ω. By including argument $\boldsymbol{x}$ in the notation, we emphasize that they may vary within the domain. Coefficients $a_{ij}(\boldsymbol{x})$ enter also Neumann and Cauchy boundary operators, and must be defined on the corresponding parts of the boundary, too. In most of practical applications, the coefficients reflect *material properties*, and there are piecewise constant. Neumann boundary condition (BC) can be treated as a special case of Cauchy BC with coefficient $\beta(\boldsymbol{x}) = 0$. Motivated with physical applications, we shall identify coefficients $a_{ij}(\boldsymbol{x}), b_j(\boldsymbol{x}), c(\boldsymbol{x}), \beta(\boldsymbol{x})$ as *material data*. Functions $u_D(\boldsymbol{x}), f(\boldsymbol{x}), g(\boldsymbol{x})$, defined on the corresponding parts of the boundary, will be call the *load data*. Notice that we use the same symbol $g(\boldsymbol{x})$ to denote the right-hand side of Neumann and Cauchy BC's. Customary, to simplify the notation, we write argument $\boldsymbol{x}$ neither for solution $u = u(\boldsymbol{x})$ nor for its derivatives. Vector $\boldsymbol{n}$, with components n_i, stands for the *exterior unit vector* defined on boundary Γ, see Fig. 8.1.

Throughout the rest of the book, we shall use the *Einstein summation convention*. Whenever we write an algebraic expression with two identical indices, we mean actually the sum of such expressions with the index changing within its range, clear from context. This allows for simplifying the statement of the BVP to the form:

$$\left\{\begin{array}{rl} -\frac{\partial}{\partial x_i}\left(a_{ij}(\boldsymbol{x})\frac{\partial u}{\partial x_j}\right)+b_j(\boldsymbol{x})\frac{\partial u}{\partial x_j}+c(\boldsymbol{x})u &= f(\boldsymbol{x}) \qquad \text{in } \Omega \\ u &= u_D(\boldsymbol{x}) \qquad \text{on } \Gamma_D \\ a_{ij}(\boldsymbol{x})\frac{\partial u}{\partial x_j}n_i &= g(\boldsymbol{x}) \qquad \text{on } \Gamma_N \\ a_{ij}(\boldsymbol{x})\frac{\partial u}{\partial x_j}n_i+\beta(\boldsymbol{x})u &= g(\boldsymbol{x}) \qquad \text{on } \Gamma_C\,. \end{array}\right.$$

The differential operator has been written in the so-called *divergence form*, natural for most applications and ready for the integration by parts to be discussed next. If we perform the external

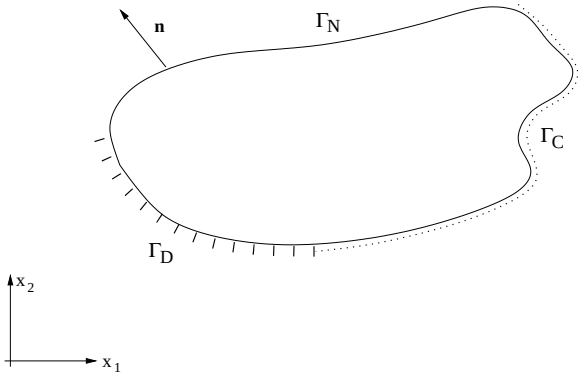


Figure 8.1: Elliptic boundary-value problem in $\mathbb{R}^2$

differentiation, we can rewrite the formula for the differential operator in a more classical form,

$$Lu = -a_{ij}\frac{\partial^2 u}{\partial x_i \partial x_j} + \bar{b_j}\frac{\partial u}{\partial x_j} + cu \,,$$

where coefficients,

$$\bar{b}_j = b_j + \frac{\partial a_{ij}}{\partial x_i} \,.$$

As the mixed second order derivatives are independent of the order of differentiation, the off-diagonal terms of matrix a_{ij} do not enter the differential equation individually and what it matters is their sum only. This motivates us to assume the symmetry condition,

$$a_{ij} = a_{ji} \,.$$

The operator L and the corresponding BVP are called *elliptic* if matrix a_{ij} is positive definite, i.e.

$$a_{ij}(\boldsymbol{x})\xi_i\xi_j > 0 \quad \forall \boldsymbol{\xi} \neq \mathbf{0}$$

As a symmetric 2×2 matrix has two real eigenvalues λ_1, λ_2, this is equivalent to say that $\lambda_{min} = \min\{\lambda_1, \lambda_2\} > 0$, and the ellipticity condition can be rewritten in a slightly more informative form,

$$a_{ij}(\boldsymbol{x})\xi_i\xi_j \geq \lambda_{min}\, \xi_i\xi_i \,.$$

The minimum eigenvalue, in general, depends upon $\boldsymbol{x}$. If it admits a lower bound in domain Ω,

$$\lambda_{min}(\boldsymbol{x}) \geq c > 0, \quad \forall \boldsymbol{x} \in \Omega \,,$$

we will say that the problem is *uniformly elliptic*. In such a case, the condition on ellipticity can be rewritten in a more concise form,

$$a_{ij}(\boldsymbol{x})\xi_i\xi_j \geq c\, \xi_i\xi_i \quad \forall \boldsymbol{x} \in \Omega \,, \tag{8.1.2}$$

where *ellipticity constant* $c > 0$. The 2D (uniform) ellipticity condition (8.1.2) is a 2D equivalent of the ellipticity assumption in 1D: $a(x) \geq a_{min} > 0$. The coefficients corresponding to the lower order terms do not influence the classification and, at this point, they are arbitrary.

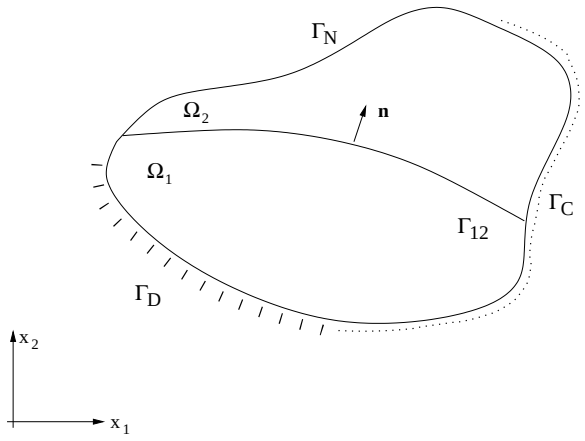


Figure 8.2: Interface problem in $\mathbb{R}^2$

Interface problems. As in the 1D case, the material data are often only piece-wise regular. Across material interfaces normal derivative of the solution becomes discontinuous and, consequently, some of the second derivatives do not exist in the classical sense. The original BVP must be replaced with an interface problem formulation.

We shall consider, for simplicity, the case of a single interface only. Domain Ω is split into two subdomains Ω_1, Ω_2 separated by interface Γ_{12}, with a normal unit $\boldsymbol{n}$ directed from Ω_1 to Ω_2, see Fig. 8.2. Following the partition of the domain, we also split the three parts of the boundary,

$$\Gamma_D^I = \Gamma_D \cap \partial\Omega_I,\ \Gamma_N^I = \Gamma_N \cap \partial\Omega_I,\ \Gamma_C^I = \Gamma_C \cap \partial\Omega_I,\ I = 1, 2\,,$$

and assume that all material and load data are specified piece-wise, with branches $a_{ij}^I, b_j^I, c^I, d_D^I, g^I, \beta^I, I = 1, 2$.

The original formulation, applied for the subdomains, is now accompanied with additional *interface conditions*.

$$\left\{\begin{array}{rcll} -\dfrac{\partial}{\partial x_i}\left(a_{ij}^I(\boldsymbol{x})\dfrac{\partial u^I}{\partial x_j}\right) + b_j^I(\boldsymbol{x})\dfrac{\partial u^I}{\partial x_j} + c^I(\boldsymbol{x})u^I &= f^I(\boldsymbol{x}) & \text{in } \Omega_I & \\ u^I &= u_D^I(\boldsymbol{x}) & \text{on } \Gamma_D^I & \\ a_{ij}^I(\boldsymbol{x})\dfrac{\partial u^I}{\partial x_j}n_i &= g^I(\boldsymbol{x}) & \text{on } \Gamma_N^I & \\ a_{ij}^I(\boldsymbol{x})\dfrac{\partial u^I}{\partial x_j}n_i + \beta^I(\boldsymbol{x})u^I &= g^I(\boldsymbol{x}) & \text{on } \Gamma_C^I\,, & \\ & & \text{for } I = 1, 2, \text{ and} & \\ u^1 = u^2, \quad a_{ij}^1\dfrac{\partial u^1}{\partial x_j}n_i = a_{ij}^2\dfrac{\partial u^2}{\partial x_j}n_i & & \text{on } \Gamma_{12}\,, & \end{array}\right. \tag{8.1.3}$$

with u^1, u^2 denoting the two branches of the solution. As in 1D, the second interface condition expresses continuity of the normal flux defined as $a_{ij}\frac{\partial u}{\partial x_j}n_i$.

Regularity of the solution. With the assumption that the material and data are regular within each of the subdomains, solution to the 1D interface problem could experience a loss of regularity only across the material interface. The situation is much more delicate in 2D. Derivatives of the solution may become singular at corner points with an internal angle exceeding π (non-convex domains), at boundary points separating different boundary conditions, and at interior points where three or more material interfaces meet. Additional irregularities of the solution may be caused by irregular load data, e.g. a discontinuous Dirichlet BC data u_D.

8.2 Variational (weak) formulation

The working horse for deriving the variational formulation is the *elementary integration by parts* formula (comp. Exercise 8.1) ,

$$\int_\Omega \frac{\partial u}{\partial x_i} v \, d\boldsymbol{x} = -\int_\Omega u \frac{\partial v}{\partial x_i} \, d\boldsymbol{x} + \int_\Gamma u v n_i \, ds \, . \tag{8.2.4}$$

The formula allows for transfer of the derivative from function u onto function v at the cost of introducing the boundary term. Here n_i is the i-th component of outward unit vector on boundary Γ.

We first consider the original BVP.

Let $v(\boldsymbol{x})$ be an arbitrary test function defined in domain Ω and on its boundary. We multiply the differential equation *pointwise* with $v(\boldsymbol{x})$ and then integrate over domain Ω to obtain,

$$\int_\Omega \left\{ \frac{\partial}{\partial x_i} \left(a_{ij} \frac{\partial u}{\partial x_j} \right) + b_j \frac{\partial u}{\partial x_j} + cu \right\} v \, d\boldsymbol{x} = \int_\Omega f v \, d\boldsymbol{x} \, .$$

We can use now the integration by parts formula to transfer the external derivatives in the differential operator onto the test function (divergence form comes handy...) to obtain,

$$\int_\Omega \left(a_{ij} \frac{\partial u}{\partial x_j} \frac{\partial v}{\partial x_i} + b_j \frac{\partial u}{\partial x_j} v + cuv \right) \, d\boldsymbol{x} - \int_\Gamma a_{ij} \frac{\partial u}{\partial x_j} n_i v \, ds = \int_\Omega f v \, d\boldsymbol{x} \, .$$

The differential operator in the boundary term that we have arrived at, is precisely the operator used in the Neumann and Cauchy BC's. We can use now those conditions to represent the first order derivatives (flux) on the boundary with load data $g(\boldsymbol{x})$ and solution $u(\boldsymbol{x})$ (Cauchy's BC only),

$$\begin{aligned} a_{ij} \frac{\partial u}{\partial x_j} n_i &= g && \text{on } \Gamma_N \\ a_{ij} \frac{\partial u}{\partial x_j} n_i &= g - \beta u && \text{on } \Gamma_C \, . \end{aligned}$$

On the other side, we have no information about the first derivatives on Dirichlet boundary Γ_D. Consequently, we *shall not test* on Γ_D, and assume that the test function vanishes there,

$$v(\boldsymbol{x}) = 0 \quad \boldsymbol{x} \in \Gamma_D \, .$$

Upon *building the Neumann and Cauchy BC's into* the integral identity, we separate the terms with unknown solution u on the left, from the known data on the right, to obtain,

$$\int_\Omega \left(a_{ij}\frac{\partial u}{\partial x_j}\frac{\partial v}{\partial x_i} + b_j\frac{\partial u}{\partial x_j}v + cuv \right) d\boldsymbol{x} + \int_{\Gamma_C} \beta uv\, ds = \int_\Omega fv\, d\boldsymbol{x} + \int_{\Gamma_N\cup\Gamma_C} gv\, ds\,.$$

This integral identity, to be satisfied *for every test function* $v(\boldsymbol{x})$, is the *variational* or *weak formulation* of the BVP (VBVP). Notice that we have not managed to build the Dirichlet BC into the formulation so we still have to state it explicitly by writing:

$$\begin{cases} u = u_D \text{ on } \Gamma_D \\ \displaystyle\int_\Omega \left(a_{ij}\frac{\partial u}{\partial x_j}\frac{\partial v}{\partial x_i} + b_j\frac{\partial u}{\partial x_j}v + cuv \right) d\boldsymbol{x} + \int_{\Gamma_C} \beta uv\, ds = \int_\Omega fv\, d\boldsymbol{x} + \int_{\Gamma_N\cup\Gamma_C} gv\, ds\,, \\ \qquad\qquad \forall v:\ v = 0 \text{ on } \Gamma_D\,. \end{cases} \tag{8.2.5}$$

We have demonstrated that every solution to the classical BVB satisfies also the VBVP. We will demonstrate now that, conversely, any solution to the VBVP, satisfies also the classical BVP. The proof is based on a more general version of Fourier's lemma.

THEOREM 10

(Fourier's lemma)
Let Ω *be an* n*-dimensional manifold* [1].
Let f *be a continuous function defined on* Ω *such that*

$$\int_\Omega fv\, d\Omega = 0 \quad ,$$

for every continuous test function v *that vanishes on boundary* $\partial\Omega$.
Then f *must vanish,* $f = 0$. ∎

Remember that dropping the condition on test functions to vanish on the boundary would make the theorem *weaker*. It is sufficient to test with functions that vanish on the boundary to conclude that tested function f (we assume no BC for f) must vanish.

We can now proceed with our demonstration. Let u be a solution to the VBVP. By reversing the integration by parts and regrouping the terms, we can conclude that

$$\begin{gathered} \int_\Omega \left\{ -\frac{\partial}{\partial x_i}\left(a_{ij}\frac{\partial u}{\partial x_j} \right) + b_j\frac{\partial u}{\partial x_j} + cu - f \right\} v\, d\boldsymbol{x} + \\ \int_{\Gamma_N} \left\{ a_{ij}\frac{\partial u}{\partial x_j}n_i - g \right\} v\, ds + \int_{\Gamma_C} \left\{ a_{ij}\frac{\partial u}{\partial x_j}n_i + \beta u - g \right\} v\, ds = 0\,, \end{gathered} \tag{8.2.6}$$

for every test function v.

We now proceed in three steps.

[1] The notion includes domain Ω and parts of its boundary

Recovering differential equation. We first test with functions v that vanish on the *whole boundary*. This makes the boundary terms disappear and, by Fourier's lemma, we conclude that

$$-\frac{\partial}{\partial x_i}\left(a_{ij}\frac{\partial u}{\partial x_j}\right)+b_j\frac{\partial u}{\partial x_j}+cu-f=0$$

We have recovered the differential equation.

Recovering the Neumann condition. Now that we know that the differential equation is satisfied, we know that the domain integral in (8.2) must vanish always, i.e. *for all* test functions, not only for those that vanish on the boundary. Consequently, we can write it off, and we end up with a reduced identity involving only boundary integrals,

$$\int_{\Gamma_N}\left\{a_{ij}\frac{\partial u}{\partial x_j}n_i-g\right\}\,ds+\int_{\Gamma_C}\left\{a_{ij}\frac{\partial u}{\partial x_j}n_i+\beta u-g\right\}\,ds=0\,, \tag{8.2.7}$$

for every test function v. So we can test now only with test functions that vanish on Γ_C. This makes the second boundary integral vanish and, by Fourier's lemma applied to Neumann boundary Γ_N, we conclude that,

$$a_{ij}\frac{\partial u}{\partial x_j}n_i-g=0 \text{ on } \Gamma_D\,.$$

We have recovered the Neumann BC.

Recovering Cauchy BC. We repeat our 'trick'. Once we know that the Neumann BC has been satisfied, we can conclude that Neumann boundary contribution in (8.2.7) vanishes for *all* test functions and we can again write it off. We are left just with the integral over Cauchy boundary Γ_C. We need to use Fourier's lemma for the last time to conclude that,

$$a_{ij}\frac{\partial u}{\partial x_j}n_i+\beta u-g=0 \text{ on } \Gamma_C\,.$$

We have recovered the Cauchy BC, and we are done.

As in 1D, in our proof of equivalence we have assumed that the variational solution is sufficiently regular so we can integrate back by parts and introduce the second derivatives.

We say again that the *two formulations are equivalent up to the regularity of the solution*.

Interface problem. As in 1D, the interface problem admits *exactly the same* variational formulation. We proceed in exactly the same way as in 1D. We consider a globally continuous test function $v(\boldsymbol{x})$, multiply the differential equation pointwise with $v(\boldsymbol{x})$, and integrate over each of the subdomains Ω_I. Next, we integrate by parts over the two subdomains and sum up the final results. On top of the previous terms, we get an additional term corresponding to the interface,

$$\int_\Omega\left(a_{ij}\frac{\partial u}{\partial x_j}\frac{\partial v}{\partial x_i}+b_j\frac{\partial u}{\partial x_j}v+cuv\right)\,d\boldsymbol{x}-\int_\Gamma a_{ij}\frac{\partial u}{\partial x_j}n_i v\,ds+\int_{\Gamma_{12}}[a_{ij}\frac{\partial u}{\partial x_j}n_i]v\,ds=\int_\Omega fv\,d\boldsymbol{x}\,,$$

where $[\cdot]$ denotes the jump of the flux. But, according to our second interface condition, the flux is continuous, i.e. the interface term vanishes. We conclude then the derivation of the weak formulation in the same way as for the original problem.

Conversely, when showing that the variational solution satisfies the classical formulation, we integrate by parts only over the subdomains, and end up with the additional interface term again,

$$\int_\Omega \left\{ -\frac{\partial}{\partial x_i}\left(a_{ij}\frac{\partial u}{\partial x_j}\right) + b_j\frac{\partial u}{\partial x_j} + cu - f \right\} v \, d\boldsymbol{x} +$$
$$\int_{\Gamma_{12}} \left[a_{ij}\frac{\partial u}{\partial x_j} n_i \right] v \, ds + \int_{\Gamma_N} \left\{ a_{ij}\frac{\partial u}{\partial x_j} n_i - g \right\} v \, ds + \int_{\Gamma_C} \left\{ a_{ij}\frac{\partial u}{\partial x_j} n_i + \beta u - g \right\} v \, ds = 0 \, .$$

We then add an extra step to our 'recovering process' based on the Fourier's lemma, to recover the extra interface condition representing the continuity of the flux.

The whole procedure may be generalized to any number of material interfaces. The variational formulation covers them all, and remains in exactly the same form.

8.3 Algebraic structure of the variational formulation

The beauty of the abstract variational formulation used throughout the first part of the book, is that it remains exactly the same for the 2D problems.

The test functions form a *(vector) space of test functions*,

$$V = \{v(\boldsymbol{x}) \ : \ v = 0 \text{ on } \Gamma_D\} \tag{8.3.8}$$

To be fully precise, we must specify the regularity of the test functions yet, to be discussed in the next chapter.

The right hand side of the VBVP is identified as a *linear functional* $l(v)$ of test function $v \in V$. Similarly, the left-hand side of VBVP is identified as a *bilinear functional* $b(u, v)$ of solution u and test function v. This means that, with fixed u, the left-hand side is linear in v and, with fixed v, is linear in u.

Next we introduce again the notion of the *lift*. A function $\tilde{u}_D$ is a *lift of Dirichlet data* u_D, if the restriction of $\tilde{u}_D$ to Dirichlet boundary Γ_D coincides with data u_D. In other words, the lift is simply an extension of u_D to the whole domain. The tricky detail comes again with regularity issues and the existence of extensions (lifts) that are of *finite energy*. For a moment, we shall assume that this is always possible. Once we have introduced lift $\tilde{u}_D$, we can subtract it from solution u and the difference will vanish on Γ_D, i.e. it will belong to the space of test functions. In other words, the solution must be the sum of the lift and a test function. The collection of such sums is identified again as the algebraic sum of lift $\tilde{u}_D$ and space V, and called an *affine space*,

$$\tilde{u}_D + V = \{\tilde{u}_D + v \ : v \in V\}$$

The final algebraic structure of the variational formulation is expressed in the form of the same *abstract variational formulation* as for the 1D problem,

$$\begin{cases} u \in \tilde{u}_D + V \\ b(v,u) = l(v), \quad \forall v \in V. \end{cases} \tag{8.3.9}$$

In the particular case of *homogeneous Dirichlet BC's* i.e., when $u_D = 0$, we can use the zero function as a lift, and the abstract VBVP reduces to the simpler form,

$$\begin{cases} u \in V \\ b(v,u) = l(v), \quad \forall v \in V. \end{cases}$$

Finally, we recall the alternative formulation of the case of non-homogeneous Dirichlet BC's. Once we have found a particular function $\tilde{u}_D$ that satisfies the non-homogeneous Dirichlet data, we make the substitution $u = \tilde{u}_D + w$ where $w \in V$ satisfies the homogeneous Dirichlet boundary conditions, and set to determine the perturbation w. The corresponding abstract formulation is then as follows.

$$\begin{cases} w \in V \\ b(w,v) = l(v) - b(v,\tilde{u}_D), \quad \forall v \in V\ . \end{cases} \tag{8.3.10}$$

The solution of the non-homogeneous case reduces then to the homogeneous one, provided we can find the lift and modify next the right-hand side according to the formula above. With $\tilde{u}_D$ fixed, the right-hand side of (8.3.10) defines the *modified* linear form,

$$l^{mod}(v) = l(v) - b(v,\tilde{u}_D)\ . \tag{8.3.11}$$

8.4 Equivalence with a minimization problem

Recall that the variational problem is equivalent with the minimization problem if the bilinear form is symmetric and positive-definite.

How do the abstract conditions on symmetry and positive-definitness of form $b(u,v)$ translate into concrete assumptions on the material data ?

The condition on symmetry is easy to answer. We have already assumed that matrix a_{ij} is symmetric. This makes the leading term corresponding to second derivatives always symmetric. So is the zero-th order term. The first order term cannot be symmetric unless it simply vanishes. Therefore, as in 1D, the symmetry is equivalent with the assumption that vector b_j vanishes.

The discussion of sufficient conditions for positiveness of the bilinear form follows the same lines as in 1D. We begin by writing down the formula for $b(u,u)$ in the symmetric case,

$$b(u,u) = \int_\Omega a_{ij} \frac{\partial u}{\partial x_j} \frac{\partial u}{\partial x_i} + cu^2 \, d\boldsymbol{x} + \int_{\Gamma_C} \beta u^2 \, ds\ .$$

The ellipticity condition (8.1.2) implies that the first term is always non-negative. It is natural[2] to assume then that the other two terms should also be non-negative. This leads to the assumption that coefficients $c(\boldsymbol{x})$ and $\beta(\boldsymbol{x})$ are non-negative,

$$c(\boldsymbol{x}) \geq 0,\ \boldsymbol{x} \in \Omega, \quad \beta(\boldsymbol{x}) \geq 0,\ \boldsymbol{x} \in \Gamma_C\ .$$

With these assumptions in place, $b(u,u) = 0$ implies that all three terms must vanish. Vanishing of the first term, and the ellipticity condition imply that the first derivatives must identically be zero, i.e. u can only be a constant function, $u = C = const$. Similarly as in 1D, there are three ways that we can force this constant to be zero.

- We can assume that $c(\boldsymbol{x})$ is positive over at least a subdomain with non-zero measure (area).
$$\int_\Omega c(\boldsymbol{x}) u^2\, dx = C^2 \int_\Omega c(\boldsymbol{x}) = 0\ ,$$
implies then that $C = 0$.
- We can assume that $\beta(\boldsymbol{x})$ is positive over at least a segment of Cauchy boundary Γ_C of non-zero measure (length), and use the same reasoning as above.
- We remember that we need the positive definitness over the space of test functions only, and make sure that the Dirichlet part of the boundary is of non-zero measure (length), $\mathrm{meas}\,\Gamma_D > 0$.

We mention at this point only that the mathematics of variational boundary-value problems does not allow for imposing the Dirichlet BC at a single point, a finite number of points or, in general, on a subset of measure zero.

As in the 1D case, the case of pure Neumann BC and $c = 0$ is tricky. We shall discuss it separately later on.

8.5 Examples from Mechanics and Physics

When $a_{ij} = \delta_{ij}$ (Kronecker's delta), $b_j = c = 0$, the differential operator reduces to probably the most celebrated case of the Laplace operator. If $f = 0$ we speak of *Laplace equation*, for a non-homogeneous right-hand side, the name of *Poisson equation* is used. The corresponding boundary differential operator reduces then to the normal derivative.

The list of problems that reduce to the solution of a single elliptic equation is very long. We shall list only a few representative examples.

Exercises

[2] But not necessary

Exercise 8.1 Prove the elementary integration by parts formula (8.2.4) for a class of *polygonal domains*. A domain is called polygonal, if its boundary can be split into n curvilinear segments,

$$\Gamma = \Gamma_1 \cup \ldots \cup \Gamma_n \,,$$

where each of the segments Γ_i can be parameterized with a C^1 function,

$$\boldsymbol{x}_i = \boldsymbol{x}_i(\xi), \quad \xi \in [a_i, b_i] \,.$$

Assume for simplicity that each of the segments can be parameterized with $\xi = x_1$ or ξ_2. Follow the following steps.

Step 1: Consider first the case of a special domain,

$$\Omega = \{(x_1, x_2) \,:\, \phi_1(x_2) < x_1 < \phi_2(x_2),\ x_2 \in (a, b)\} \,.$$

Use the 1D integration formula and the definition of the line integral,

$$\int_{\widehat{AB}} f(\boldsymbol{x})\, ds = \int_a^b f(\boldsymbol{x}(\xi)\, \underbrace{|\frac{d\boldsymbol{x}}{d\xi}|\, d\xi}_{ds},$$

where $\boldsymbol{x} = \boldsymbol{x}(\xi),\ \xi \in [a, b]$ is *a parameterization* of curve $\widehat{AB}$, to verify the formula for this case.

Step 2: Consider an analogous domain with roles of x_1 and x_2 interchanged.

Step 3: Argue that every polygonal domain can be split into a finite number of subdomains that fall in the categories discussed in the first two steps. Apply then the formula for each of the subdomains and sum up over the subdomains.

Chapter 9

Sobolev Spaces

The assumption of *finite energy solutions* leads to the definition of the Sobolev space. In this chapter we continue to discuss the main facts about the Sobolev spaces in the multidimensional context. The theory becomes much more technical than in 1D, and any precise study of the subject would turn this monograph into a text on Sobolev spaces. We shall restrict ourselves thus only to reciting the main results on Sobolev spaces, and examples illustrating the main points.

9.1 Sobolev space $H^1(\Omega)$

Distributional derivatives. As in 1D, the space of infinitely differentiable functions on Ω with supports in Ω, denoted $C_0^\infty(\Omega)$, is the *space of test functions*, and it provides a starting point for the *theory of distributions*. A function w_i is called the *distributional derivative* of function u with respect to (wrt) x_i, if,

$$\int_\Omega u \frac{\partial \phi}{\partial x_i} = -\int_\Omega w_i \phi \quad \forall \phi \in C_0^\infty(\Omega)\,.$$

For classical derivatives, the definition above reduces to the elementary integration by parts formula discussed earlier. The boundary term vanishes because the (distribution theory) test functions vanish (with all derivatives) on the boundary of the domain.

Typically we assume that both function u and its distributional derivative are L^p functions with the same exponent p.

In order to understand the difference between the classical and distributional derivatives, partition domain Ω into subdomains Ω_I with boundaries $\partial\Omega_I$, and consider a function $u(\boldsymbol{x})$ that reduces to a C^1 branch u_I in each of the subdomains, see Fig. 9.1 for illustration.

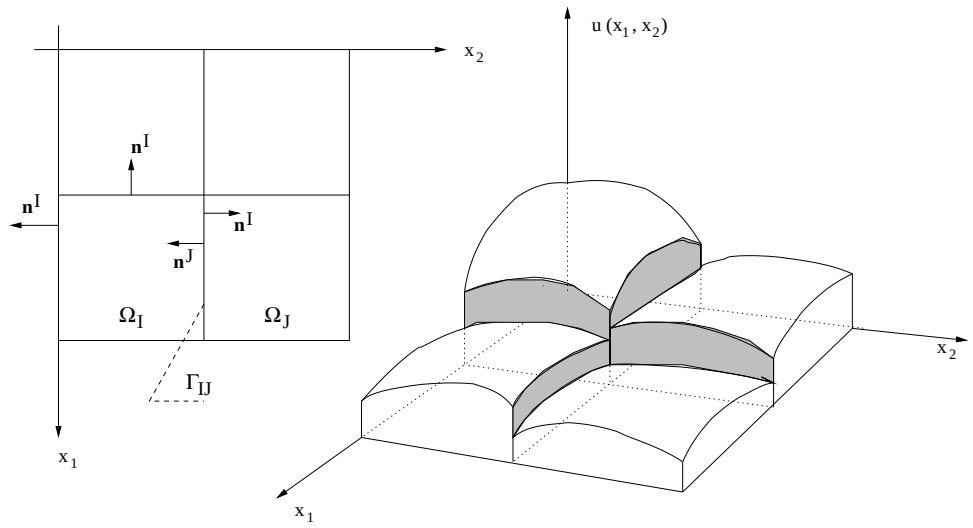


Figure 9.1: A piecewise C^1 function supported on domain Ω partitioned into subdomains Ω_I

Let $v \in C_0^\infty(\Omega)$. Then,

$$\begin{aligned}\int_\Omega u\frac{\partial\phi}{\partial x_i} &= \sum_I \int_{\Omega_I} u_I \frac{\partial\phi}{\partial x_i} \\ &= \sum_I \left\{ -\int_{\Omega_I} \frac{\partial u_I}{\partial x_i}\phi + \int_{\partial\Omega_I} u\phi n_i^I \right\} ,\end{aligned}$$

where $\boldsymbol{n}_I$ is the outward normal unit to $\partial\Omega_I$. We introduce now a function w_i that coincides with derivatives $\frac{\partial u_I}{\partial x_i}$ in subdomains and takes arbitrary values on $\partial\Omega_I$ (Lebesgue integral is insensitive to the behavior of functions on subsets of measure zero). Let Γ_{IJ} denote the interface between two subdomains Ω_I and Ω_J. Reordering boundary terms and using the fact that the test functions vanish on Γ, we obtain,

$$\int_\Omega u\frac{\partial\phi}{\partial x_i} = -\int_\Omega w_i\phi + \sum_{I,J}\int_{\Gamma_{IJ}} (u_I n_{I,i} + u_J n_{J,i}) ,$$

where the summation extends over all non-empty interfaces Γ_{IJ}. As,

$$\boldsymbol{n}_I = -\boldsymbol{n}_J \text{ on } \Gamma_{IJ} ,$$

the interface contributions will vanish if function u is *globally continuous*. [1] Thus, as long as function u is globally continuous, we can differentiate it locally over each subdomain, and then glue the resulting derivatives into a global function to obtain the distributional derivative. Obviously, function u, in general, will not be differentiable in the classical sense at the interfaces.

[1] Otherwise we get *Dirac's delta's* contributions, and the distributional derivative is not *regular*.

Finite energy solutions. Sobolev space of order one For problems admitting the minimization principle, it is natural to assume that we shall restrict ourselves only to a class of functions that produce a finite energy,

$$X = \{u \,:\, \int_\Omega a_{ij}\frac{\partial u}{\partial x_j}\frac{\partial u}{\partial x_i} + cu^2\, d\boldsymbol{x} + \int_{\Gamma_C} \beta u^2\, ds < \infty\} \tag{9.1.1}$$

As in 1D, it is convenient to introduce a functional space that would serve more then one set of material data. Assuming that the material data are bounded, this leads to the definition of the *Sobolev space of order one*

$$H^1(\Omega) = \{u \,:\, \int_\Omega \left(|\boldsymbol{\nabla} u|^2 + |u|^2\right)\, d\boldsymbol{x} < \infty\}\,,$$

with the corresponding inner product,

$$(u,v)_{H^1(\Omega)} = \int_\Omega \{\sum_{j=1}^{2} \frac{\partial u}{\partial x_j}\frac{\partial v}{\partial x_j} + uv\}\, d\boldsymbol{x}\,,$$

and the norm,

$$\|u\|_{H^1(\Omega)} = \|u\|_{1,\Omega} = (u,u)^{\frac{1}{2}}_{H^1(\Omega)} = \left(\int_\Omega \left(|\boldsymbol{\nabla} u|^2 + |u|^2\right)\, d\boldsymbol{x}\right)^{\frac{1}{2}}\,.$$

We shall also use the corresponding first order seminorm,

$$|u|_{H^1(\Omega)} = |u|_{1,\Omega} = \left(\int_\Omega \left(|\boldsymbol{\nabla} u|^2\right)\, d\boldsymbol{x}\right)^{\frac{1}{2}}\,,$$

and the L^2-norm,

$$\|u\|_{L^2(\Omega)} = \|u\|_{0,\Omega} = \left(\int_\Omega |u|^2\, d\boldsymbol{x}\right)^{\frac{1}{2}}\,.$$

9.2 *Sobolev spaces of an arbitrary order*

Once we have introduced the Sobolev space $H^1(\Omega)$ to identify finite energy solutions, it is natural to use the same idea to measure (higher) regularity of the solution using the same concept. We begin with Sobolev spaces $H^n(\Omega)$ of natural order $n = 2, 3, \ldots$.

$$H^n(\Omega) = \{u \in L^2(\Omega) \,:\, D^{\boldsymbol{\alpha}} u \in L^2(\Omega),\ |\boldsymbol{\alpha}| \le n\}$$

We use the multi-index notation for the higher order derivatives,

$$\boldsymbol{\alpha} = (\alpha_1, \alpha_2), \alpha_i \in I\!N,\ i = 1,2, \quad |\boldsymbol{\alpha}| = \alpha_1 + \alpha_2$$

$$D^{\boldsymbol{\alpha}} u = \frac{\partial^{|\boldsymbol{\alpha}|} u}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2}}\,.$$

The Sobolev spaces are Hilbert spaces with the inner product,

$$(u,v)_{H^n(\Omega)} = \int_\Omega \sum_{|\boldsymbol{\alpha}|\le n} D^{\boldsymbol{\alpha}} u D^{\boldsymbol{\alpha}} v \, d\boldsymbol{x} \,,$$

and the corresponding norm,

$$\|u\|_{H^n(\Omega)} = \|u\|_{n,\Omega} = (u,u)^{\frac{1}{2}}_{H^n(\Omega)} \,.$$

We shall also use the seminorm of order n whose definition involves only the highest order derivatives,

$$|u|_{H^n(\Omega)} = |u|_{n,\Omega} = \left(\int_\Omega \sum_{|\boldsymbol{\alpha}|=n} |D^{\boldsymbol{\alpha}} u(\boldsymbol{x})|^2 \, d\boldsymbol{x} \right)^{\frac{1}{2}} \,.$$

Next we extend the notion of Sobolev spaces to arbitrary positive real exponents $s \in \mathbb{R}, s \ge 0$. Things become more technical. There is more than one way to do it, and various constructions may result in different definitions that are not necessary equivalent to each other. Regularity of the boundary of the domain enters the picture as well. One of the most popular constructions is obtained by using the *real interpolation method* [113]. Given a parameter $s = n + \theta, 0 < \theta < 1$, we define an intermediate space $H^s(\Omega)$,

$$H^{n+1}(\Omega) \subset H^s(\Omega) \subset H^n(\Omega) \,.$$

The most profound result of the interpolation method is the possibility of extending results concerning continuity of operators defined of Sobolev spaces. Once we have an operator A defined on space $H^n(\Omega)$ with values in some normed space Y, which admits a continuity estimate,

$$\|Au\|_Y \le C_n \|u\|_{H^n(\Omega)} \,,$$

with a sharper estimate for functions coming from the smaller, higher order space $H^{n+1}(\Omega)$,

$$\|Au\|_Y \le C_{n+1} \|u\|_{H^{n+1}(\Omega)} \,,$$

we can immediately derive an estimate for the fractional space,

$$\|Au\|_Y \le C_n^\theta C_{n+1}^{1-\theta} \|u\|_{H^s(\Omega)} \,.$$

Given spaces of arbitrary order $H^s(\Omega),\ s = n + \theta, 0 < \theta < 1$, we consider corresponding subspaces $H^s_0(\Omega)$ of functions that vanish (along with all derivatives of order $\le n$) on the boundary of the domain. These are formally defined by taking the closure of the space of test functions in the H^s-norm,

$$H^s_0(\Omega) = \overline{C^\infty_0(\Omega)}^{H^s(\Omega)} \,.$$

Sobolev spaces of negative order $H^{-s}(\Omega)$ are then defined as topological duals of spaces $H^s_0(\Omega)$,

$$H^{-s}(\Omega) = (H^s_0(\Omega))' \,.$$

Elements of Sobolev spaces with negative exponent are thus functionals that 'do not see the boundary values'.

9.3 *Trace Theorem*

9.4 *Embedding Theorems*

9.5 *Well posedness of the variational BVP*

Chapter 10

2D hp Finite Element Method on Regular Meshes

We discuss the fundamentals of hp finite elements in two space dimensions. We begin by partitioning domain Ω into finite elements. The 2D elements have shapes of triangles or quadrilaterals (quads), possibly with curvilinear edges. In this chapter, we shall focus on *regular meshes*. A mesh, consisting possibly of both triangles and quads is called *regular*, if the intersection of any two elements in the mesh is either empty, reduces to a single vertex, or consists of a *whole* common edge shared by the elements. The case of *irregular meshes* arising naturally from h-refinements, will be discussed in the next chapters.

Generation of FE meshes in 2D is much more technical then for the 1D model problem. Each finite element must be parameterized with an element map, mapping a corresponding master triangular or square element onto the physical element, and those maps must be compatible with each other on inter-element boundaries. Except for simple polygonal domains, mesh generation involves an approximation of the actual domain. The computational domain Ω_h, in general, does not coincide with the actual domain Ω in which the BVP has been formulated. Mesh generation for higher order elements differs considerably from standard techniques used for low order elements, and it must be based on a precise *modeling of geometry*. We shall present a version of such a modeling based on a *Geometry Modeling Package* (GMP) developed in the course of this project.

We shall begin with the construction of two master elements: *master square*, and *master triangle* of variable order. Next we discuss the idea of a *parametric element*, and the construction of globally defined basis functions, obtained by 'gluing' element shape functions. We then review quickly the fundamental elements of the FE technique: computation of element matrices, assembling of global matrices, incorporation of Dirichlet BC's, and a graphical postprocessing. Finally, we generalize to two dimensions the projection-based interpolation.

The geometry modeling and mesh generation will be discussed in a subsequent chapter.

10.1 Quadrilateral master element

The element occupies the standard reference square, $\hat{K} = [0,1]^2$. The element space of shape functions $X(\hat{K})$ is a subspace of $Q^{(p_h,p_v)}$, i.e. polynomials that are of order p_h in ξ_1, and order p_v with respect ξ_2. In order to be able to match in one mesh elements of different orders, we associate with each of the element edges a possibly different order of approximation $p_i, i = 1,\ldots,4$, with the assumption that,

$$p_1, p_3 \le p_h \text{ and } p_2, p_4 \le p_v.$$

The element space of shape functions is now identified as the subspace of polynomial space $Q^{(p_h,p_v)}$, consisting of functions whose restrictions to edge $\hat{e}_i$ reduce to polynomials of (smaller or equal) degree p_i,

$$X(\hat{K}) = \{\hat{u} \in Q^{(p_h,p_v)} \ : \ \hat{u}|_{\hat{e}_i} \in \mathcal{P}^{p_i}(\hat{e}_i)\} \ .$$

We shall use also an abbreviated notation $Q^p_{p_e}$, where $p = (p_h, p_v)$ is the (vector-valued) order of the element, and p_e refers to order of approximation for a generic edge e of the element.

The element shape functions are constructed as *tensor products* of the 1D shape functions. It is convenient to group them into subsets associated with the element vertices, edges and the element interior. Shape functions belonging to the same group have the same *nodal connectivity*, and the corresponding basis functions are built using the same logic. For instance, all basis functions associated with an inter-element edge "consist of" two shape functions corresponding to the two adjacent elements and the common edge, whereas basis functions corresponding to an interior of an element "consist" of just one contributing shape function.

To facilitate the communication, we introduce again the notion of abstract *nodes*, see Fig. 10.1.

- four vertex nodes: $\hat{\boldsymbol{a}}_j, j = 1,\ldots,4$,
- four mid-edge nodes: $\hat{\boldsymbol{a}}_j, j = 5,\ldots,8$,
- middle node: $\hat{\boldsymbol{a}}_9$.

With each of the nodes, we associate the corresponding order of approximation, $p = 1$ for the vertex nodes, the edge order p_i for the i-th mid-edge node, and the *anisotropic* order (p_h, p_v) for the middle node. The corresponding shape functions are now defined as tensor products of 1D shape functions $\hat{\chi}_i(\xi)$,

- one bilinear shape function for each of the vertex nodes,

$$\begin{aligned}
\hat{\phi}_1(\xi_1,\xi_2) &= \hat{\chi}_1(\xi_1)\hat{\chi}_1(\xi_2) &= (1-\xi_1)(1-\xi_2) \\
\hat{\phi}_2(\xi_1,\xi_2) &= \hat{\chi}_2(\xi_1)\hat{\chi}_1(\xi_2) &= \xi_1(1-\xi_2) \\
\hat{\phi}_3(\xi_1,\xi_2) &= \hat{\chi}_2(\xi_1)\hat{\chi}_2(\xi_2) &= \xi_1\xi_2 \\
\hat{\phi}_4(\xi_1,\xi_2) &= \hat{\chi}_1(\xi_1)\hat{\chi}_2(\xi_2) &= (1-\xi_1)\xi_2 \ ,
\end{aligned}$$

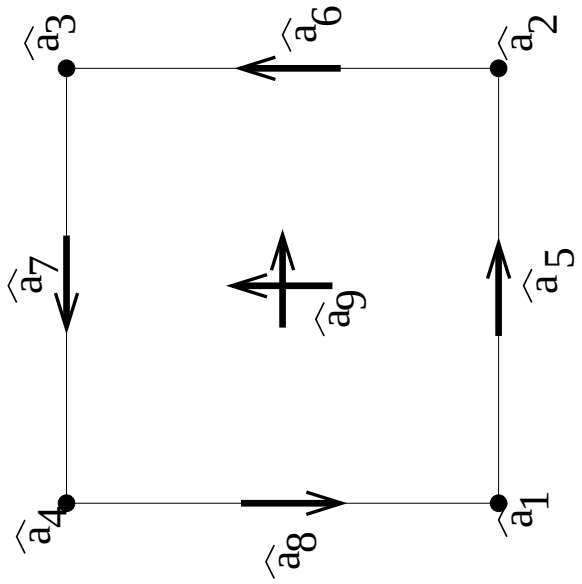


Figure 10.1: Quadrilateral master element

- $p_i - 1$ shape functions for each of the mid-edge nodes,

$$\begin{aligned}
\hat{\phi}_{5,j}(\xi_1,\xi_2) &= \hat{\chi}_{2+j}(\xi_1)\hat{\chi}_1(\xi_2) \qquad j = 1,\ldots,p_1-1 \\
\hat{\phi}_{6,j}(\xi_1,\xi_2) &= \hat{\chi}_2(\xi_1)\hat{\chi}_{2+j}(\xi_2) \qquad j = 1,\ldots,p_2-1 \\
\hat{\phi}_{7,j}(\xi_1,\xi_2) &= \hat{\chi}_{2+j}(1-\xi_1)\hat{\chi}_2(\xi_2) \quad j = 1,\ldots,p_3-1 \\
\hat{\phi}_{8,j}(\xi_1,\xi_2) &= \hat{\chi}_1(\xi_1)\hat{\chi}_{2+j}(1-\xi_2) \quad j = 1,\ldots,p_4-1\,,
\end{aligned}$$

- $(p-1)(q-1)$ *bubble shape functions* for the middle node,

$$\hat{\phi}_{9,ij}(\xi_1,\xi_2) = \hat{\chi}_{2+i}(\xi_1)\hat{\chi}_{2+j}(\xi_2) \qquad i = 1,\ldots,p_h-1,\ j = 1,\ldots,p_v-1\,.$$

Please note the difference between the first and third, and the second and fourth edges. When restricted to an element edge, the 2D shape functions reduce to 1D shape functions, with the local coordinate ξ oriented counterclockwise. More precisely, the parametrization for the four element edges is:

$$\begin{aligned}
&\xi_1 = \xi && \xi_2 = 0 \\
&\xi_1 = 1 && \xi_2 = \xi \\
&\xi_1 = 1-\xi && \xi_2 = 1 \\
&\xi_1 = 0 && \xi_2 = 1-\xi\,,
\end{aligned} \tag{10.1.1}$$

with $\xi \in [0,1]$. We emphasize the abstract character of the "nodes" introduced above. They are merely an abstraction for the element vertices, edges, and its interior, and should not be confused with the classical notion of the Lagrange or Hermite nodes.

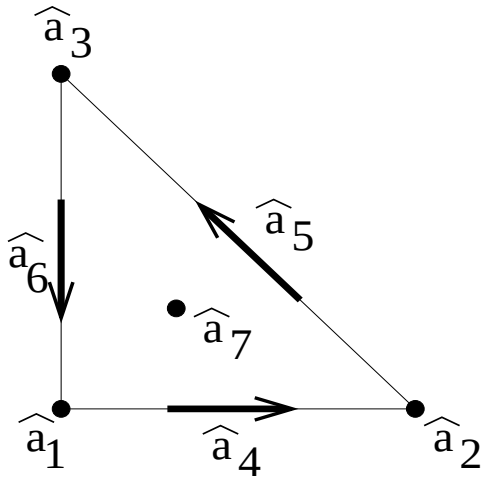


Figure 10.2: Triangular master element

Finally, notice that the quadrilateral element shape functions have been defined exclusively in terms of 1D shape functions. Choosing Peano or integrated Legendre polynomials for the 1D shape functions, we get two different sets of shape functions for the square element.

10.2 Triangular master element

The element occupies the standard unit right triangle illustrated in Fig. 10.2. It has seven nodes: three vertex nodes $\hat{\boldsymbol{a}}_i, i = 1, 2, 3$, three mid-edge nodes $\hat{\boldsymbol{a}}_i, i = 4, 5, 6$, and middle node $\hat{\boldsymbol{a}}_7$. The element space of shape functions consists of polynomials of order p whose restrictions to element edges $\hat{e}_i$ reduce to polynomials of order p_i,

$$X(\hat{K}) = \{\hat{u} \in \mathcal{P}^p(\hat{K}) \,:\, \hat{u}|_{\hat{e}_i} \in \mathcal{P}^{p_i}(\hat{e}_i), \quad i = 1, 2, 3\} \,,$$

where, similarly to the quad element, we assume that

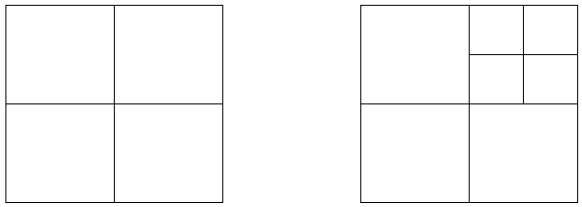
$$p_1, p_2, p_3 \leq p.$$

We shall also use the notation $\mathcal{P}^p_{p_e}$ with p_e standing for order of approximation for generic edge e. The element shape functions are conveniently defined using area (affine, barycentric) coordinates:

$$\lambda_1 = 1 - \xi_1 - \xi_2, \; \lambda_2 = \xi_1, \; \lambda_3 = \xi_2.$$

For each of the three vertex nodes, we introduce a linear shape function equal to the corresponding affine coordinate,

$$\hat{\phi}_i = \lambda_i, \quad i = 1, 2, 3 \,.$$



(a) (b)

Figure 10.3: Extension of the 1D edge kernel functions to the triangle

The linear shape function associated with i-th vertex equals one at the vertex, and it vanishes at the remaining vertices.

For each of the three mid-edge node $\hat{\boldsymbol{a}}_{3+i}$, we introduce $p_i - 1$ shape functions that reduce along the edge to 1D bubble functions, and vanish along the remaining two edges. This implies that the formula for the i-th node shape functions must include factor $\lambda_i \lambda_{i+1}$[1] Introducing *kernel functions* for the 1D shape functions,

$$\hat{\chi}_j^{ker}(\xi) = \frac{\hat{\chi}_{2+j}(\xi)}{\xi(1-\xi)}, \quad j = 1, \ldots, p-1 \,,$$

we represent the 1D bubble shape functions in the form,

$$\hat{\chi}_{2+j}(\xi) = \xi(1-\xi)\hat{\chi}_j^{ker}(\xi) \,.$$

The linear terms represent the restriction of factors λ_i and λ_{i+1} to the edge. The task of constructing the edge shape functions will thus be completed if we construct an extension of the 1D kernel functions to the whole element. The simplest extension is constructed by considering straight lines parallel to the line connecting the mid-point of the edge with the opposite vertex,

$$\lambda_{i+1} - \lambda_i = const \,,$$

and requesting the extension to be constant along those lines. The idea reflects a similar behavior of the edge shape functions for the quadrilateral element.

$$\hat{\phi}_{3+i,j} = \lambda_i \lambda_{i+1} \hat{\chi}_j^{ker}((\lambda_{i+1} - \lambda_i + 1)/2), \quad i = 1, 2, 3, \quad j = 1, \ldots, p_i - 1 \,.$$

The construction is illustrated in Fig. 10.3 for an arbitrary triangle using the concept of affine coordinates. Given a point with affine coordinates $\lambda_1, \lambda_2, \lambda_3$, we project the point in the direction of the bisecting line along the i-th edge, and determine the corresponding edge coordinate $\xi = (\lambda_{i+1} - \lambda_i + 1)/2$ of the projection, comp. Exercise 10.2. We then use the value of the kernel shape function at that point to define the extension.

[1] Mid-edge nodes and vertex nodes are numerated modulo 3, i.e., for $i = 3$, $i + 1 = 1$.

Finally, we need to introduce shape functions for the middle node. The shape functions span the family of polynomials of order p vanishing on the boundary of the triangle. Such functions must be of the form,

$$\lambda_1\lambda_2\lambda_3\ \phi(\xi_1,\xi_2)\ ,$$

where ϕ is an arbitrary polynomial of order $p-3$. The dimension of the space of the triangle bubble shape functions is thus $(p-2)(p-1)/2$. The simplest choice might be,

$$\begin{aligned}
&\lambda_1\lambda_2\lambda_3\\
&\lambda_1\lambda_2^2\lambda_3,\ \lambda_1\lambda_2\lambda_3^2\\
&\dots\\
&\lambda_1\lambda_2^{p-2}\lambda_3,\ \lambda_1\lambda_2^{p-3}\lambda_3^2,\dots,\ \lambda_1\lambda_2\lambda_3^{p-2}\ ,
\end{aligned}$$

(recall that $\lambda_2=\xi_1, \lambda_3=\xi_2$). In an effort to define the triangle shape functions exclusively in terms of 1D kernel functions, we shall modify the fomulas above to a slightly different form,

$$\hat{\phi}_{7,n,j}=\lambda_1\lambda_2\lambda_3\hat{\chi}_j^{kern}(\lambda_2)\hat{\chi}_{n-1-j}^{kern}(\lambda_3),\quad j=1,\dots,n-2,\ n=3,\dots,p\ .$$

The shape functions have been organized here hierarchically with respect to order $n=3,\dots,p$, with the second index j denumerating the shape functions of the same order.

Notice that the choice of 1D kernel shape functions will automatically induce the corresponding set of shape functions for the triangular element. The kernel functions corresponding to the Peano and integrated Legendre polynomials are represented by the formulas,

$$\begin{aligned}
&\hat{\chi}_j=(2\xi-1)^{j-1}, && j=1,\dots,p-1\\
&\hat{\chi}_j=\tfrac{(-1)}{j(j+1)}\tfrac{dl_j}{d\xi}, && j=1,\dots,p-1\ ,
\end{aligned}$$

(see the discussion of 1D shape functions in Part 1).

When restricted to element edges, the edge shape functions reduce to 1D shape functions, with the counterclockwise orientation of the element edges. The corresponding parametrizations for the element edges are,

$$\begin{aligned}
&\xi_1=\xi && \xi_2=0\\
&\xi_1=1-\xi && \xi_2=\xi\\
&\xi_1=0 && \xi_2=1-\xi\ ,
\end{aligned}\tag{10.2.2}$$

with $\xi\in[0,1]$.

10.3 Parametric element

We use the same procedure as in 1D to define parametric (deformed) quads and triangles. Given a bijective map,

$$\boldsymbol{x}_K:\hat{K}\to K\ ,$$

from either master quad or triangle onto an element K, we define the element space of shape functions as the collection of compositions of inverse $\boldsymbol{x}_K^{-1}$ and the master element shape functions,

$$X(K) = \{u = \hat{u} \circ \boldsymbol{x}_K^{-1} \ : \ \hat{u} \in X(\hat{K})\}.$$

Accordingly, the element shape functions are defined as,

$$\phi_i(\boldsymbol{x}) = \hat{\phi}_i(\boldsymbol{\xi}), \text{ where } \boldsymbol{x}_K(\boldsymbol{\xi}) = \boldsymbol{x}.$$

For each element K, we shall speak about its vertex, mid-edge, and middle nodes, understood again simply as an abstraction for the element vertices, edges and interior. The parametric transformation maps master element $\hat{K}$ nodes into nodes of element K.

We restrict ourselves to the *isoparametric* deformations only, i.e. we assume that the map $\boldsymbol{x}_K$ lives in the corresponding space of shape functions for the master element. More precisely,

$$\boldsymbol{x}_K(\boldsymbol{\xi}) = \sum_j \boldsymbol{x}_{Kj} \hat{\phi}_j(\boldsymbol{\xi}) \, .$$

Here j is the index denumerating shape functions of the master element, and $\boldsymbol{x}_{Kj}$ denote the corresponding *vector-valued* geometry degrees of freedom, of dimension two for planar problems, or dimension three for meshes in $\mathbb{R}^3$ [2]. Note that only the degrees of freedom corresponding to the vertex nodes have the interpretation of the vertex nodes coordinates. If the map $\boldsymbol{x}_K$ is affine (this implies that all geometry d.o.f. except for the vertex nodes coordinates are zero), the parametric element is called an *affine element*, and the corresponding shape functions are polynomials.

The geometry d.o.f. are generated in the process of mesh generation, discussed in the next chapter.

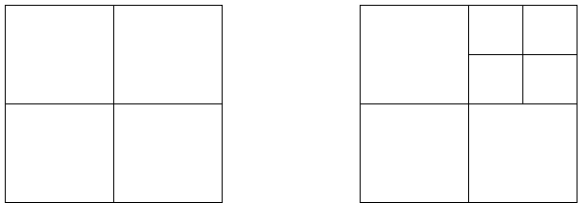
10.4 Finite element space. Construction of basis functions

Recall that in this chapter we discuss the case of regular meshes only. A mesh is *regular* if the intersection of any two elements is either empty, or it consists of a single vertex node or a (whole) common edge. Otherwise the mesh is called *irregular*. Irregular meshes appear naturally as a result of h-refinements discussed in the next chapters. The difference between regular and irregular meshes is illustrated in Fig. 10.4. Domain Ω is partitioned into triangular and quadrilateral elements. In order to facilitate the discussion, we assume for a moment that no approximation of the boundary of the domain is necessary, i.e. that the boundary is at most polynomial, and it can be represented exactly with isoparametric elements of sufficiently high order.

We now define finite element space $X_h = X_{hp}$ as the collection of continuous functions whose restrictions to an element K live in the element space of shape functions,

$$X_h = \{u \ : \ u \text{ is continuous an } u|_K \in X(K), \text{ for every element } K \text{ in the mesh}\} \, .$$

[2] For instance, in Boundary Element Method calculations



(a) (b)

Figure 10.4: Examples of (a) regular and (b) irregular meshes

Global basis functions will be obtained by "gluing together" element shape functions. The procedure is illustrated in Fig. 10.5. The restriction of a basis function to a contributing element K reduces to one of the element shape functions, premultiplied possibly with a *sign factor* $c_{K,k}$,

$$e_i|_K(\boldsymbol{x}) = c_{K,k}\phi_{K,k}(\boldsymbol{x}).$$

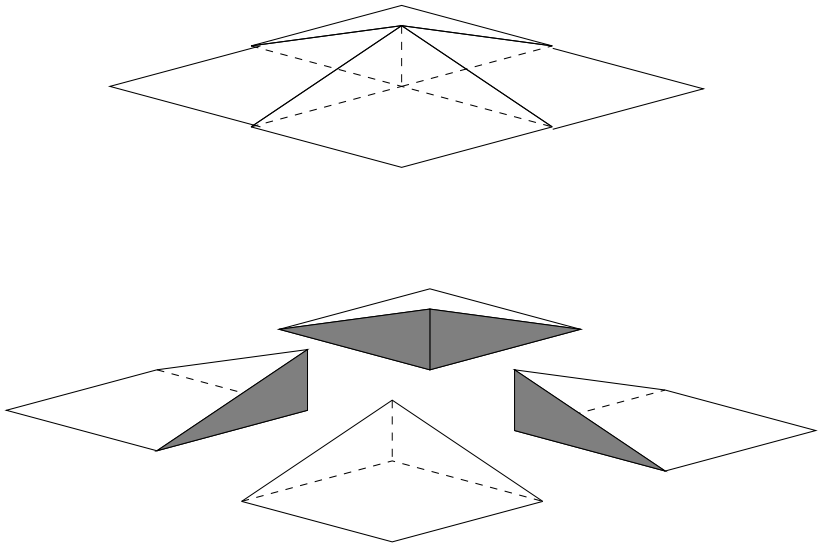
The *d.o.f. connectivity* $i = i(K,k)$ equals the number of the global basis function corresponding to element K and its k-th shape function. The necessity of introducing the sign factor results from the fact that we have decided to parametrize locally element edges *always counterclockwise*. Consequently, the edge shape functions of odd degree corresponding to two neighboring elements *do not match each other*, and one of them has to be premultiplied by the -1 factor.

The sign factor issue is handled by introducing the notion of orientation for the element edges (mid-edge nodes). We assume that each edge comes with its *global orientation*. Now, each of element edges has also its own *local orientation*, determined by the numbering of the nodes (element local system of coordinates), and the counterclockwise rule. The local orientation of the mid-edge nodes has been indicated in Figures 10.1 and 10.2 with arrows. If the local orientation of an element edge is now inconsistent with the global one, *and the shape function is of odd degree*, the corresponding sign factor is -1, otherwise it is equal 1. These sign factors for element edges are part of the connectivity information and, for initial mesh elements, we shall store them as a separate item in the data structure arrays. In the code, we set the global orientation for initial mesh element edges by following the standard rule, and assume that the *global orientation of an edge* follows from the numbering of its vertex endpoints nodes - the edge will always be oriented from the vertex node with a smaller number to the vertex node with a bigger number.

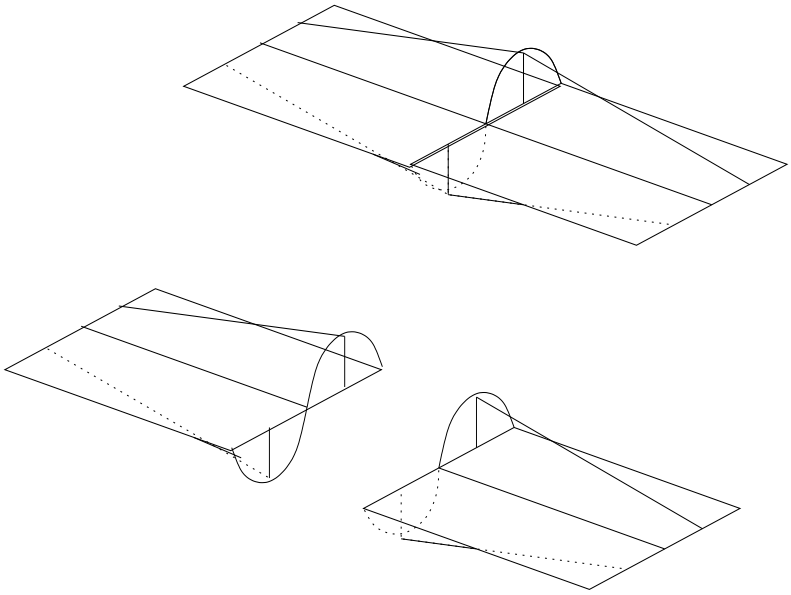
We emphasize again that the information about the sign factors is relevant only for the mid-edge shape functions of odd degree, for all other shape functions the sign factors are always equal to one.

The algorithm for assembling the global stiffness matrix, and the global load vector is now a slight extension of the corresponding 1D algorithm, taking care of the sign factors.

```
initiate global load vector l and global stiffness matrix B with zeros
for each element K in the mesh
```



a)



b)

Figure 10.5: Construction of basis functions on regular meshes for (a) a vertex node, (b) a mid-edge node. Notice the necessity of introducing the sign factor for the mid-edge node basis function.

```
for each element shape function k1
  determine d.o.f. connectivity i = i_{K,k1} and sign factor c_{K,k1}
  l_i = l_i + c_{K,k1} * l_{K,k1}
  for each element shape function k2
    determine connectivity j = j_{K,k2} and sign factor c_{K,k2}
      B_ij = B_ij + c_{K,k1} * c_{K,k2} * b_{K,k1,k2}
  end of the second loop through element K shape functions
 end of the first loop through element shape functions
end of loop through elements
```

As in 1D, the element d.o.f. connectivities are determined through element *nodal connectivities*, and a global *ordering of nodes*. Given an ordering of elements, we define the corresponding *natural ordering of nodes*, which in turn implies the *natural ordering of nodal d.o.f.*. The ordering for initial mesh elements is provided by the initial mesh generator; for refined meshes, we shall use the *natural ordering of elements* implied by the the h-refinement tree structure. All these concepts were already discussed in context of the 1D code and they will remain identical both in 2D and 3D.

10.5 Calculation of element matrices

Similarly as in 1D, we introduce the element bilinear and linear forms,

$$\begin{aligned} b_K(u,v) &= \int_K \left(a_{ij} \frac{\partial u}{\partial x_j} \frac{\partial v}{\partial x_i} + b_j \frac{\partial u}{\partial x_j} v + cuv \right) d\boldsymbol{x} + \int_{\Gamma_C \cap \partial K} \beta uv \, ds \\ l_K(v) &= \int_K fv \, d\boldsymbol{x} + \int_{(\Gamma_N \cup \Gamma_C) \cap \partial K} gv \, ds \, . \end{aligned}$$

We shall use the quad element for discussion and point out differences between the two types of elements as they occur. First of all, we always switch from physical coordinates x_1, x_2 to master coordinates ξ_1, ξ_2. Recall that, for the isoparametric element, the transformation $\boldsymbol{x} = \boldsymbol{x}_K(\boldsymbol{\xi})$ is given by:

$$x_i = \sum_k x_{i,k} \, \hat{\phi}_k(\xi_1, \xi_2) \, , \tag{10.5.3}$$

where the summation extends over all shape functions (d.o.f.) of the element, and $x_{i,k}$ are the element *geometry d.o.f.*. Again, only those corresponding to vertex nodes have the interpretation of actual physical coordinates of the node. The geometry d.o.f. are generated by a mesh generator discussed in Chapter 12, and they are stored in data structure arrays. Notice the difference with the 1D code where only the vertex nodes were assigned the corresponding geometry d.o.f. (coordinates). If we had followed the same strategy in 2D, we would have been able to generate only elements with straight edges. Due to the need for approximating curvilinear geometries, we use all geometry d.o.f.

There is also a need for the geometry d.o.f. for the middle node, this will become clear during the discussion of geometrical modeling an mesh generation. Remember also that the code supports 2D elements on surfaces in 3D - for boundary element calculations. Finally, the concept of an *isoparametric* approximation of geometry is very natural from the programming point of view - we use the same representation for the geometry as for the solution.

Given master element coordinates ξ_1, ξ_2, we can compute not only the values of the corresponding physical coordinates but also their derivatives wrt master coordinates,

$$\frac{\partial x_i}{\partial \xi_j} = \sum_k x_{i,k} \frac{\partial \hat{\phi}_k}{\partial \xi_j}(\xi_1, \xi_2) \; . \tag{10.5.4}$$

We can then calculate the corresponding jacobian of the transformation,

$$|\frac{d\boldsymbol{x}}{d\boldsymbol{\xi}}| = \det(\frac{\partial x_i}{\partial \xi_j}) \; ,$$

and, by inverting the Jacobian matrix, the derivatives of the master element coordinates wrt the physical coordinates. In 2D, the explicit formulas are as follows,

$$\begin{aligned}
\frac{\partial \xi_1}{\partial x_1} &= \frac{\partial x_2}{\partial \xi_2} / |\frac{d\boldsymbol{x}}{d\boldsymbol{\xi}}| \\
\frac{\partial \xi_1}{\partial x_2} &= - \frac{\partial x_1}{\partial \xi_2} / |\frac{d\boldsymbol{x}}{d\boldsymbol{\xi}}| \\
\frac{\partial \xi_1}{\partial x_1} &= - \frac{\partial x_2}{\partial \xi_1} / |\frac{d\boldsymbol{x}}{d\boldsymbol{\xi}}| \\
\frac{\partial \xi_2}{\partial x_2} &= \frac{\partial x_1}{\partial \xi_1} / |\frac{d\boldsymbol{x}}{d\boldsymbol{\xi}}| \; .
\end{aligned}$$

We shall assume that the jacobian is always positive.

The derivatives of the inverse of element map $\boldsymbol{x}_K$ are necessary to evaluate derivatives of the element shape functions wrt physical coordinates, using the chain formula,

$$\frac{\partial \phi}{\partial x_i} = \frac{\partial \hat{\phi}}{\partial \xi_j} \frac{\partial \xi_j}{\partial x_i} \quad \text{(summation in } j \text{ !)} \; .$$

This leads to the following formula for the element integral contributions to the element bilinear form,

$$\int_K \left(a_{ij} \frac{\partial u}{\partial x_j} \frac{\partial v}{\partial x_i} + b_j \frac{\partial u}{\partial x_j} v + cuv \right) d\boldsymbol{x} = \int_{\hat{K}} \left(\bar{a}_{nm} \frac{\partial \hat{u}}{\partial \xi_m} \frac{\partial \hat{v}}{\partial \xi_n} + \bar{b}_m \frac{\partial \hat{u}}{\partial \xi_m} v + \bar{c} uv \right) d\boldsymbol{\xi} \tag{10.5.5}$$

The coefficients defining the differential operator in master element coordinates are defined as follows,

$$\begin{aligned}
\bar{a}_{nm}(\boldsymbol{\xi}) &= a_{ij}(\boldsymbol{x}(\boldsymbol{\xi})) \frac{\partial \xi_j}{\partial x_m} \frac{\partial \xi_i}{\partial x_n} |\frac{d\boldsymbol{x}}{d\boldsymbol{\xi}}| = \hat{a}_{ij}(\boldsymbol{\xi}) \frac{\partial \xi_j}{\partial x_m} \frac{\partial \xi_i}{\partial x_n} |\frac{d\boldsymbol{x}}{d\boldsymbol{\xi}}| \\
\bar{b}_m(\boldsymbol{\xi}) &= b_j(\boldsymbol{x}(\boldsymbol{\xi})) \frac{\partial \xi_j}{\partial x_m} |\frac{d\boldsymbol{x}}{d\boldsymbol{\xi}}| = \hat{b}_j(\boldsymbol{\xi}) \frac{\partial \xi_j}{\partial x_m} |\frac{d\boldsymbol{x}}{d\boldsymbol{\xi}}| \\
\bar{c}(\boldsymbol{\xi}) &= c(\boldsymbol{x}(\boldsymbol{\xi})) |\frac{d\boldsymbol{x}}{d\boldsymbol{\xi}}| = \hat{c}(\boldsymbol{\xi}) |\frac{d\boldsymbol{x}}{d\boldsymbol{\xi}}| \; .
\end{aligned} \tag{10.5.6}$$

By placing 'hats' over functions u and v, we emphasize that they are treated now as functions of master element coordinates ξ.

We have a similar transformation for element integral contribution to the element linear form,

$$\int_K f v \, d\boldsymbol{x} = \int_{\hat{K}} \bar{f} \hat{v} \, d\boldsymbol{\xi} \, , \tag{10.5.7}$$

where,

$$\bar{f}(\boldsymbol{\xi}) = \hat{f}(\boldsymbol{\xi}) |\frac{d\boldsymbol{x}}{d\boldsymbol{\xi}}| \, . \tag{10.5.8}$$

Computation of the boundary terms. In order to compute the boundary integral contributions, we need first to parametrize the boundary, i.e. the element edges. A natural parametrization is provided by the composition of parametrization for master element edges, comp. (10.1.1), (10.2.2), with element map $\boldsymbol{x}_K$,

$$(0,1) \ni \xi \to \boldsymbol{x}_K(\boldsymbol{\xi}(\xi)) \in \mathbb{R}^2 \, .$$

We use again the chain formula to compute the necessary derivatives,

$$\frac{dx_i}{d\xi} = \frac{\partial x_i}{\partial \xi_n} \frac{d\xi_n}{d\xi} \, .$$

The line integral jacobian is defined as,

$$\frac{ds}{d\xi} = \sqrt{\frac{dx_i}{d\xi} \frac{dx_i}{d\xi}} \, .$$

Finally, the boundary integral contributions are computed using the formulas,

$$\begin{aligned} \int_e \beta u v \, ds &= \int_0^1 \bar{\beta} \hat{u} \hat{v} \, d\xi \\ \int_e g v \, ds &= \int_0^1 \bar{g} \hat{v} \, d\xi \, , \end{aligned} \tag{10.5.9}$$

where

$$\bar{\beta}(\xi) = \beta(\boldsymbol{x}(\boldsymbol{\xi}(\xi))) \frac{ds}{d\xi}, \qquad \bar{g}(\xi) = g(\boldsymbol{x}(\boldsymbol{\xi}(\xi))) \frac{ds}{d\xi} \, . \tag{10.5.10}$$

The integration extends over all edges e that belong to the appropriate part of the boundary. Notice that a majority of elements is adjacent to neither Cauchy nor Neumann boundaries, and the corresponding boundary integral contributions vanish.

Formulas for the element matrices are obtained substituting master element shape functions for solution $\hat{u} = \hat{\phi}_j$ and test function $\hat{v} = \hat{\phi}_i$. Notice that it is exactly the transformation from master element to physical element that motivates introducing the physical element shape $\phi(\boldsymbol{x})$ functions through the corresponding master element shape functions $\hat{\phi}(\boldsymbol{\xi})$.

Numerical integration. The 1D Gaussian quadrature generates immediately 2D quadrature for the master square element,

$$(\xi_{l_1}, \xi_{l_2}),\ w_{l_1} w_{l_2},\ l_1 = 1, \ldots, N_h,\ l_2 = 1, \ldots, N_v\ .$$

With N_h points in the 'horizontal', and N_v points in the 'vertical' direction, we can integrate *exactly* all polynomials from space $Q^{(p_h, p_v)}$ where $p_h = 2N_h - 1$ and $p_v = 2N_v - 1$.

Construction of Gaussian quadrature for triangles is much more involved and, most of the time, the corresponding Gaussian points and weights are determined numerically, by solving systems of nonlinear algebraic equations. In our implementation, we have used quadrature for triangles derived in [65].

In FE computations, the number of Gaussian points is typically selected in such a way that, for affine elements and constant material data, the quadrature delivers exact values for the stiffness matrix. With element piecewise constant coefficients, and constant jacobians, the highest degree contributions come from the zero order terms. For 1D element of order p, the zero order term contributes with polynomial of order $2p$. The minimum number of quadrature points is thus $N = p + 1$. For a quad element with order $p = (p_h, p_v)$, we use $p_h + 1$ Gaussian points in the horizontal direction, and $p_v + 1$ Gaussian points in the vertical direction. Selection of quadrature points for the triangle is more complicated as some of quadrature points may be outside of the triangle, or weights may be negative. We choose to work with only those quadrature rules that have quadrature points within the master triangle and positive weights.

Step 0: prepare element data that do not vary with integration points
  get order for the element nodes p, p_e
  get geometry d.o.f. $x_{i,k}$
Step 1: compute element integrals contributions
initiate element load vector $\boldsymbol{l}$ and element stiffness matrix $\boldsymbol{B}$ with zero
for each integration points $\boldsymbol{\xi}_l$
  compute values of master element shape functions and their
    derivatives wrt master element coordinates, $\hat{\phi}_k, \partial\hat{\phi}_k/\partial\xi_j$
  use formulas (10.5.3) and (10.5.4) to compute the corresponding physi
    coordinates x_i of the integration point, and their derivatives
    wrt master element coordinates $\partial x_i/\partial\xi_j$
  compute jacobian $|d\boldsymbol{x}/d\boldsymbol{\xi}|$ and inverse Jacobian matrix $\partial\xi_j/\partial x_i$
  get material data a_{ij}, b_j, c and load data f at the point
  use (10.5.6) and (10.5.10) to transform the material and load data
    to $\bar{a}_{nm}, \bar{b}_m, \bar{c}, \bar{f}$
  for each element shape function $k1$
    use (10.5.7) to accumulate for the load vector:

$l_{k1} = l_{k1} + \bar{f} * \hat{\phi}_{k1} * w_l$
for each element shape function $k2$
use (10.5.5) to accumulate for the stiffness matrix:
$B_{k1,k2} = B_{k1,k2} + \left\{ \bar{a}_{mn} \frac{\hat{\phi}_{k2}}{\partial \xi_m} \frac{\hat{\phi}_{k1}}{\partial \xi_n} + \bar{b}_m \frac{\hat{\phi}_{k2}}{\partial \xi_m} \hat{\phi}_{k1} + \bar{c} \hat{\phi}_{k2} \hat{\phi}_{k1} \right\} w_l$
end of the second loop through element shape functions
end of the first loop through element shape functions
end of loop through integration points
Step 2: compute boundary integrals contributions
for each element edge e that lies on the boundary
exit if not on the Neumann or Cauchy boundary
for each integration point ξ_l
evaluate the corresponding master element coordinates $\boldsymbol{\xi}$
proceed in the same way as for the element integrals to
evaluate $\hat{\phi}_k, \partial \hat{\phi}_k / \partial \xi_j, x_i, \partial x_i / \partial \xi_j, \partial \xi_j \partial x_i$
evaluate boundary jacobian $ds/d\xi$
get material data β (if on Γ_C) and load data g
transform the material and load data to $\bar{\beta}, \bar{g}$
for each element shape function $k1$
accumulate for the load vector:
$l_{k1} = l_{k1} + \hat{g} \hat{\phi}_{k1} w_l$
exit if not on Cauchy boundary Γ_C
for each element shape function $k2$
accumulate for the stiffness matrix:
$B_{k1,k2} = B_{k1,k2} + \hat{\beta} \hat{\phi}_{k2} \hat{\phi}_{k1} w_l$
end of second loop through shape functions
end of first loop through shape functions
end of loop through 1D integration points
end of loop through element edges

10.6 Modifi ed element. Imposing Dirichlet boundary conditions

As discussed earlier, assembling of element matrices into the global matrices is accompanied for some shape functions by a change of sign. Existing frontal solvers which perform the assembling of global matrices based on information on nodal connectivities (provided by the user), do not support such an operation. It is for that reason that it is convenient to separate these two operations, the sign change, and the assembling procedure, from each other, by introducing the notion of a *modified element* and *modified element matrices*. The need for introducing the idea of modified element will be even more evident for irregular meshes discussed in next chapters.

The algorithm for generating the modified matrices is as follows.

Step 1: determine the d.o.f. sign factors c_k
Step 2: determine the element matrices B_{ij}, l_i
Step 3: determine the modified element matrices B_{ij}^{mod}, l_i^{mod}
for each shape function i
  $l_i^{mod} = c_i * l_i$
  for each shape function j
    $B_{ij}^{mod} = c_i * c_j * B_{ij}$
  end of second loop through shape functions
end of first loop through shape functions

Once we have accounted for the orientation of the nodes, we follow with the modifications due to Dirichlet boundary conditions. In 1D, the Dirichlet data was reduced to a single vertex node, and the corresponding vertex basis function provided a natural lift for the data. In 2D the situation is more complicated.

Fig. 10.6 depicts a typical situation. A segment of boundary is assigned the Dirichlet BC flag. We assume that the neighboring segments belong to either Neumann or Dirichlet boundary. The initial mesh is generated in such a way that the endpoints of the Dirichlet segment coincide with two vertex nodes. The resulting FE mesh approximates the boundary segment. The Dirichlet BC data $u_D(\boldsymbol{x})$, defined on Γ_D has to be approximated with a function u_{Dh} that it is defined on the FE boundary, and 'lives' in the FE space. We shall discuss the process of approximating the Dirichlet data in detail in the following chapters. For now, we shall assume, that the approximate Dirichlet data u_{Dh} has already been determined, and it is represented in terms of d.o.f. for all nodes belonging to the (approximate) Dirichlet boundary. Similarly as in 1D, the element shape functions provide then a lift of the (approximate) Dirichlet data, used in the FE computations.

The (modified) element matrices will undergo now an extra modification to account for the Dirichlet boundary condition. The strategy is identical with the 1D implementation. The equations corresponding to Dirichlet BC are decoupled from the rest of the equations, with the diagonal entries set to one, and the load vector reset to the value of the d.o.f. already stored in a data structure array. This will 'trick' the solver to return the same value and, consequently, we will not have to differentiate between Dirichlet and non-Dirichlet nodes in the solution process and when storing the solution in the data structure arrays. To maintain the symmetry (if present) of element stiffness matrices, we modify the load vector for all other d.o.f. in the element. The modification to incorporate Dirichlet boundary conditions has to be done for all elements containing even a single Dirichlet node, not only for those that are geometrically adjacent to the Dirichlet boundary. It is for that reason that we need to define the BC flag for nodes rather than for elements.

The formal algorithm for modifying the element matrices to account for the Dirichlet BC's is as

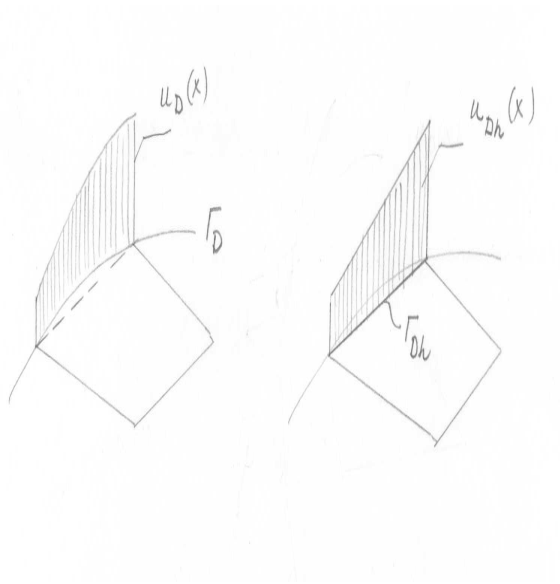


Figure 10.6: Approximation of Dirichlet data

follows.

```
for each element node nod
  exit if not a Dirichlet node
  for each d.o.f.  j of node nod
    get the Dirichlet d.o.f.  u_{D,j}
    modify the load vector:
    for each k
      if k = j then
        Bload(k) = u_D^j
      else
        Bload(k) =Bload(k)−Astiff(k, j) ∗ u_D^j
      endif
    end of loop through the element d.o.f.
    modify the stiffness matrix:
    for each k2
      for each k1
        if k1 = k2 then
          Astif(k1, k2) = 1
        else
          Astif(k1, k2) = 0
        endif
```

```
        end of the second loop through the element d.o.f.
      end of the first loop through the element d.o.f.
    end of loop through d.o.f.  of the node
  end of loop through nodes of the element
```

We emphasize that the modification due to the Dirichlet BC has to be done *after* the modification accounting for the orientation of the mid-edge nodes. This is because the values of d.o.f. for the Dirichlet mid-edge node correspond to the global, and not local orientation of those nodes.

10.7 Postprocessing. Local access to element d.o.f.

Once the global system of equations has been solved, and solution d.o.f. stored in the data structure arrays, we need to be able to access them to do all kind of postprocessing, beginning with a simple evaluation of the solution values at specific points. We access element d.o.f. *after* the modification involving the sign factors. Once the element d.o.f. connectivities $i = i_{K,k}$ and the corresponding sign factors $c_{K,k}$ have been determined, the modification of element d.o.f. is straightforward.

```
for each d.o.f.  k of element K
  get d.o.f.  connectivity i = i_{K,k} and sign factor c_{K,k}
  u_k^{loc} = u_i^{glob} * c_{K,k}
end of loop through element d.o.f.
```

The same algorithm is used to return element geometry d.o.f., necessary for the calculation of element matrices discussed earlier. Given now an element K and master element coordinates $\boldsymbol{\xi}$ of a point within the element, we compute the corresponding value of the solution at the points using a formula identical with (10.5.3),

$$u = \sum_k u_k^{loc} \hat{\phi}_k(\boldsymbol{\xi}) \, .$$

Similarly, we can compute the derivatives with respect to master element coordinates and, using the chain formula, derivatives wrt physical coordinates as well,

$$\begin{aligned} \frac{\partial u}{\partial \xi_i} &= \sum_k u_k^{loc} \frac{\partial \hat{\phi}_k}{\partial \xi_i}(\boldsymbol{\xi}) \\ \frac{\partial u}{\partial x_j} &= \frac{\partial u}{\partial \xi_i} \frac{\partial \xi_i}{\partial x_j} \, , \end{aligned}$$

with derivatives $\partial \xi_i / \partial x_j$ calculated the same way as in the calculation of element matrices.

The situation is more difficult if, instead of element number and master element coordinates (ξ_1, ξ_2), we are given just *physical coordinates* of a point $\boldsymbol{x}$ in the FE domain. This happens, for instance, if we wish to evaluate the solution values along a given crossection. Then, for given point (x_1, x_2) we must be able to return the corresponding element K and the corresponding master element coordinates (ξ_1, ξ_2). The computation involves inverting element map (10.5.3). The situation is trivial for an affine map. In the case of a general isoparametric element, inverting the element map involves solution of a system of two nonlinear equations given by (10.5.3). We use then the standard Newton-Raphson procedure, starting with an initial value corresponding to an affine map approximation of the actual element map (obtained by using vertex geometry d.o.f., for quads we use three vertices only). We have to loop through all elements and try them one by one, until we find the right one. If the returned master element coordinates are within the range of the master element, we are in the right element, if not, we continue with the next one. If the Newton-Raphson algorithm does not converge, this indicates that we are in a wrong element, too.

10.8 Projection-based interpolation

According to our discussion in Section 5.9, the definition of the two elements should be completed with the introduction of *interpolation operators*. We shall generalize the 1D interpolation procedure introduced in Section 5.2. The idea is based on the same three assumptions as in 1D: *locality, optimality* and *global continuity*. Given a function U defined on a finite element, we define its projection-based interpolant u^p in three steps.

1. Locality and global continuity conditions imply that interpolant u^pmust coincide with U at the vertex nodes,
$$u^p(\boldsymbol{a}) = U(\boldsymbol{a}), \quad \text{for each vertex node } \boldsymbol{a}\ ,$$

2. Locality and global continuity imply that values of the interpolant along an edge e must be determined using values of function U along the same edge only. Optimality condition implies that we must use a projection,
$$\|u^p - U\|_e \to \min\ .$$
We will have to decide what 'edge norm' $\|\cdot\|_e$ we should use.

3. Once we have determined the interpolant over the element boundary, locality and optimality imply again that we should use a projection,
$$\|u^p - U\|_K \to \min\ .$$

The choice of the element norm suggests itself: we should use the Sobolev norm. After all, we want to minimize the FE error measured in that norm. Similarly as in 1D, we settle for using the

first order seminorm. The difference between using the norm or seminorm, for $h/p \to 0$ becomes insignificant. The choice of the norm for edges is more delicate, and it is dictated by the Trace Theorem - we need to use the fractional norm $H^{\frac{1}{2}}(e)$ (see [43] for a detailed analysis). It is difficult to evaluate the fractional norm explicitly. In practical computations, we shall replace it with the first order Sobolev norm weighted with an additional geometrical factor,

$$\|u\|_e^2 \approx \int_e |\frac{\partial u}{\partial s}|^2 \underbrace{\left(\frac{ds}{d\xi}\right)}_{\text{weight}} ds = \int_0^1 |\frac{d\hat{u}}{d\xi}|^2 \, d\xi \, .$$

The weighted norm scales the same way as the fractional norm in terms of element size (edge length) h. The interpolation procedure is equivalent to the following system of variational conditions.

$$\begin{cases} u^p(\boldsymbol{a}) = U(\boldsymbol{a}) & \text{for each vertex } \boldsymbol{a}, \\ \displaystyle\int_e \frac{d(u^p - U)}{ds} \frac{d\phi}{ds} \frac{ds}{d\xi} \, ds, & \text{for every edge bubble function } \phi \, , \\ \displaystyle\int_K \boldsymbol{\nabla}(u^p - U) \boldsymbol{\nabla}\phi \, d\boldsymbol{x}, & \text{for every element bubble function } \phi \, . \end{cases}$$

Notice the generality of the procedure. It can be applied for any of the two discussed master elements, a parametric element corresponding to an arbitrary map, and a group elements, e.g. resulting from an h-refinement. In each of those cases we deal with a 'space of shape functions' onto which we project. For master (and affine) elements, these will be polynomials, for a refined element - piece-wise polynomials, or their corresponding counterparts implied by the definition of the general parametric element. By edge bubble functions we understand edge traces of functions from the 'space of shape functions' that vanish at vertices, by element bubble functions we understand those functions that vanish on the element boundary.

It is perhaps easier to see the recursive logic of the interpolation, if we explicitly identify vertex, edge, and element contributions to the interpolant,

$$u^p = u_1^p + \underbrace{\sum_e u_{2,e}^p}_{u_2^p} + u_3^p \, .$$

Here u_1^p is *any* function from the 'space of shape functions' onto which we project which coincides with the interpolated function U at vertex nodes, $u_{2,e}^p$ is an edge bubble corresponding to edge e, and u_3^p is an element bubble. The edge and element contributions are determined by solving linear variational problems,

$$\begin{cases} \displaystyle\int_e \frac{du_{2,e}^p}{ds} \frac{d\phi}{ds} \frac{ds}{d\xi} \, ds = \int_e \frac{d(U - u_1^p)}{ds} \frac{d\phi}{ds} \frac{ds}{d\xi} \, ds, & \text{for every edge bubble } \phi \, , \\ \displaystyle\int_K \boldsymbol{\nabla} u_3^p \boldsymbol{\nabla}\phi \, d\boldsymbol{x} = \int_K \boldsymbol{\nabla}(U - u_1^p - u_2^p) \boldsymbol{\nabla}\phi \, d\boldsymbol{x}, & \text{for every element bubble } \phi \, . \end{cases}$$

For an element, the interpolation can be performed either on the physical or on the master element. Let us focus on the physical element first. Switching to master element coordinates, the interpolation over a triangular element reduces to the following conditions,

$$\begin{cases} \hat{u} \in \mathcal{P}^p_{p_e}(\hat{K}) & \\ \hat{u}^p(\boldsymbol{a}) = \hat{U}(\boldsymbol{a}) & \forall \boldsymbol{a}, \\ \displaystyle\int_0^1 \frac{d(\hat{u}^p - \hat{U})}{d\xi}\frac{d\phi}{d\xi}\, d\xi, & \forall e, \quad \forall \phi \in \mathcal{P}^{p_e}_{-1}(e), \\ \displaystyle\int_{\hat{K}} \bar{a}_{ij}\frac{\partial(\hat{u}^p - \hat{U})}{\partial \xi_j}\frac{\partial \phi}{\partial \xi_i}\, d\boldsymbol{\xi} = 0 & \forall \phi \in \mathcal{P}^p_{-1}(\hat{K})\,. \end{cases}$$

Analogous formulas hold for a quadrilateral element, with space $\mathcal{Q}^p_{p_e}$ replacing $\mathcal{P}^p_{p_e}$. Metric $\bar{a}_{ij}$ results from the change of variables, and it is given by the formula,

$$\bar{a}_{ij} = \frac{\partial \xi_i}{\partial x_k}\frac{\partial \xi_j}{\partial x_k}\,|\frac{\partial x_i}{\partial \xi_j}|\,.$$

In the case of the interpolation on the master element, metric $\bar{a}_{ij}$ is replaced with the diagonal matrix. The two interpolation procedures will therefore yield exactly the same result only in the case when $\hat{a}_{ij} = c\delta_{ij}$ (multiplication by a constant does not matter). This is the case only if the master and physical element are similar, i.e. the element map reduces to a similarity.

In other words, in contrary to the 1D case, commutativity property (5.6.4) does not longer hold. We arrive at the dilemma which of the two interpolations should be used: should we interpolate directly on the physical element, or should we switch first to the master element, interpolate there, and bring the interpolant back to the physical space? We will return to this issue in the next chapters.

Exercises

Exercise 10.1 Recall the definition of the *affine coordinates*. Given three non-collinear points $\boldsymbol{a}_j \in \mathbb{R}^2, j = 1, 2, 3$, and an arbitrary point $\boldsymbol{a}$, we can always represent vector $\boldsymbol{a} - \boldsymbol{a}_1$ as a linear combination of vectors $\boldsymbol{a}_2 - \boldsymbol{a}_1$ and $\boldsymbol{a}_3 - \boldsymbol{a}_1$,

$$\boldsymbol{a} - \boldsymbol{a}_1 = \lambda_2(\boldsymbol{a}_2 - \boldsymbol{a}_1) + \lambda_3(\boldsymbol{a}_3 - \boldsymbol{a}_1),$$

or, equivalently,

$$\boldsymbol{a} = \sum_{j=1}^{3} \lambda_j \boldsymbol{a}_j,$$

where, by definition, $\lambda_1 = 1 - \lambda_2 - \lambda_3$.

- Show that the construction of the affine coordinates is *independent* of the choice of the origin. For instance, we can introduce first λ_1, λ_2 by:

$$\boldsymbol{a} - \boldsymbol{a}_3 = \lambda_1(\boldsymbol{a}_1 - \boldsymbol{a}_3) + \lambda_2(\boldsymbol{a}_2 - \boldsymbol{a}_3),$$

  and then define $\lambda_3 = 1 - \lambda_1 - \lambda_2$. The resulting affine coordinates are identical.

- Show that the affine coordinates are *invariant* under affine transformation. More precisely, given an affine map,

$$\boldsymbol{x} \to \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{B}\boldsymbol{x} + \boldsymbol{b},$$

  where $\boldsymbol{B}$ is a non-singular linear map (matrix), show that affine coordinates of point $\boldsymbol{a}$ with respect to points $\boldsymbol{a}_j$ are equal to the affine coordinates of point $\boldsymbol{f}(\boldsymbol{a})$ with respect to points $\boldsymbol{f}(\boldsymbol{a}_j)$.

- Use the invariance property to interpret the affine coordinates as the area coordinates by showing that,

$$\lambda_j = \frac{\text{area of triangle } \{\boldsymbol{a}\boldsymbol{a}_{j+1}\boldsymbol{a}_{j+2}\}}{\text{area of triangle } \{\boldsymbol{a}_1\boldsymbol{a}_2\boldsymbol{a}_3\}}\,, \quad j = 1, 2, 3\,,$$

  with index j changing modulo 3.

- Compute affine coordinates of the centroid of the triangle.

Exercise 10.2 Use the affine coordinates to compute the edge coordinate $\xi = (\lambda_{i+1} - \lambda_i + 1)/2$ used in constructing the extension of edge kernel functions, comp. Fig. 10.3.

Exercise 10.3 The choice of direction $\lambda_{i+1} - \lambda_i = const$ in the construction of the extension of the edge kernel functions is symmetric with respect vertices $i, i+1$. Show that using any other set of parallel lines,

$$\alpha\lambda_i + \beta\lambda_{i+1} = const\,,$$

will result in constructing a valid polynomial extension as well. Interpret the case $\lambda_i = const$ or $\lambda_{i+1} = const$. Discuss an alternative idea of defining the value of the extension at a point within the triangle, by projecting the point onto the i-th edge in the direction of the line passing through the point and the vertex opposite to the edge (the direction of the projecting line will depend then upon the point). Explain why we cannot use this idea when defining polynomial extensions.

Exercise 10.4 The use of affine coordinates in the definition of the shape functions for a triangle guarantees the *rotational invariance*. Let $\boldsymbol{f} : \mathbb{R}^2 \to \mathbb{R}^2$ be the affine map that maps vertex $\boldsymbol{a}_j$ into vertex $\boldsymbol{a}_{j+1}$,

$$\boldsymbol{f}(\boldsymbol{a}_j) = \boldsymbol{a}_{j+1}, \quad j = 1, 2, 3.$$

Show that

$$\lambda_{j+1}(\boldsymbol{x}) = \lambda_j(\boldsymbol{f}^{-1}(\boldsymbol{x})), \quad j = 1, 2, 3.$$

Show that the same property holds for the edge shape functions for the master triangular element introduced in the text,

$$\hat{\phi}_{3+i+1,j}(\boldsymbol{x}) = \hat{\phi}_{3+i,j}(\boldsymbol{f}^{-1}(\boldsymbol{x})), \quad i = 1, 2, 3, \quad j = 1, \ldots, p-1\,.$$

Demonstrate that it is impossible to introduce middle node (bubble) shape functions that would be simultaneously hierarchical and rotationally invariant.

Exercise 10.5 Prove that the shape functions introduced for the quadrilateral and triangular master elements are linearly independent. *Hint:* Assume that a linear combination of the element shape functions vanishes. Reduce the linear combination to a single element vertex and draw a conclusion about the degree of freedom correspondiong to the vertex shape function. Reduce then the linear combination to an edge and draw a conlusion about the d.o.f. corresponding to the edge shape functiont, etc. Spell out constructive assumptions for a completely arbitrary set of vertex, edge, and interior shape functions to be linearly independent.

Chapter 11

$2Dhp$ Code

In this chapter we introduce the reader to our $2Dhp$ code. The logic of the code is very much the same as for $1Dhp$, so we will not discuss everything with such a great detail as we have done it in 1D. We shall review the content of the code, and discuss the 2D data structure. After a short explanation of the graphics package, we review the fundamental routines performing the tasks discussed in the previous chapter - computation of element matrices, modifications to account for nodes orientations and Dirichlet BC. We update the information on the interface with the frontal solver, and discuss the next assignment - writing your own processor. Discussion on initial mesh generation is delayed to the next chapter.

11.1 Getting started

The code consists of the following subdirectories.

- *adapt* - routines executing automatic hp-refinements,
- *adaptE* - routines executing automatic hp-refinements for Maxwell problems,
- *adapth* - routines executing automatic h-refinements,
- *antenna* - element routines for an antenna problem,
- *blas* - basic linear algebra routines,
- *commons* - system common blocks
- *constrs* - constrained approximation routines,
- *constr_util* - constrained approximation utilities,
- *datstrs* - data structure routines,

- *elasticity* - element routines for the linear elasticity problem,
- *elem_util* - element utilities,
- *frontsol* - frontal solver routines (real version),
- *frontsolz* - frontal solver routines (complex version),
- *gcommons* - graphics common blocks,
- *graph_2Dhp* - the $2Dhp$ code graphics programs,
- *graph_geom* - the GMP (Geometric Modeling Package) graphics programs,
- *graph_interf* - X-windows interface routines,
- *graph_util* - graphics utilities,
- *hp_interp* - projection based interpolation routines,
- *laplace* - element routines for the model problem,
- *main* - the main program,
- *maxwell* - element routines for Maxwell equations,
- *meshgen* - initial mesh generation routines,
- *meshmods* - mesh modification routines,
- *solver1* - interface with the frontal solver,
- *utilities* - general utilities routines.

Additionally, you should find files *head, makefile* and subdirectory *documentation*.

Dependently upon values of flags *COMPLEX*, *EM* and *MAXEQNS* in the makefile, the routines can be preprocessed to yield different versions of the code. For a while, we shall restrict ourselves to the case of a single, real-valued equation discussed so far. The parameters should be set to *COMPLEX=0, EM=0, MAXEQNS=2*. Please ignore for now directories *adaptE, antenna, elasticity, frontsolz, maxwell*.

11.2 Data structure in FORTRAN 90

The very first parameter in *module/data_structure2D*, *NDIMEN=2* specifies the geometry of the problem and it is set to 2. After resetting the parameter to *NDIMEN=3* the package can be used as a basis to write an adaptive boundary element code. The code does not contain any boundary element routines, though.

We introduce three *user-defined structures* [90], p.353 (see *module/data_structure2D*):

- type *initial mesh element*.
- type *vertex*,
- type *node*,

The attributes of an initial mesh element are:

- element type (a character variable indicating triangular or quadrilateral element)
- integer array *nodes* containing the seven (triangles) or nine (quads) node numbers for the element, listed in the order: vertices, mid-edge nodes, and the middle node,
- integer *orient* containing orientations for the element edges, packed into one integer using the binary system,
$$\textit{orient} = \begin{cases} n_1 + 2n_2 + 4n_3 & \text{for triangles} \\ n_1 + 2n_2 + 4n_3 + 8n_4 & \text{for quads,} \end{cases}$$
where n_i denotes the orientation of the i-th edge of the element,
$$n_i = \begin{cases} 0 & \text{if the global and local orientations are consistent} \\ 1 & \text{otherwise.} \end{cases}$$
- integer array $neig$ containing numbers of three (triangles) or four (quads) neighbors of the element,
- an extra integer *geom_interf* for interfacing with the Geometrical Modeling Package (GMP).

The attributes of a vertex node include: boundary condition flag, father of the node, GMP interface flag, and two arrays *coord*, and *zdofs*, containing geometry and solution d.o.f.

The attributes of a non-vertex node include: node type (a character indicating whether the node is a mid-edge, a triangle, or a quad middle node), integer order of approximation, integer boundary condition flag, refinement flag, isotropy flag, father of the node, sons of the node, GMP interface flag, and two real arrays, *coord*, containing geometrical degrees of freedom, and *zdofs*, containing

the solution degrees of freedom. Both the geometry and solution d.o.f. are allocated dynamically, dependently upon the order of approximation for the node. The array *sons* is also allocated dynamically, dependently upon the refinement kind.

Note the following details:

- the integer specifying order for a quad middle node is actually a nickname defined as

$$order = p_h * 10 + p_v \ ,$$

  where p_h, p_v are the *horizontal* and the *vertical* orders of approximation for the element;

- both *coord* and *zdofs* are defined as arrays with two indices. The first index of *coord* corresponds to the number of coordinates and varies between 1 and *NDIMEN = 2,3*. The first index of array *zdofs* indicates a component of the solution, and for the single elliptic equation being discussed, it is always equal one.

Please ignore the remaining attributes of the data structure objects for now.

The entire information about a mesh is stored in three allocatable arrays, *ELEMS*, *NVERS* and *NODES*, as declared in the data structure module. As in 1D, the module also includes a declaration for a number of integer attributes of the mesh like number of elements in the initial mesh and current mesh, number of vertex and non-vertex nodes, etc.

11.3 Fundamentals

System files, IO. The code uses the same system files and organization of IO as $1Dhp$.

The main program. Program *main/main* is the main driver for the code. The structure of the program is the same as in 1D. The preprocessor stage consists of two main steps:

- routine *geometry/input_geometry* reads in data for the *Geometrical Modeling Package* (GMP) defining the geometrical object (domain) to be considered;

- routine *meshgen/hp2gen* generates the initial mesh.

We shall discuss both programs and the corresponding input data specified in *files/input* in the next chapter. For now, you may copy any of the input files in directory *files/inputs* onto *files/input* and use it to study the code.

The preprocessing stage includes also reading global control parameters, initiating quadrature data, precomputing matrices for element calculations and mesh optimization, and initiating a number of arrays supporting refinements. The main menu offers the same options as in the 1D code plus the possibility of visualizing the geometry model.

Graphics. The main graphics driver for the code - *graph_2Dhp/graph* contains calls to the following graphics programs:

- *graph_2Dhp/grsurf* - displays a graphical representation of the current FE mesh, or a contour map for a defined scalar-valued field,
- *graph_2Dhp/mesh* - allows for an interactive modification of the mesh,
- *graph_2Dhp/section* - draws solution along a specified section,
- *graph_2Dhp/rates* - plots convergence history.

The graphics routines are using the same interface with Xwindows as $1Dhp$.

Due to the possibility of displaying two-dimensional meshes on surfaces (for boundary element computations), the graphics package is set up to use the standard *painter's algorithm*.

Quadrature data. All quadrature data are initiated in routine *elem_util/ const* and stored in common blocks *commons/cint* and *commons/cintr*. The necessary order of quadrature for triangle of order $nord$ is specified by value in array *NSELECT(nord)* in *commons/cintr*. Notice that, for the triangle, the corresponding number of integration points does not grow exactly in proportion to the order of approximation, see array *NRGAUPO* initiated in routine *elem_util/const*.

11.4 The element routine

Please open routine *laplace/elem* and compare it, step by step, with discussion in Section 10.5. The routine covers both the quadrilateral and the triangular elements. All local variables have been dimensioned using data structure parameters *NDIMEN,MAXEQNS* and parameter *NPMAX* from *commons/cpmax* specifying the anticipated maximum order of approximation, with dimensions corresponding to the quad element. Due to the design of the data structure objects and supported quadrature data, *NPMAX* must not exceed 9.

The order of approximation, returned by *datstrs/findap* contains four integers for the triangular element: the order for the three edges, and the order for the middle node. For the quad element, array *norder* contains four integers specifying the order for the edges, with the fifth integer specifying the order for the element middle node. Recall that the middle node order is returned in the form of a nickname containing *horizontal order nordh*, and *vertical order nordv*,

*norder(5) = nordh*10 + nordv* .

Number of element d.o.f., calculated in *datstrs/celndof*, is necessary for initiating element matrices with zeros. Element geometry d.o.f. are returned by *constrs/nodcor*, and boundary conditions flags

are returned by *datstrs/find_bc*. Similarly as in 1D, for an initial mesh element, both routines return simply an info stored in the data structure arrays. For elements resulting from h-refinements, the information is reconstructed from the nodal trees; we will discuss it later. Notice that, contrary to the 1D code, the elements are *isoparametric* - all nodes have the corresponding geometry d.o.f.

The routine is organized into two separate parts: for the triangular and for the quadrilateral element. You should be able to relate the code with the discussion in Section 10.5.

In the very end, if the global control flag *INT_FLAG* is up, the element load vector is recalculated using an adaptive integration similar to the 1D version discussed in the previous chapters. This option should be used for studying problems with known exact solution.

Finally, notice that the element routine takes into account only Neumann boundary conditions. You will be asked to add Cauchy BC in the next assignment. Accounting for Dirichlet BC's will be discussed next.

11.5 Modified element. Imposing Dirichlet boundary conditions.

As discussed earlier, assembling of element matrices into the global matrices is accompanied for some shape functions by a change of sign. Existing frontal solvers that perform the assembling of global matrices based on information on nodal connectivities (provided by the user), do not support such an operation. It is for that reason that it is convenient to separate[1] these two operations, the sign change, and the assembling procedure, from each other, by introducing the notion of the *modified element*.

The information about the modified element includes:

- a list of element vertex nodes: *Nod1*$(i), i = 1, \ldots,$ *Nvert*,
- a list of element non-vertex (mid-edge and middle) nodes: *Nod2*$(i), i = 1, \ldots,$ *Nrnod*,
- the corresponding number of shape functions (d.o.f) per node: *Ndof2*$(i), i = 1, \ldots,$ *Nrnod*,
- the modified element load vector: *Bload*$(k), k = 1, \ldots,$ *Nrdof*,
- the modified element stiffness matrix: *Astiff*$(k,l),\ k,l = 1, \ldots,$ *Nrdof*.

The shape functions are ordered following the order of nodes and the corresponding order of shape functions for each of the nodes. The total number of shape functions (degrees of freedom) for the element, *Nrdof*, is obtained by summing up the numbers of shape functions for each node:

$$Ndof = Nvert + \sum_{i=1}^{Nrnod} Ndof2(i)$$

[1] The need for this separation will be even more clear for h-adapted meshes discussed in the next chapters.

For regular meshes discussed in this chapter, the lists of vertex and non-vertex nodes are organized in the same order as they are listed in the data structure array *ELEMS(nel)%nodes*. The corresponding matrices are obtained by multiplying the local matrices by the sign factors discussed earlier. Let ϕ_k denote k-th shape function of element K, and let e_i be the corresponding basis function. Recall the definition of the sign factor,

$$c(k) = \begin{cases} 1, & \text{if } e_i|_K = \phi_k \\ -1, & \text{if } e_i|_K = -\phi_k \, . \end{cases}$$

The calculation of the modified element matrices is done in routine *constrs/celem* following the algorithm discussed in Section 10.6.

Imposing Dirichlet boundary conditions. We impose the Dirichlet boundary conditions *always* on the modified element matrices and it follows the discussion in Section 10.6. The modifications are done in routine *laplace/bcmod*.

11.6 Assignment 4: Assembling of global matrices

Once we have taken care of the sign factors, and the Dirichlet BC's, the assembling algorithm is identical with its 1D version.

```
for each element in the mesh
  compute the element matrices
  modify the element matrices to include the sign factors
  modify the (modified) element matrices to account for Dirichlet BC
  determine the (modified) element d.o.f.  connectivities
  assemble the the (modified) element matrices into the global matrices
end of loop through elements
```

Natural order of nodes and nodal d.o.f. We use the same algorithm as in 1D to establish the *natural order of nodes* and the corresponding *natural order of d.o.f.*. The order of nodes is again established by a *natural order of elements*, and the order of nodes within an element. For an initial mesh, the order of elements is determined by the initial mesh generator; for refined meshes, we shall use the same algorithm as in 1D, based on middle nodes trees, and realized by calling routine *datstrs/nelcon*.

Interface with the frontal solver. The interface with the frontal solver follows exactly the same logic as in the 1D code. All interface routines have the same names. We point out only to three differences.

- The nodes of the modified element are returned by routine *constrs/logic*. For initial mesh elements the nodal connectivities are simply recovered from data structure array *ELEMS*. For refined meshes, the connectivities are reconstructed using a 2D analog of the algorithm discussed for the 1D code. The algorithm is more complicated as it also covers the case of elements with *constrained nodes*. We will discuss it in detail in a subsequent chapter. All these details are absolutely inessential for the logic of the frontal solver. The only information that the frontal solver needs is the element nodes nicknames, and then the corresponding element matrices.

- Consistently with our discussion, routine *solver1/solin2* provides element matrices for the modified element, returned by *constrs/celem*.

- The frontal solver uses different storage mode for symmetric problems. Dependently upon flag *ISYM* from common block *frontsol/surfsc1*, routine *constrs/celem* returns the modified element stiffness matrix assembled either in a symmetric (lower half of the matrix only) or the full mode. At present, the code is set up to solve the Laplace problem that yields symmetric matrices.

We are ready now to formulate our next assignment.

Assignment 4.

1. Use file *files/inputs/input_Lqt*, corresponding to an L-shape domain to generate the initial mesh. Run the code and visualize the mesh consisting of five elements: four triangles and one quadrilateral. Modify the boundary conditions flags in the input file (the very end of the file), for all three kinds of boundary conditions to be present. Use the interactive mesh modification to increase the order of elements to $p \geq 3$, to create a mesh with a non-uniform order of approximation. Include the picture with the mesh in your assignment. Mark manually the selected boundary conditions.

2. Select one of the polynomial solutions in routine *laplace/exact* or provide a new one. Make sure that the solution can be recovered exactly with your FE mesh, i.e. that the selected order of elements is greater or equal than the polynomial order in the manufactured solution.

3. Modify routine *laplace/elem* to include the Cauchy BC. Add routine *laplace/getc* specifying Cauchy material data coefficient β. Modify routine *laplace/getg* to account for both Neumann and Cauchy BC's.

4. Write your own version of routine *constrs/celem* for regular meshes only. You may restrict yourself to the initial meshes only. In such a case, the element nodal connectivities and orientation for the mid-edge nodes can be found in data structure array *ELEMS*. For a more

general case of refined meshes, you will need to call routine *constrs/history*. The information about the element nodes and their orientation is then returned, as in 1D, through *module/tree*. Test your routine against *constrs/celem* for all five elements in the mesh.

5. Following the guidelines from the 1D code, write your own processor, based on the interface with the Gaussian solver *blas/gausse*. Use the picture with the mesh to mark the global (sequential) number for the first d.o.f. associated with each node. After assembling each of the five elements, print a graphical image of the global stiffness matrix and the global load vector, with a mark indicating each entry corresponding to the assembled element contributions.

6. Test your solver against the manufactured exact solution. You may also test it against the frontal solver results. Provide an evidence that you have successfully debugged the code.

Hint: It is very unlikely that you will be able to complete the assignment for the ultimate, variable order mesh at once. You may want to begin with simple meshes first, starting with linear elements, and increasing gradually the order of approximation in one element at a time. If this does not work, you may want to change the input file to *files/inputs/ input_quad*, *files/inputs/input_trian*, or *files/inputs/input_wedge* that will generate for you meshes consisting of one or two elements only.

Chapter 12

Geometric Modeling and Mesh Generation

This chapter will combine a discussion on fundamental modeling techniques with an introduction to our *Geometrical Modeling Package* (GMP). We present the idea of a *manifold*, parametrizations, and discuss the fundamental requirement of *compatibility of parametrizations*. Then we demonstrate how the condition is met in practice by using the techniques of *transfinite interpolation*, and *implicit parametrizations*. Finally we discuss the use of the *projection-based interpolation* introduced earlier, for the initial mesh generation, and geometry updates following mesh refinements, as well as the approximation of Dirichlet data.

12.1 Manifold representation

We assume that domain $\Omega \subset \mathbb{R}^2$ can be represented as a union of curvilinear triangles and quadrilaterals forming a *regular mesh*. This means that any two subdomains can share either a vertex or a whole edge but *not* a portion of an edge only. Example of an admissible partition is shown in Fig. 12.1. Each of the curvilinear triangles or rectangles forming the partition comes with its own parametrization, a map from the reference triangle or square onto the curvilinear *figure*,

$$\boldsymbol{x} = \boldsymbol{x}(\boldsymbol{\eta}), \quad x_i = x_i(\eta_j), \quad j = 1, 2,\ i = 1, \ldots, n\ . \tag{12.1.1}$$

The concept is almost identical with the idea of the reference element discussed in the previous chapters. We shall use consistently letters η and ξ to distinguish between reference and master coordinates. The reference triangle and rectangle are two-dimensional figures, we always have $\boldsymbol{\eta} = (\eta_1, \eta_2)$, but the number of physical coordinates x_i may be equal to two, for planar domains, or to three, for manifolds in $\mathbb{R}^3$. The second case is important when using the code to support a Boundary Element discretization for a 3D problem. Techniques for generating parametrizations for planar or curved manifolds are different.

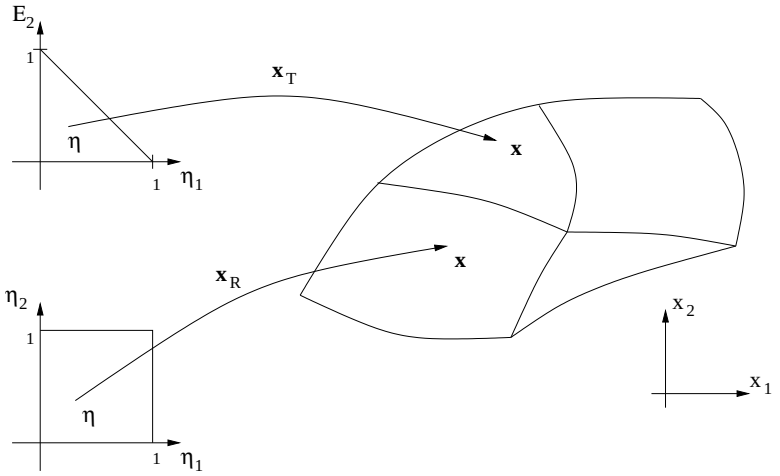


Figure 12.1: A two-dimensional manifold in $\mathbb{R}^2$

The parametrizations are assumed to be compatible with each other. Intuitively speaking, when you generate a mesh along a curve shared by two figures, you can use either parametrization for the two neighboring figures, and the results should be the same. In order to express this condition precisely, we introduce explicitly the following geometrical entities:

Points, Curves, Triangles, Rectangles, Hexahedra, Prisms.

The last two objects are relevant for the modeling of 3D manifolds, and we will only briefly comment on them, refering to [123] for details.

The points come with two or three physical coordinates, and a complete curve connectivities information, i.e. we specify how many curves are connected with the point and list their curve numbers.

Each curve is represented with a map from the reference interval into $\mathbb{R}^n$,

$$[0,1] \ni \eta \rightarrow \boldsymbol{x}_C(\eta) \in \mathbb{R}^n \ .$$

The parametrization determines the *orientation of the curve*. We also specify a complete topology information (point and figures connectivities). This includes the two end points whose coordinates must be compatible with the curve parametrization, i.e. $\boldsymbol{x}_C(0)$ coincides with the coordinates of the point marking the beginning of the curve, and $\boldsymbol{x}_C(1)$ coincides with the coordinates of the point marking the end of the curve.

We also list *figures* adjacent to the curve. In 2D, the number of adjacent figures is either one (when the curve lies on the boundary of the domain), or two (the curve lies in the interior of the domain). In 3D modeling, the number of adjacent figures may be greater than two. The figure numbers are listed in an arbitrary order, but with a sign indicating the compatibility of *local* and

global orientations. This is the same concept as for finite elements. The edges of a reference triangle of rectangle are assumed to be oriented counterclockwise. This corresponds to the following edge parametrizations. For triangle:

$$\begin{cases} \eta_1 = \zeta \\ \eta_2 = 0 \end{cases} \quad \begin{cases} \eta_1 = 1 - \zeta \\ \eta_2 = \zeta \end{cases} \quad \begin{cases} \eta_1 = 0 \\ \eta_2 = 1 - \zeta \, , \end{cases}$$

and for rectangle,

$$\begin{cases} \eta_1 = \zeta \\ \eta_2 = 0 \end{cases} \quad \begin{cases} \eta_1 = 1 \\ \eta_2 = \zeta \end{cases} \quad \begin{cases} \eta_1 = 1 - \zeta \\ \eta_2 = 1 \end{cases} \quad \begin{cases} \eta_1 = 0 \\ \eta_2 = 1 - \zeta \, . \end{cases}$$

If the orientation induced by the local and global parametrizations of a curve are identical, the figure's nickname is listed with a positive sign, otherwise it is listed with a negative sign.

Finally, for each figure, we list the three or four curves constituting its edges, with a sign indicating again the compatibility of the local and global orientations. In 3D, we add the information on adjacent hexahedra or prisms.

Let $\boldsymbol{\eta}(\zeta)$ be the local parametrization for a particular edge of a triangle T or rectangle R, occupied by a curve C. We account for the (in)compatibility of orientations by introducing the map

$$\zeta(\eta) = \begin{cases} \eta & \text{if the local and global orientations are the same,} \\ 1 - \eta & \text{otherwise .} \end{cases}$$

The condition on compatibility of parametrizations can now be expressed precisely,

$$\boldsymbol{x}_T(\boldsymbol{\eta}(\zeta(\eta))) = \boldsymbol{x}_C(\eta) \quad \text{for triangles, or} \quad \boldsymbol{x}_R(\boldsymbol{\eta}(\zeta(\eta))) = \boldsymbol{x}_C(\eta) \quad \text{for rectangles.}$$

12.2 Construction of compatible parametrizations

The parametrizations are constructed using a 'bottom - up' strategy.

First we prescribe the points and their coordinates.

Next we construct the curve parametrizations and make sure that they match the already prescribed point coordinates. The simplest example is a segment of straight line. If $\boldsymbol{x}_{P_1}$ and $\boldsymbol{x}_{P_2}$ denote the coordinates of the beginning and end of the curve, the parametrization is simply defined as,

$$\boldsymbol{x}_C(\eta) = \boldsymbol{x}_{P_1}(1 - \eta) + \boldsymbol{x}_{P_2}\eta \, ,$$

i.e. we use the linear *blending functions*.

For a segment of circle, shown in Fig 12.2, we can use the complex plane to construct a convenient parametrization,

$$\boldsymbol{x} = \boldsymbol{x}_O + \mathbf{OA} \cdot e^{i\theta\eta} \, ,$$

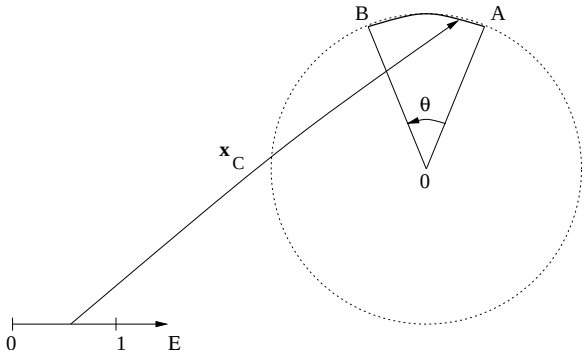


Figure 12.2: Segment of a circle in $\mathbb{R}^2$

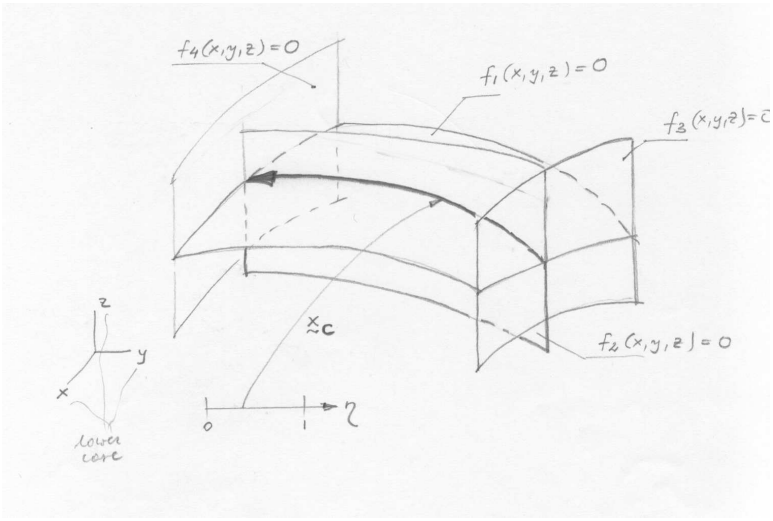


Figure 12.3: Implicit curve

or, in terms of coordinates x, y,

$$\begin{cases} x = x_O + (x_A - x_O)\cos(\theta\eta) - (y_A - y_O)\sin(\theta\eta) \\ y = y_O + (x_A - x_O)\sin(\theta\eta) - (y_A - y_O)\cos(\theta\eta)\,. \end{cases}$$

Angle θ is determined using coordinates of points A, B, O.

Another technique, especially useful in 3D modeling, is that of an *implicit curve*. We assume that the curve is specified in terms of four surfaces iluustrated in Fig. 12.3. Each of the four surfaces is assumed to be given in the (implicit) algebraic form,

$$f_i(x, y, z) = 0 \quad i = 1, \ldots, 4\,,$$

with the possibility of evaluating $f_i(\boldsymbol{x}), \boldsymbol{\nabla} f_i(\boldsymbol{x})$ at an arbitrary point $\boldsymbol{x}$.

The intersection of the first two surfaces determines the geometry of a curve. Surfaces $f_3 = 0$, and $f_4 = 0$ are cutting off a segment $\overset{\frown}{AB}$ of the curve. The surfaces must be compatible with the

end points of the curve, in terms of position and orientation. We assume that

$$\boldsymbol{\nabla} f_3(A) \cdot \mathbf{AB} > 0, \quad \boldsymbol{\nabla} f_4(B) \cdot \mathbf{AB} > 0 \, .$$

If the condition is not satisfied, we replace f_i, with $-f_i, i = 3, 4$. We consider then the following system of nonlinear equations parametrized with parameter $\eta \in [0, 1]$,

$$\begin{cases} f_1(x, y, z) = 0 \\ f_2(x, y, z) = 0 \\ f_3(x, y, z)(1 - \eta) + f_4(x, y, z)\eta = 0 \, . \end{cases}$$

Newton-Raphson technique is used to solve the system numerically for x, y, z, starting with the point provided by the explicit parametrization of the straight line segment AB. Once coordinates x, y, z are known, we differentiate the equations above wrt η,

$$\begin{cases} \frac{\partial f_1}{\partial x}(x, y, z)\frac{dx}{d\eta} \frac{\partial f_1}{\partial y}(x, y, z)\frac{dy}{d\eta} \frac{\partial f_1}{\partial z}(x, y, z)\frac{dz}{d\eta} = 0 \\ \frac{\partial f_2}{\partial x}(x, y, z)\frac{dx}{d\eta} \frac{\partial f_2}{\partial y}(x, y, z)\frac{dy}{d\eta} \frac{\partial f_2}{\partial z}(x, y, z)\frac{dz}{d\eta} = 0 \\ \left(\frac{\partial f_3}{\partial x}(x, y, z)\frac{dx}{d\eta} \frac{\partial f_3}{\partial y}(x, y, z)\frac{dy}{d\eta} \frac{\partial f_3}{\partial z}(x, y, z)\frac{dz}{d\eta} \right) (1 - \eta) \\ \quad + \left(\frac{\partial f_4}{\partial x}(x, y, z)\frac{dx}{d\eta} \frac{\partial f_4}{\partial y}(x, y, z)\frac{dy}{d\eta} \frac{\partial f_4}{\partial z}(x, y, z)\frac{dz}{d\eta} \right) \eta = 0 \, , \end{cases}$$

and solve the *linear* system of equations for derivatives $\frac{dx}{d\eta}, \frac{dy}{d\eta}, \frac{dz}{d\eta}$ (Implicit Function Theorem).

In 2D modeling, $f_1(x, y, z) = z$, and the remaining surfaces are assumed to be independent of z. In other words, we use curves in place of surfaces.

Dependently upon the selected technique of parametrizations, the curves are classified with different labels that have to be specified in the input file, see [123] for a complete discussion.

Transfinite interpolation for a rectangle. Given a parametrization for edges of a rectangle, we can extend it to the whole rectangle by means of the transfinite interpolation with linear blending functions technique. The concept is very much related to the construction of hierarchical shape functions discussed in the previous chapter. We begin by constructing the *edge bubble functions*. It is convenient to use now a parametrization of edges that is compatible with the reference coordinates,

$$\begin{cases} \eta_1 = \zeta \\ \eta_2 = 0 \end{cases} \quad \begin{cases} \eta_1 = 1 \\ \eta_2 = \zeta \end{cases} \quad \begin{cases} \eta_1 = \zeta \\ \eta_2 = 1 \end{cases} \quad \begin{cases} \eta_1 = 0 \\ \eta_2 = \zeta \, . \end{cases}$$

For each of the four curves constituting the rectangle edges, we express the curve parametrizations $\boldsymbol{x}_{edge1}, \boldsymbol{x}_{edge2}, \boldsymbol{x}_{edge3}, \boldsymbol{x}_{edge4}$, in terms of the local coordinate ζ. If the local and global orientations are consistent, we substitute $\zeta = \eta$, otherwise we set $\zeta = 1 - \eta$. Given the parametrizations of the four egdes in terms of ζ coordinate, we define now the edge buble functions:

First edge:

$$\Delta \boldsymbol{x}_{edge1}(\zeta) = \boldsymbol{x}_{edge1}(\zeta) - (\boldsymbol{x}_1(1-\zeta) + \boldsymbol{x}_2\zeta) \ ,$$

Second edge:

$$\Delta \boldsymbol{x}_{edge2}(\zeta) = \boldsymbol{x}_{edge2}(\zeta) - (\boldsymbol{x}_2(1-\zeta) + \boldsymbol{x}_3\zeta) \ ,$$

Third edge:

$$\Delta \boldsymbol{x}_{edge3}(\zeta) = \boldsymbol{x}_{edge3}(\zeta) - (\boldsymbol{x}_4(1-\zeta) + \boldsymbol{x}_3\zeta) \ ,$$

Fourth edge:

$$\Delta \boldsymbol{x}_{edge4}(\zeta) = \boldsymbol{x}_{edge4}(\zeta) - (\boldsymbol{x}_1(1-\zeta) + \boldsymbol{x}_4\zeta) \ .$$

Here $\boldsymbol{x}_j,\ j = 1,\ldots,4$ denote the vertices of the rectangle. The edge bubble functions are thus obtained by subtracting from the original parametrizations the parametrizations corresponding to linear segments. If the rectangle has straight edges, the edge buble functions will be zero.

The transfinite interpolation [74] is now constructed using the formula,

$$\begin{aligned}\boldsymbol{x}(\eta_1,\eta_2) \quad &= \underbrace{\boldsymbol{x}_1(1-\eta_1)(1-\eta_2) + \boldsymbol{x}_2\eta_1(1-\eta_2) + \boldsymbol{x}_3\eta_1)\eta_2 + \boldsymbol{x}_1(1-\eta_1)\eta_2}_{\text{bilinear contribution}} \\ &+ \underbrace{\Delta\boldsymbol{x}_{edge1}(\eta_1)(1-\eta_2) + \Delta\boldsymbol{x}_{edge2}(\eta_2)\eta_1 + \Delta\boldsymbol{x}_{edge3}(\eta_1)\eta_2 + \Delta\boldsymbol{x}_{edge4}(\eta_2)(1-\eta_1)}_{\text{edge contributions}} \ .\end{aligned}$$

By construction, the parametrization for the rectangle is compatible with the prespecified parametrizations of its edges. By differentiating the formula above, we can also compute the derivatives of the parametrization wrt reference coordinates η_j,

Transfinite interpolation for a triangle. Recall the strategy used for extending 1D bubble shape functions to the whole triangle, to define the edge bubble shape functions. Given a parametrization $\boldsymbol{x}_{edge,j}$ for j-th edge of a triangle in terms of the local coordinate ζ,

$$\left\{\begin{array}{l}\eta_1 = \zeta \\ \eta_2 = 0\end{array}\right. \qquad \left\{\begin{array}{l}\eta_1 = 1-\zeta \\ \eta_2 = \zeta\end{array}\right. \qquad \left\{\begin{array}{l}\eta_1 = 0 \\ \eta_2 = 1-\zeta\ ,\end{array}\right.$$

we define the corresponding *edge kernel function* as,

$$\boldsymbol{x}^{kern}_{edge,j} = \frac{\boldsymbol{x}_{edge,j}(\zeta) - (\boldsymbol{x}_j(1-\zeta) + \boldsymbol{x}_{j+1}\zeta)}{(1-\zeta)\zeta} \ .$$

The transfinite interpolant is now defined as follows,

$$\boldsymbol{x} = \underbrace{\sum_{j=1}^{3} \boldsymbol{x}_j\lambda_j}_{\text{linear contributions}} + \underbrace{\sum_{j=1}^{3} \lambda_j\lambda_{j+1}\boldsymbol{x}^{kern}_{edge,j}\left(\frac{\lambda_{j+1}-\lambda_j+1}{2}\right)}_{\text{edge contributions}} \ .$$

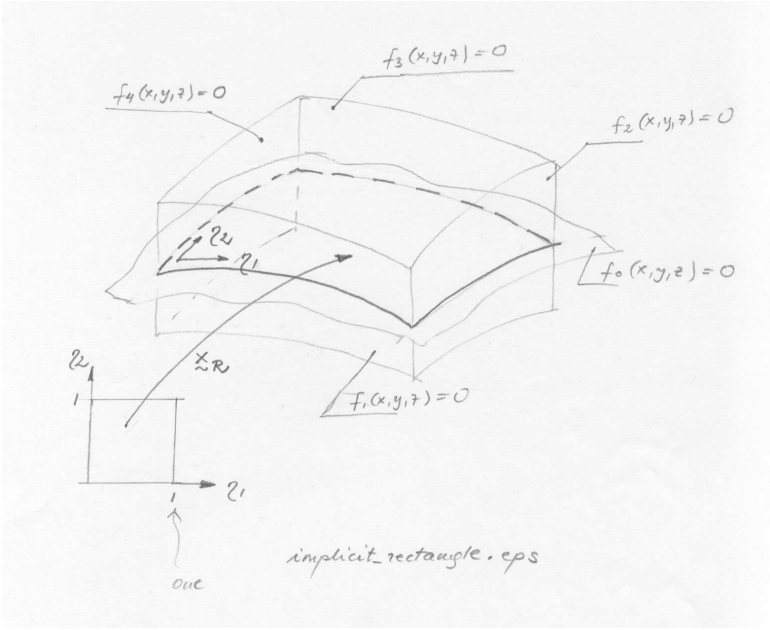


Figure 12.4: Implicit rectangle

Index j is varied modulo 3, $\boldsymbol{x}_j$ denote the vertex coordinates, and λ_j stand for the barycentric coordinates, i.e. the linear shape functions,

$$\lambda_1 = 1 - \eta_1 - \eta_2, \quad \lambda_2 = \eta_1, \quad \lambda_3 = \eta\,.$$

12.3 Implicit parametrization of a rectangle

The transfinite parametrization technique provides means for extending a parametrization of the boundary of a figure into *a parametrization* defined on the whole reference figure. In 3D FE or BE computations, it is important to define a parametrization that would conform to a prespecified surface. In other words, if we have a rectangle that lies on a surface, and we already know a parametrization for its edges, we need to construct such a parametrization that its values would fall on the specified surface. The transfinite interpolation technique does not provide means for such a construction, and it is insufficient.

We assume, comp. Fig. 12.4, that the rectangle is defined in terms of five algebraic surfaces $f_i(x, y, z) = 0, i = 0, \ldots, 4$. The rectangle lies on surface $f_0(x, y, z) = 0$, and the remaining four surfaces cut off the rectangle edges. Constructing *an implicit parametrization* for the rectangle is

easy. We use the same idea as for the implicit curve, and solve the nonlinear system of equations,

$$\begin{cases} f_0(x,y,z) = 0 \\ f_1(x,y,z)(1-\eta_2) + f_3(x,y,z)\eta_2 = 0 \\ f_4(x,y,z)(1-\eta_1) + f_2(x,y,z)\eta_1 = 0\,. \end{cases}$$

The interpretation is straightforward. The second equation defines an intermediate surface between $f_1 = 0$ and $f_3 = 0$, whose location depends upon parameter η_2. For $\eta_2 = 0, 1$, the equation reduces to $f_1 = 0$ $f_3 = 0$, respectively. The last equation is subjected to the same interpretation. Point $\boldsymbol{x}$, whose location depends upon parameters η_1, η_2, is the intersection of the three surfaces.

The trouble with the definition above is that it nowhere incorporates the existing parametrization for the edges of the rectangle. Consequently, the resulting parametrization is, in general, incompatible with the parametrization of the edges. In order to enforce the compatibility with the edge parametrizations, we introduce four *stretching functions* $s_i(\zeta),\ s_i(0) = 0, s_i(1) = 1, i = 1, \ldots, 4$, for the rectangle edges, and replace the system of equations above, with a more complicated version,

$$\begin{cases} & f_0(x,y,z) = 0 \\ (1-\eta_1) & (f_1(x,y,z)(1-s_4(\eta_2)) + f_3(x,y,z)s_4(\eta_2)) \\ +\eta_1 & (f_1(x,y,z)(1-s_2(\eta_2)) + f_3(x,y,z)s_2(\eta_2)) = 0 \\ (1-\eta_2) & (f_4(x,y,z)(1-s_1(\eta_1)) + f_2(x,y,z)s_1(\eta_1)) \\ +\eta_2 & (f_4(x,y,z)(1-s_3(\eta_1)) + f_2(x,y,z)s_3(\eta_1)) = 0\,. \end{cases}$$

The stretching functions are determined by requesting that the nonlinear system of equations is satisfied for edge parametrization $\boldsymbol{x}_{edge,j}(\zeta)$. For instance, for the first edge, $\eta_1 = \zeta, \eta_2 = 0$, the first and the second equations are satified automatically, but the third equation yields,

$$(f_4(\boldsymbol{x}_{edge,1}(\zeta))(1-s_1(\zeta)) + f_2(\boldsymbol{x}_{edge,1}(\zeta))s_1(\zeta) = 0\,,$$

and can be solved for the first edge stretching function $s_1(\zeta)$. By differentiating the equation, we can also determine the derivative of the stretching function. Proceeding in the same way with the remaining three edges, we determine the next three stretching functions. Knowing the values of the stretching functions and their derivatives, we use the Newton Raphson technique to solve the nonlinear system of equations for the value of the implicitly defined parametrization $\boldsymbol{x}(\boldsymbol{\eta})$. Once the value of the parametrization is known, we can use the Implicit Funtion Theorem to determine derivatives $\partial x_i/\partial \eta_j$.

The discussed technique will not be used in the 2D simulations. We include it for a completness of our discussion, and in anticipation of a possible use of the code for BE calculations. For other implicit parametrizations, including the definition of an implicit triangle, and a very special parametrization for a spherical octant that guarantees full rotational invariance with respect to its vertices, we refer once again to [123].

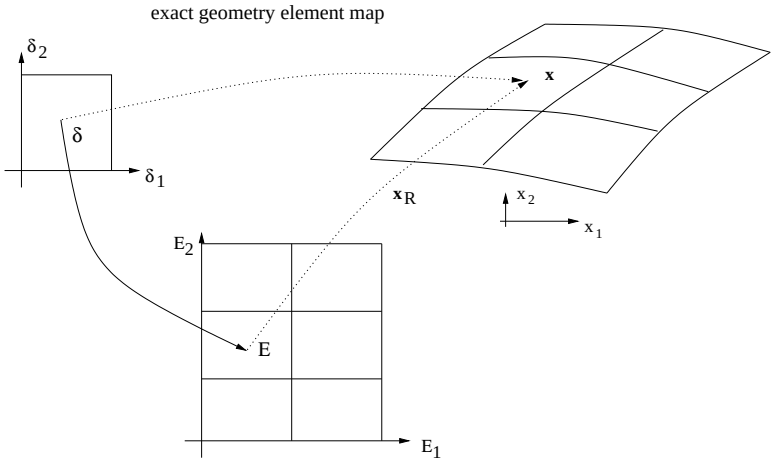


Figure 12.5: Interpolation for geometry d.o.f.

12.4 Initial mesh generation

We are ready to describe now our algebraic mesh generator. The principle is very simple. Each of the reference triangles is covered with a uniform triangular grid of m^2 triangles, and each of the reference rectangles is covered with a uniform grid of $m_1 \times m_2$ rectangles. The numbers of boundary segments m, m_1, m_2 must be compatible with each other. In most of applications, we set them to the same number $m_1 = m_2 = m$ for the whole domain. Besides the number of subdivisions, we specify for each of the reference triangles and rectangles, the corresponding order of approximation. The order may be different for different figures but it must not exceed the maximum order of approximation *NPMAX*. For rectangles, the initial mesh order may be anisotropic, we may choose different orders for the two reference directions. The order for the *curves* is then determined by using the *nimimum rule*.

Once the order for a particular element is known, the corresponding geometry d.o.f. are obtained by interpolating the GMP parametrization maps. The concept is illustrated in Fig. 12.5. For each element in the mesh, we consider the composition of an affine map $\boldsymbol{\xi} \rightarrow \boldsymbol{\eta}(\boldsymbol{\xi})$, mapping the master element onto the corresponding element in the reference figure, with the GMP parametrization map,

$$\boldsymbol{x}(\boldsymbol{\xi}) = \boldsymbol{x}_R(\boldsymbol{\eta}(\boldsymbol{\xi})) \text{ for a rectangle } R, \text{ or } \boldsymbol{x}(\boldsymbol{\xi}) = \boldsymbol{x}_T(\boldsymbol{\eta}(\boldsymbol{\xi})) \text{ for a triangle } T.$$

Notice that GMP provides not only the parametrization $x_i(\eta_j)$ but also the corresponding derivatives $\partial x_i/\partial \eta_j$. Given the map $\xi \rightarrow \boldsymbol{x}(\boldsymbol{\xi})$, we interpolate it, one component at the time, using the projection-based interpolation described in Section 10.8.

A natural question arises: given all the work done on geometry modeling, why don't we use directly the map $\boldsymbol{x}(\boldsymbol{\xi})$ for element calculations ? The concept, known as the *exact geometry element*, was introduced by Barna Szabo, and it is used in e.g. STRESS CHECK and PROBE. The exact

geometry elements are especially essential in BE calculations [50, 30]. We decide to replace the exact geometry maps with their interpolants for two reasons.

- We leave ourselves the possibility of running the code, without the modeling package. In the event of interfacing the code with a professional CAD program, we leave ourselves the option of generating the mesh and dispatching the CAD program, when solving the approximate problem. The geometrical modeler and the solver need not to be executed at the same time.
- The isoparametric elements always reproduce linear maps (linearized rigid body motion) whereas the exact geometry elements do not. This property may be essential in interpreting FE results in a preasymptotic range.

And now comes the code. The mesh generation routines are collected in directories *meshgen* and *hp_interp*. The main driver for the algebraic mesh generator is *meshgen/hp2gen*. It starts by reading the information listed at the end of each of the provided input files:

- Parameter *MFRSOL* specifying the length of the work buffer for the frontal solver. This parameter should be as large as you can afford in terms of the available computer memory but not too large, in order to avoid the use of virtual memory and paging.
- Data structure parameters *MAXELIS,MAXVERS,MAXNODS,NREQNS*.
- For each reference triangle:
  - number of subdivisions m and the order of approximation p,
  - boundary condition flags.
- Same info for each reference rectangle:
  - numbers of subdivisions m_1, m_2 and the order of approximation $p_1 * 10 + p2$,
  - boundary condition flags.
- The order in which the figures should be processed when generating the initial mesh.

The boundary condition flags for curves constituting edges of a reference figure are listed in the order of edges, and stored in a single nickname. For instance, the boundary condition flags for a triangle edges: 0,2,1 are listed as (from right to left !) 120. All element edges falling into the interior of a reference figure are assigned zero boundary condition flags.

The order in which the reference triangles and rectangles are listed, will affect the ultimate order of elements, and the band width of the stiffness matrix. For each reference figure, the elements are denumerated row-wise (left to right), in bottom to top direction. The ordering depends upon the way the reference coordinates have been set up, and it should be anticipated when preparing the

input file for GMP. For instance, if we deal with an elongated rectangle, we make sure that the short edge is listed first. For simple topologies, it is easy to prepare and modify the input file by hand. For complex topologies we usually write a separate program to prepare the input file, or develop special interfaces with existing unstructured mesh generators [122]. In such a case, one should keep in mind the anticipated order of elements, whe preparing the input file.

Upon reading and storing the initial mesh data, *meshgen/hp2gen* allocates dynamically the data structure arrays and begins the actual mesh generation. The routine does not perform the interpolation of GMP maps discussed above. It only reserves the necessary entries in the data structrure arrays, allocates the memory for both geometry and solution d.o.f., and initiates them with zeros.

We shall not discuss the routine in a great detail assuming that, by now, the reader is able to read our code. We only mention a few important points.

- The number of subdivisions for each reference figure is stored permamently in arrays *NSUBTR, NSUBRE* in *commons/cnsub.blk*. These numbers will be needed during geometry updates accompanying mesh refinements.

- For each element in the initial mesh, we store a single nickname encoding the reference figure number, and the local number of the element within the figure. For instance, for the 4th rectangle in 3rd row of 5th reference rectangle, that has been partitioned into 7×8 elements, the local element number is $(3-1)*7+4 = 18$ and the stored nickname is

  $$\textit{ELEMS(nel)\%geom_interf} = -(5 * 10000 + 18) \, .$$

  As you can see, we do not anticipate generating more than 10000 quadrilateral elements within one reference rectangle.

- The nodes are generated in the following order.

  Vertex nodes: points, (interiors of) curves, (interiors of) triangles, (interiors of) rectangles.

  Mid-edge nodes: curves, triangles, rectangles,

  Middle nodes: triangles, rectangles.

  For each initial mesh element, we store the corresponding nodes numbers, and for each initial mesh node, we store the number of *one of connected* elements. As for every element in the mesh, we store all its neighbors across the element edges, a local search allows always for determining ell elements connected to a node.

- In the very end of the routine, we generate the info on orientation of the mid-egde nodes. This is done using the standard assumption: the mid-edge nodes are oriented from the vertex with a lower number to the vertex with a bigger number.

- Study in detail routine *meshmods/nodgen* that generates a new node.

The actual interpolation of the GMP maps and initiation of geometry d.o.f. is performed in routine *hp_interp/update_gdof*. Please study routines *hpvert, hpmedg, hpmbas, hpmsid* determining geometry d.o.f. for vertex, mid-edge, mid-triangle and mid-quad nodes. Along with geometry d.o.f., we initiate also solution d.o.f. corresponding to nodes marked with Dirichlet BC flag (=1). This is done be interpolating the Dirichlet data provided by routine *laplace/dirichlet*. For a known exact solution, routine *laplace/dirichlet* calls simply *laplace/exact*; for a real life problem, routine *laplace/dirichlet* must be provided by the user.

Routine *hp_interp/update_gdof* will also be used for geometry updates after mesh refinements, it serves an arbitrary mesh. It is also possible to update only Dirichlet d.o.f. by calling routine *hp_interp/update_DirichletBC*.

Chapter 13

The hp Finite Element Method on h-Refined Meshes

13.1 Introduction. The h-refinements

Dividing a *father* element into smaller element *sons* is known as an h-refinement. The code supports the following h refinements:

- $h4$-refinement of triangles,
- vertical or horizontal $h2$-refinement of quads,
- $h4$-refinement of quads.

The words 'vertical' or 'horizontal' refer to the local coordinates for the quad element. If the element is broken across the first axis, the refinement is vertical, if the element is broken across the second axis, the refinement is horizontal. Throughout the code, we use an array *kref*(2) to indicate the requested kind of refinement for a quad element. If the element is broken across the i-th axis, *kref*(i) = 1, otherwise *kref*(i) = 0. Thus, *kref* = (1,1) indicates the $h4$- refinement, *kref*$= (1,0)$ the vertical $h2$-refinement, and *kref*$= (0,1)$ the horizontal $h2$-refinement. Consequently, quad elements can be refined into four element sons in different ways, by either executing a single $h4$-refinement, or a sequence of $h2$-refinements of the element, and then its sons.

During the h-refinements, new elements are created and some old ones disappear, new nodes are generated and some existing nodes are deleted. Routines that record the corresponding changes in the data structure arrays have been collected in directory $meshmods$. Contrary to the initial mesh, for which all necessary information is stored explicitly, the information for elements that have resulted from h-refinements, will be recovered from the data structure arrays through special algorithms. The corresponding routines have been collected in directories $datstrs$ and $constrs$.

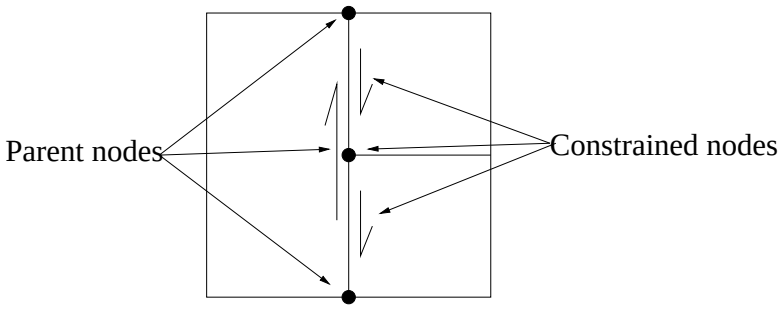


Figure 13.1: Constrained nodes

Perhaps the most profound effect of the h-refinements is the introduction of *constrained (hanging) nodes*. For 2D meshes, we can have constrained vertex or mid-edge nodes, middle nodes are always unconstrained. A typical situation is illustrated in Fig. 13.1. As the result of a refinement, a *big* element, across one of its edges, has two smaller *neighbors*. The vertex node that lies in the interior of the big element edge, and the two mid-edge nodes lying on the small element edges, are constrained nodes. The nodes lying on the big element edge are identified as the *parent nodes* of the constrained nodes. We shall guarantee that parent nodes of constrained nodes are themselves *always* unconstrained, by using a special *1-irregular meshes refinement algorithm* discussed below.

A mesh with constrained nodes is called an *irregular* mesh. Construction of basis functions for irregular meshes is more difficult than for the regular ones. A shape function for a big element cannot be matched with single shape functions from neighboring small elements. Instead, one has to use a linear combination of the small element shape functions. The situation is illustrated for the simplest case of a mesh of bilinear elements in Fig. 13.2.

13.2 1-Irregular Mesh Refinement Algorithm

It may happen that parent nodes of a constrained node are themselves constrained. In such a case the original node is said to be *multiply constrained*. It is desirable for many reasons to avoid multiple constrained nodes and limit ourselves to 1-irregular meshes only. We list a few arguments.

- The code is simpler. Handling multiply constrained nodes adds significantly to the complexity of coding.
- Interpolant of a globally continuous function should also be globally continuous. In presence of constrained nodes, this implies that we loose the local character of the projection-based interpolation. If an element is adjacent to a 'big element', and one of its edges is constrained by a 'big edge', the d.o.f. for the big edge have to be determined first. Next, the h-extension operator (constrained approximation) technique has to be used to determine the small edge d.o.f. This implies that the global interpolation has to be done in a proper order: unconstrained

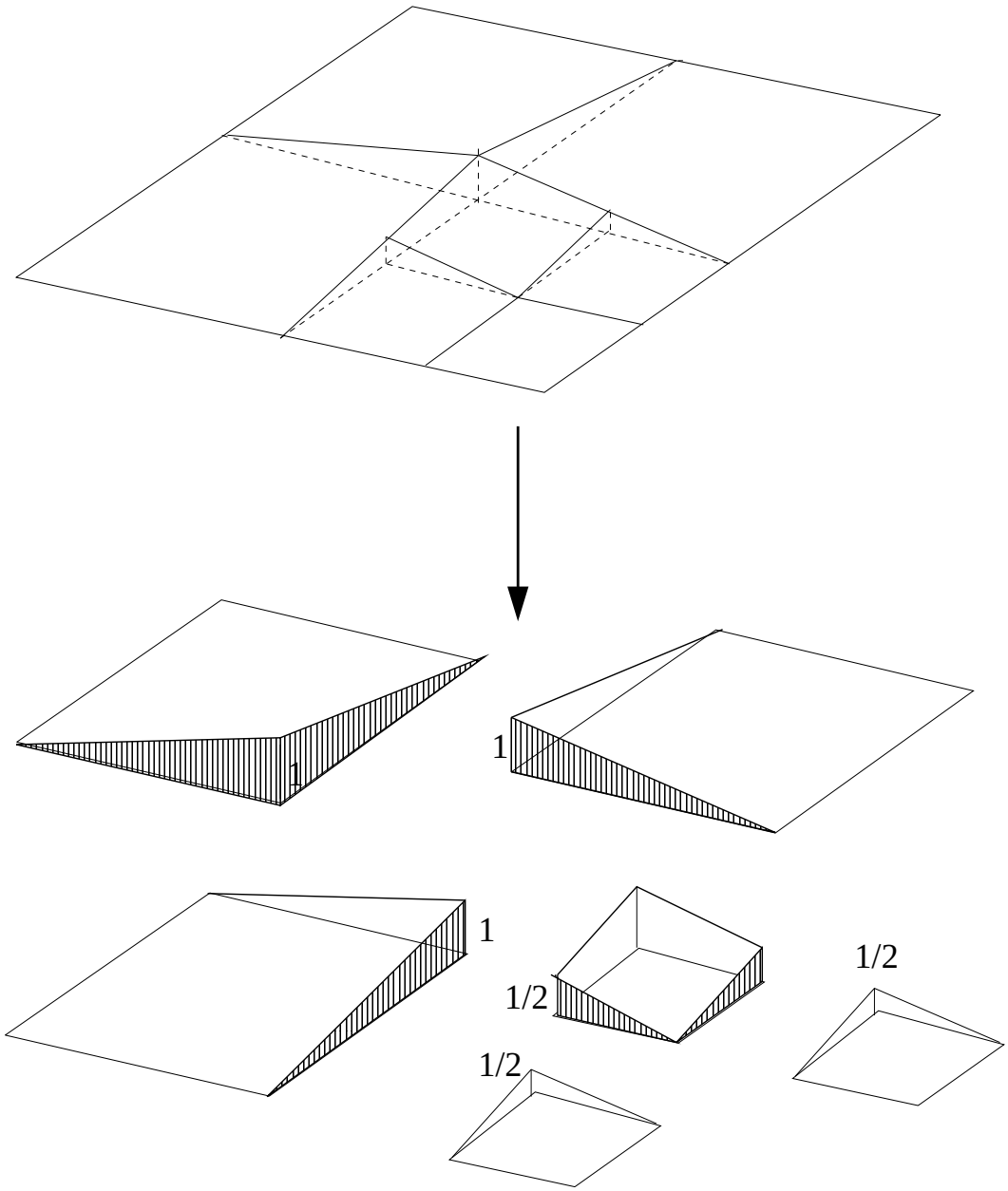


Figure 13.2: Construction of basis functions for an irregular mesh

vertices, edges with unconstrained vertices, constrained vertex nodes, edges with constrained vertex nodes, constrained edges and, finally, (always unconstrained) middle nodes. This is exactly the order in which we update the geometry d.o.f. in *hp_interp/update_gdof*. The algorithm gets much more complicated for multiply constrained nodes.

- For the interpolation procedure described above, the de Rham diagram discussed in the last part of this monograph, does not commute. In order to enable commutativity, one has to replace projections over constrained elements with projections over groups of elements. The procedure gets really out of hand for multiply constrained nodes.

In order to avoid the multiply constrained nodes, we prohibit any h-refinement of an element, unless *all* its nodes are unconstrained. The following algorithm is a generalization of an original algorithm of Rheinboldt et al. for $h4$-refinements of one-irregular rectangular meshes [109]. The procedure uses the standard idea of a "shelf" (a waiting list).

```
Arguments in:
      an element number mdle and requested kind of refinement kref

      put element mdle and requested refinement kref on the shelf
10    stop if the shelf is empty
      pick the last mdle and kref from the shelf
      for each of element mdle mid-edge nodes
        if the node is constrained then
          put mdle and kref back on the shelf
          determine father mid-edge node medg of the constrained
          mid-edge node
          determine the big element neighbor mdlen of medg
          determine refinement flag krefn for the big neighbor
            dependently upon its isotropy flag
          add mdlen and krefn to the shelf
          go to 10
        elseif kref does not call for breaking the edge (valid for
        quads only)
          determine the neighbors of the mid-edge node across
          the element edge
          if there are two small neighbors then
            set kref=(1,1) and reshelve mdle and kref
            go to 10
          endif
        endif
```

```
end of loop through the mid-edge nodes
for each of element mdle vertex nodes
  if the node is constrained then
    put mdle and kref back on the shelf
    determine father mid-edge node medg of the constrained
    vertex node
    determine the big element neighbor mdlen of medg
    add mdlen and kref= (1,1) to the shelf
    go to 10
  endif
end of loop through the vertex nodes
break the element according to the info in kref
go to 10
```

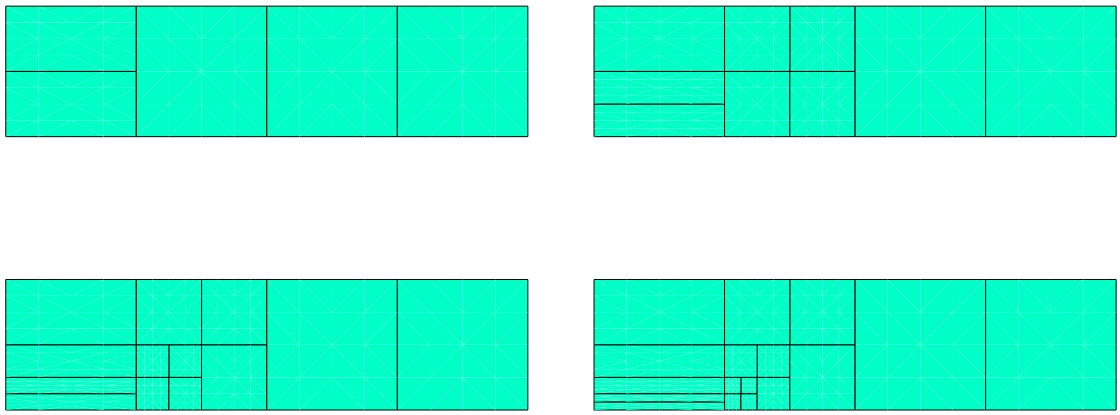


Figure 13.3: Terminating a 'boundary layer' with $h4$-refinements

A number of comments:

1. As mentioned earlier, elements in the code are identified with their middle node numbers. Thus, whenever we refer to an element number, we mean its middle node number.

2. The algorithm eliminates constrained nodes by enforcing refinements of 'big' neighbors. A minimum refinement to eliminate the constrained node is anisotropic. Nevertheless, the neighbor is refined according to its predefined isotropy flag. If the flag calls for the $h4$-refinement or the $h2$-refinement *complementary* to the minimum $h2$-refinement necessary to

eliminate the constrained node, the neighbor is broken in both directions. The default value of the isotropy flag, preset during the node generations, corresponds to the isotropic $h4$-refinement, and it is modified later by the programs enabling automatic h- or hp-refinements. The preference given to isotropic refinements reflects the fact that we anticipate using the anisotropic refinements only in special cases like boundary layers. Enforcing $h4$-refinements of the 'big' neighbors allows then for a nice termination of the refinement, contrary to enforcing only $h2$-refinements of the neighbors. The situation is illustrated in Figures 13.3,13.4. We begin with a mesh of four elements and we continue refining the (current) element in the left lower corner in the horizontal direction. In Fig. 13.3 we enforce $h4$-refinement of the big neighbor, in Fig. 13.4, only the minimal $h2$-refinement, necessary for eliminating the constrained node. The first strategy results in a nice termination of the 'boundary layer', the second causes the refinements to spread.

3. We modify an anisotropic refinement of a quad element to fit refinements of neighbors. This eliminates incidental anisotropic refinements which are intended only to capture solutions that depend only upon one of the two element local coordinates (e.g. in the case of boundary layers and meshes aligned with the boundary).

4. For quadrilateral meshes, eliminating constrained mid-edge nodes implies the elimination of constrained vertex nodes, for triangular meshes, it does not. Fig. 13.5 shows a simple triangular mesh, where the element in the middle has no unconstrained mid-edge nodes, but all its vertex nodes are constrained.

5. The algorithm requires the use of the following information.

   - Given an element (its middle node), return its mid-edge and vertex nodes. In the case of a constrained node, return its parent nodes. This is a standard *nodal connectivities* information.
   - Given a mid-edge node, return its neighbors.

   Note that the algorithm does not require determining (at least directly) element neighbors for a given element.

13.3 Data structure in Fortran 90 - continued.

The genealogical information for nodes. During h-refinement, the content of the data structure must be updated. Contrary to previous implementations, the new data structure records only a very minimal information, and only for nodes. *No information for the new elements, resulting from h-refinements is stored whatsoever !*. The information stored for nodes supports the genealogical information (trees). When a *father* node is refined (broken), we store the information about its *nodal*

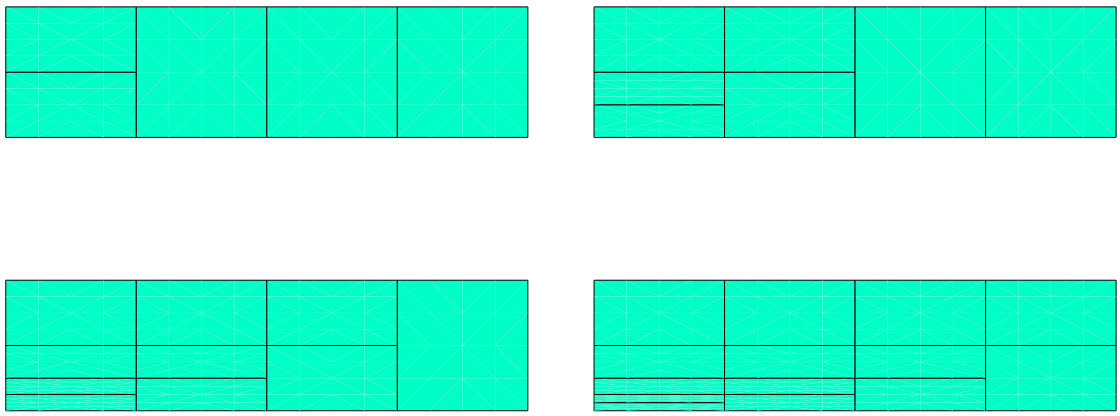


Figure 13.4: Terminating a 'boundary layer' with $h2$-refinements

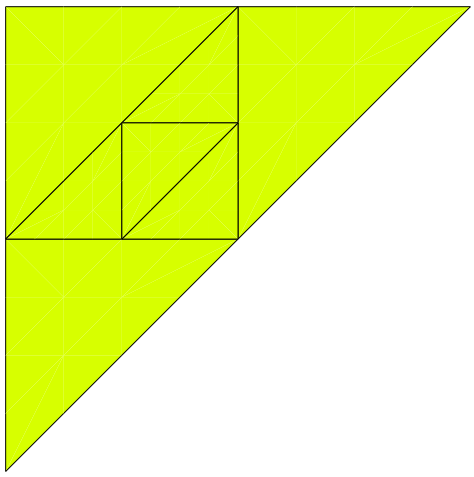


Figure 13.5: Example of a triangle with unconstrained edges and constrained vertices

sons. A mid-edge node has three sons: two mid-edge nodes and one vertex node. An ($h4$-)refined middle node for a triangle has seven sons: four middle nodes, and three mid-edge nodes. An $h2$-refined middle node of a quad has three sons: two middle nodes, and one mid-edge node. Finally, an $h4$-refined middle node for a quad, has nine sons: four middle nodes, four mid-edge nodes, and one vertex node. The numbering of the sons and the orientation of the new mid-edge nodes is

depicted in Fig. 13.6. The numbers of the new nodes are stored in array *NODES(nod)%sons*, whose length depends on the type of the node and the refinement kind and, for that reason, the array is allocated dynamically. The executed *refinement kind* is stored in *NODES(nod)%ref_kind*. Notice that vertex nodes are never refined and, therefore, this information is relevant only for the non-vertex nodes. For the newly generated nodes we store their father in either *NVERS(nod)%father* or *NODES(nod)%father*. Finally, *NODES(nod)%connect* is a pointer to an entry in an extra, temporary array storing the *extended parent information* for the new nodes. When a mid-edge node is refined, the extended parents include its vertex nodes, for a middle node, it includes element mid-edge and vertex nodes. This extra information is necessary when constructing *extension operators* for multigrid operations. We shall return to this subject in version 2.1 of this code.

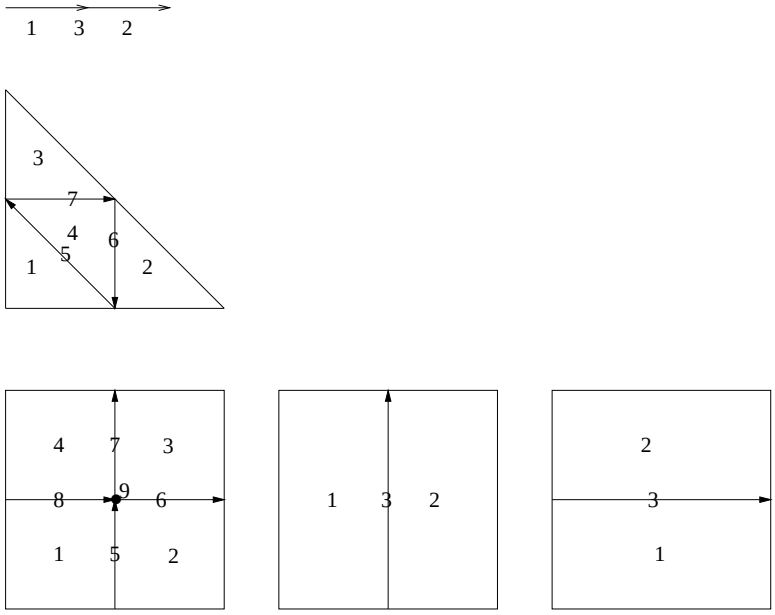


Figure 13.6: Numbering of sons for broken father mid-edge, triangle and quad middle nodes

The natural order of elements As elements are identified through their middle nodes, the genealogical information for the middle nodes includes the genealogical information for elements. This supports the so called *natural ordering of elements*. The ordering follows the numbering of elements in the initial mesh (introduced by the mesh generator) and the tree structure of elements following the *leaves* of the tree (*active* elements). Figure 13.7 shows a typical binary tree and the resulting natural order of elements.

The algorithm is executed by routine $/datstrs/nelcon$. A typical loop through all *active* elements in the mesh then looks as follows

mdle = 0

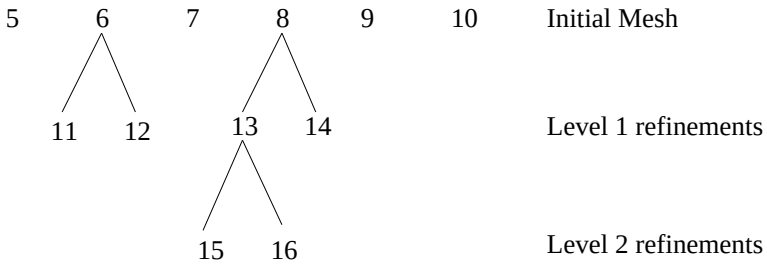


Figure 13.7: The natural order of elements.

```
do iel = 1, NRELES
        call nelcon(mdle, mdle)
        ⋮
enddo
```

13.4 Constrained approximation for C^0 discretizations

As illustrated in Fig 13.2, the construction of basis functions for irregular meshes is more complicated than that for regular ones discussed in the previous section. In general, restriction of a basis function to a contributing element does not reduce to (plus/minus) the element shape function. Instead, we have to assume that the basis function must be represented as a linear combination of the element shape functions:

$$e_i|_K = \sum_k c_{ik,K}\phi_{k,K}. \tag{13.4.1}$$

The calculation of the global stiffness is now more complicated that for regular meshes (comp. (**??**)):

$$S_{ij} = b(e_i, e_j) = \int_\Omega \boldsymbol{\nabla} e_i \boldsymbol{\nabla} e_j = \sum_K \sum_k \sum_l c_{ik,K} c_{jl,K} b_K(\psi_k, \psi_l) \tag{13.4.2}$$

where the element stiffness matrix (**??**) is the same as for regular meshes. Similarly, the global load vector is evaluated using the strategy:

$$L_i = l(e_i) = \int_\Omega f e_i \, d\boldsymbol{x} + \int_{\Gamma_2} g e_i \, ds = \sum_K \sum_k c_{ik,K} l_K(\psi_k) \tag{13.4.3}$$

with the element load vector defined by (**??**). The difference between (**??**) and (13.4.2), and (**??**) and (13.4.3) lies in the extra summations over the element shape shape functions and the connectivity

coefficients $c_{ik,K}$. In practice, we sum up only over those indices for which the coefficients are non-zero. The calculations for regular meshes are then just a special case of those for irregular meshes with the summation extending over one index only and the connectivity coefficient reduced to the sign factor.

The actual algorithm of assembling the global matrices is not done from the point of the view of the receiver (the global matrices) but contributor (the element matrices) and it looks as follows.

initiate global matrices S_{ij} and L_i with zeros,
for each element K in the mesh,
  calculate element stiffness matrix $S_{kl,K}$ and load vector $L_{k,K}$
  for each local d.o.f. k
    for each connected global d.o.f. i
      determine connectivity $i = i(k, K)$ and the connectivity coefficient $c_{ik,K}$
      accumulate for the global load vector:
      $L_i = L_i + c_{ik,K} * L_{k,K}$
      for each local d.o.f. l
        for each connected global d.o.f. j
          determine connectivity $j = j(l, K)$ and coefficient $c_{jl,K}$
          accumulate for the global stiffness matrix:
          $S_{ij} = S_{ij} + c_{ik,K} * c_{jl,K} * S_{kl,K}$
        end of the second loop through the connected global d.o.f.
      end of the second loop through local d.o.f.
    end of the first loop through the connected global d.o.f.
  end of the first loop through the local d.o.f.
end of loop through elements

Modified element. Even more than for regular meshes, it is convenient to separate the extra summation over the connected d.o.f. from the classical assembling procedure by introducing the definition of the *modified element*. The information about the modified element is identical with that for regular meshes and it includes:

- a list of element vertex nodes: $Nod1(i), i = 1, \ldots, Nvert$,
- a list of element non-vertex nodes: $Nod2(i), i = 1, \ldots, Nrnod$,
- the corresponding number of shape functions (d.o.f) per node: $Ndof2(i), i = 1, \ldots, Nrnod$,
- the modified element load vector: $Bload(k), k = 1, \ldots, Nrdof$,
- the modified element stiffness matrix: $Astiff(k, l),\ k, l = 1, \ldots, Nrdof$.

The list of vertex nodes $Nod1$ contains numbers of all vertex nodes connected to element vertices. This includes the element unconstrained vertex nodes and parent vertex nodes of the constrained vertices. Notice that the two sets (unconstrained nodes and parents of constrained nodes) need not be disjoint. Similarly, the list of non-vertex nodes $Nod2$ contains both the element unconstrained mid-edge and middle nodes and parent mid-edge nodes of element constrained nodes. The interpretation of the other quantities is the same as for regular meshes. Once the lists of the modified vertex and non-vertex nodes are created, the corresponding modified element d.o.f. are numbered following the order of nodes on the lists and then the order of d.o.f. corresponding to each of the nodes. Next, connectivities of the local d.o.f. k to the modified element d.o.f. $NAC(j,k), j = 1,\ldots,NRCON(k)$ and the corresponding connectivity coefficients $CONSTR(j,k), j = 1,\ldots,NRCON(k)$ are determined. The calculation of the modified element matrices is then done using the same algorithm as above:

initiate $Bload$ and $Astiff$ with zeros
for each local d.o.f. $k2$
  for each connected modified element d.o.f. $j2 = 1,\ldots,NRCON(k2)$
    $i2 = NAC(j2,k2)$
    accumulate for the modified load vector:
    $Bload(i2) = Bload(i2) + CONSTR(j2,k2) * Bloc(k2)$
    for each shape function $k1$
      for each connected modified element d.o.f. $j1 = 1,\ldots,NRCON(k1)$
        $i1 = NAC(j1,k1)$
        accumulate for the modified stiffness matrix:
        $Astiff(i1,i2) = Astiff(i1,i2)$
        $+CONSTR(j1,k1) * CONSTR(j2,k2) * Aloc(k1,k2)$
      end of the second loop through connected d.o.f.
    end of the second loop through the element d.o.f.
  end of the first loop through connected d.o.f.
end of the first loop through the element d.o.f.

Here $Bloc, Aloc$ denote the element (local) load vector and stiffness matrix.

At this point it should become clear that the whole difficulty in implementing the constrained approximation is in constructing the arrays $NRCON, NAC, CONSTR$. The implementation presented in this document is already the fourth iteration that the first author has been involved with. In our first code [54], the entire information about the constraints was reconstructed from information stored for *active elements* ONLY. In the second implementation [48] we went to other extreme and stored the information about the constraints explicitly in the data structure arrays. This was very convenient for the constrained approximation but had made the mesh modification routines very

complicated. In the last implementation [61], we reconstructed the information about the constraints for an element from the nodal connectivities stored for the elements and its father (and possibly grandfather). In the presented (ultimate?) implementation, arrays $NRCON, NAC, CONSTR$ are constructed from a local data base on constraints generated during the reconstruction of element nodal connectivities. This combines the simplicity of implementation in [48] with no need for storing directly any information for constrained nodes. The algorithm described next is absolutely crucial for this new implementation.

13.5 Reconstructing element nodal connectivities.

Instead of presenting a formal algorithm, we shall illustrate it with an example presented in Fig. 13.8. The first picture represents a mesh that has resulted from a sequence of $h4$-refinements of an initial mesh of just four elements. Suppose we want to find the nodes of the element marked in red in the first picture. The corresponding denumeration of elements, i.e. their middle nodes numbers are shown in the second picture. We are interested in the element 115. We begin by ascending the tree of elements up [1] all the way to an initial mesh element, in this case the left corner element 13 of the initial mesh shown in the third picture. When ascending the tree, we remember the way back to the original element. For the initial mesh ancestor, the corresponding nodes are stored in data structure array *ELEMS(nel)%nodes* where nel is the element number of the initial mesh ancestor (using consecutive numbering, in this case $nel = 2$). Given the nodes of the initial mesh element, we descend back to the original element, reconstructing the nodes of the ancestors from the nodal trees. The fourth picture shows the reconstruction of nodes for element 19. We then turn to element 59, and reconstruct nodes for its element sons, shown in the fifth picture.

Only in the last stage, shown in the sixth picture, we encounter nodes (of the original element father) that have not been refined. For those nodes, we construct the necessary information about the corresponding constraints and store it a local data base in modul *module/tree*. The negative numbers for the element nodes indicate constrained nodes and serve as pointers to a local data base for the constrained nodes.

The reconstruction is done in routine *constrs/history*. Besides the information on constraints, the reconstructed information stored in modul *module/tree* contains not only the element nodes, but also their order (relevant for non vertex nodes), their orientation (relevant for mid-edge nodes) and the element refinement history, including the initial mesh ancestor and the refinement path. With a call made to routine *history*, the content of modul *module/tree* is first checked, and the routine is executed only if the information stored there does not correspond to the element on the input. That way we avoid an unnecessary multiple execution of the algorithm. A necessary, precomputed logical information used by routine *constrs/history*, is stored in module *module/refinements* and

[1] Computer trees grow downward...

module/edges.

Definition of the modified element. The lists of nodes for the modified elements and the corresponding constrained approximation coefficients are determined by routine *constrs/logic*, which calls specialized versions *constrs/logict* for triangular, and *constrs/logicq* for quad elements. The lists of nodes for the modified elements are constructed in auxiliary routines *constrs/logict_nodes* and *constrs/logicq_nodes*, respectively. For the example discussed above, and element 115, the corresponding vertex nodes are determined as follows

- regular vertex node 34,
- vertex parent nodes 25,5 of the second, constrained element vertex node marked with -13,
- the third, regular vertex node 5 is already on the list,
- vertex parent node 30 of the fourth, constrained vertex node, marked with -23; the first parent vertex node 5, is already on the list.

The list of non-vertex nodes for the modified element is assembled simultaneously with the list of vertex nodes. After the loop through the vertex nodes, the list contains already mid-edge parent node 90 (of the second vertex node), and mid-edge parent node 100 of the fourth vertex node. We proceed now by looping through mid-edge nodes, adding to the list of non-vertex nodes of the modified element:

- regular (first) mid-edge node 118,
- regular (fourth) mid-edge node 119.

Finally, we add to the list, (always regular) middle node 115. Once the lists of the modified element nodes have been determined, we determine the corresponding constrained approximation coefficients using the information in the local data base for the element constrained nodes. This is done in routine *constrs/logic2*.

We encourage the user to study the code to get a better idea how these things are organized. For a k-th d.o.f that belongs to an unconstrained node, the constrained approximation matrices reduce to those discussed in the previous section, i.e. $NRCON(k) = 1$, $NAC(1,k)$ contains the corresponding location of the d.o.f. on the list of the modified element d.o.f. (with the order induced by the order of its nodes), and $CONSTR(1,k)$ equals the sign factor, depending upon the orientation.

13.6 Determining neighbors for mid-edge nodes

Given a mid-edge node, we set to determine its element (middle nodes) neighbors on both sides. The neighbor information is used in the 1-irregular meshes refinement algorithm, in routines performing refinements, in graphics and (not presented here) in a-posteriori error estimation. It is an important piece of information about the mesh.

Routine $datstrs/find_medg_neig$ determines the *youngest equal size* neighbors for a mid-edge node that exists in the data structure array, active or not. If the edge is on the boundary, one neighbor is returned, for internal edges we get two. The overall strategy is very similar to that for determining element connectivities.

Step 1: Go up the tree of refinements for the mid-edge node until you reach a middle node or an initial mesh mid-edge node. A mid-edge node must have a father mid-edge node, a father middle node, or belong to the initial mesh. In the father is another mid-edge node, we replace the node with its father, and store the information about the path (first or second son, going left or right). If the father of the currently processed mid-edge node is a middle node, we know its neighbors from the way we denumerate sons of a middle node, comp. Fig. 13.6. For instance, for an $h11$ refined quad middle node, neighbors of the first edge-son (son 5 altogether) are sons number 1 and 2. If the edge belongs to the initial mesh, we store in place of its father a link to one of the neighboring initial mesh elements. The information about the element neighbors is then used to recover the second neighbor for the edge. In the end of the first step, we know the path to the mid-edge ancestor and the middle node neighbors (one or two) of the ancestor.

Step 2: For each of the middle node neighbors of the ancestor edge, descend the refinement tree for the middle node, following the stored path information, until you reach the youngest element, adjacent to the original mid-edge node, that occupies the whole edge, comp. routine $datstrs/descend$.

In summary, we ascend the mid-edge nodes trees, jump to the middle nodes neighbors, and descend the middle nodes trees.

Routine $datstrs/neig_edge$ determines neighbors for an *active mid-edge node*. There may be one big or two small neighbors on each of the mid-edge node sides. The routine starts by determining the equal size neighbors, and then descends further the neighbor middle nodes trees until it reaches unrefined middle nodes (elements) adjacent to the edge.

13.7 Additional comments

Assembling of global matrices. Interfacing with solvers. The global assembling procedure is identical with that for regular meshes discussed in the previous section. Once the modified elements matrices are calculated, the regular and irregular meshes are treated in an identical way. This is especially convenient when it comes to writing an interface with a linear equations solver. In particular, the interface with the frontal solver discussed in the last section remains exactly the same. In principle, the constrained approximation has been entirely hidden from the user in the constrained approximation routines. The user is free to modify the element routine (in particular to accommodate different boundary-value problems) and the constrained approximation routines will automatically produce the modified element arrays allowing then again for a standard interface with a solver.

Evaluation of local d.o.f. Routines *constrs/nodcor* and *constrs/solelm* return for an element *local* geometry and actual d.o.f. necessary to evaluate the geometry or the solution (e.g. for postprocessing). The evaluation is based again on the constrained coefficients stored in arrays $NRCON, NAC$ and $CONSTR$. First, the corresponding modified element d.o.f. are copied from the data structure arrays into a local vector $dofmod$ and then the local d.o.f. vector dof is evaluated using the algorithm:

initiate dof with zeros
for each local d.o.f. k
    for each connected modified element d.o.f. $j = 1, \ldots, NRCON(k)$
      $i = NAC(j, k)$
      accumulate for the local d.o.f.:
      $dof(k) = dof(k) + CONSTR(j, k) * dofmod(i)$
    end of loop through connected d.o.f.
end of loop through the element d.o.f.

In this way, the conversion of the global d.o.f. to the local ones is again hidden from the user who needs only to know how to call the two routines.

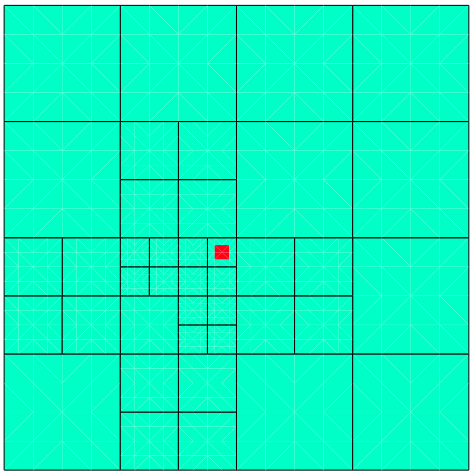
p-refinements. The p-refinement for an element is executed in routine $meshmods/modord$. Given new orders for an element nodes, the routine modifies the nodes initiating new d.o.f. and performing the necessary changes in data structure arary $NODES$. The new d.o.f. are initiated using the old d.o.f. by complementing them with zeros (if the polynomial order increases) or by simply deleting the redundant d.o.f. [2]. In the case of a constrained mid-edge node, we identify

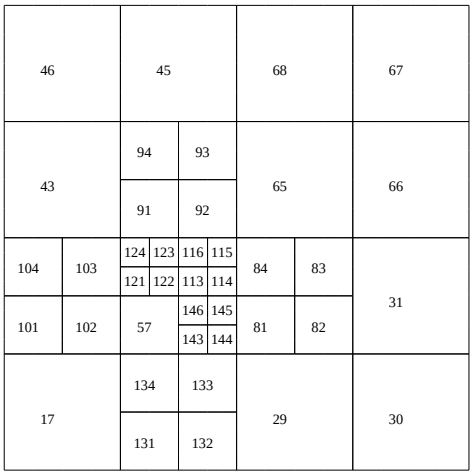
[2] This ought to be replaced with a projection-based interpolation...

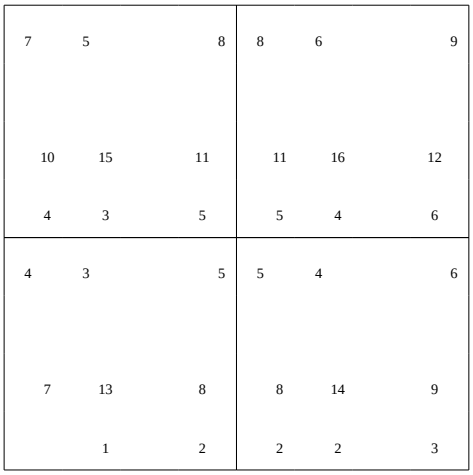
and modify its father. The *minimum rule* is enforced, i.e. the order for the egdes never exceeds orders of the neighboring middle nodes. The actual modification of the nodes is done in routine $meshmods/nodmod$.

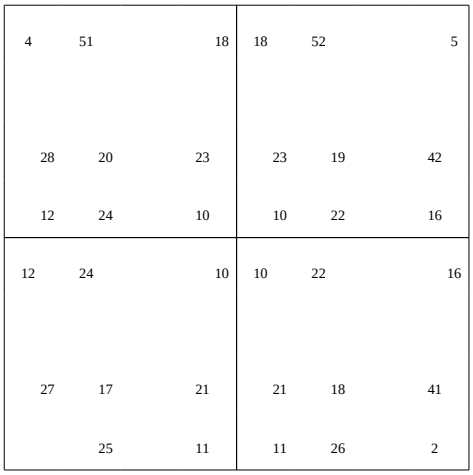
Finally, routine $meshmods/enrich$ requires on input only the order of approximation for the element middle node, enforcing the same order for the element edges.

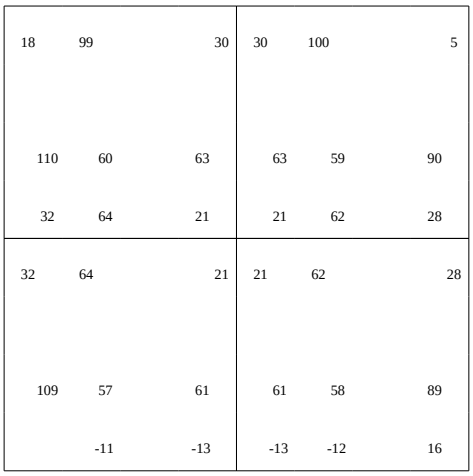
We invite again the user to play with the interactive routine for mesh modifications in order to verify the discussed rules.











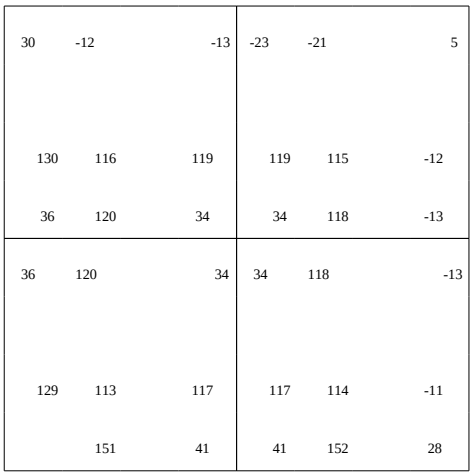


Figure 13.8: Reconstructing nodal connectivities for an element

Chapter 14

Automatic hp-Adaptivity in 2D

14.1 The main idea

We shall present only the main steps of the algorithm, coded in routines collected in directory $adapt$. For a detailed discussion, we refer to [58].

The main idea is based on minimizing a projection-based interpolation error, or the hp-interpolation error, as we have called it. There are two main points.

- The projection-based interpolation delivers optimal h- convergence, and *almost optimal* [1] p-convergence rates [43]. Thus, minimizing the interpolation error should deliver almost optimal meshes.
- Contrary to the actual approximation error, global in its nature, the interpolation error is determined *locally*, over one element at a time. This makes the minimization algorithm feasible.

Now, we need something to interpolate. In place of the unknown exact solution, the algorithm uses the solution (or its approximation) on a *globally hp-refined grid*. For practical, large problems, the global hp-refinement produces a huge mesh and the use of a direct solver on such a mesh is prohibitively expensive. The solution is obtained then using a two-grid or a multi-grid solver. We do not include the two-grid solver in this version of the code and, for now, the code is using the frontal solver only. We hope to deliver the two-grid solver in a forthcoming Version 2.1 of the code. Additionally, our experiments indicate that the solution on the fine grid may be replaced with a couple of smoothing operations only.

In summary, the mesh optimization algorithm produces a sequence of *coarse meshes* and an accompanying sequence of *fine meshes*. The error of the coarse grid solution is estimated simply by evaluating (the norm of) the difference between the fine and coarse mesh solutions.

[1] Up to a logarithmic factor

We shall proceed now with a short discussion of a typical step of the algorithm performed in routine $adapt/hp_strategy$.

14.2 Discussion of one step of the algorithm

Suppose now that we are given an existing FE *coarse* hp mesh. We begin by performing a global hp-refinement, i.e. we split every element into four element sons, and we raise the order of approximation by one, uniformly throughout the whole mesh. This gives us the *fine mesh*. We solve then the problem of interest on the fine mesh.

The main idea is now as follows. Given the fine mesh solution, we determine optimal refinement of the coarse grid, by minimizing the hp-interpolation error for the fine mesh solution interpolated on the next, optimal coarse mesh. The hp-interpolation is understood here in a generalized sense explained next.

Step 1: Linear (bilinear) interpolant. We interpolate fine mesh solution $u_{h/2,p+1}$ at coarse grid nodes, and subtract resulting linear (bilinear for quads) interpolant $u^1_{h,p}$ from the fine mesh solution.

Step 2: Optimal refinement of edges. We begin by computing for each edge the corresponding *edge interpolant*, obtained by projecting difference of $u_{h/2,p+1} - u^1_{h,p}$ on the space of edge shape functions $V_{hp}(e)$, vanishing at the edge endpoints,

$$\left\{ \begin{array}{l} w^e_{hp} \in V_{hp}(e) \\ \|w - w^1_{hp} - w^e_{hp}\| \rightarrow \min \ . \end{array} \right. \tag{14.2.1}$$

Ideally, we should use here the $H^{\frac{1}{2}}_{00}(e)$-norm. Instead, in the practical implementation, we have replaced it with a weighted H^1_0 norm,

$$\|u\|^2_e = \int_e \left(\frac{\partial u}{\partial s} \right)^2 \left(\frac{\partial s}{\partial \xi} \right)^{-1} ds = \int_0^1 \left(\frac{\partial u}{\partial \xi} \right)^2 d\xi \ . \tag{14.2.2}$$

Here $\boldsymbol{x} = \boldsymbol{x}(\xi),\ \xi \in (0,1)$ is the FE parametrization for the edge, and

$$\frac{\partial s}{\partial \xi} = \sqrt{\sum_{i=1}^{2} \left(\frac{\partial x_i}{\partial \xi} \right)^2} \ . \tag{14.2.3}$$

Note that this weighted H^1_0 norm scales with the length of the edge the same way as the $H^{\frac{1}{2}}_{00}$ norm.

The projection problem is equivalent to solving a system of equations,

$$\left\{ \begin{array}{l} u^e_{hp} \in V_{hp}(e) \\ (u^e_{hp}, v_{hp})_e = ((u_{h/2,p+1} - u^1_{hp}), v_{hp})_e \quad \forall v_{hp} \in V_{hp}(e) \ , \end{array} \right. \tag{14.2.4}$$

where inner product $(\cdot,\cdot)_e$ corresponds to the norm discussed above.

Having solved the projection problem, we compute the corresponding projection error (squared),

$$\eta_e = \|u_{h/2,p+1} - u^1_{hp} - u^e_{hp}\|^2_e \,. \tag{14.2.5}$$

Next, we compute the corresponding *edge interpolants* corresponding to a p-refined edge, and a sequence of *competitive* h-refinements. By the competitive h refinement we understand here breaking the edge, and assigning such degrees of polynomial shape functions for the sons that the total number of d.o.f. for the h-refined element is the same as for the p-refined element. Thus, if the original order of approximation for the edge is p_e, then the new orders of approximation for the broken edge sons p^1_e, p^2_e, must satisfy the condition

$$p^1_e + p^2_e = p_e + 1 \,. \tag{14.2.6}$$

For example, for a linear edge element, the choice is between the quadratic element and two linear elements, for a quadratic edge, the choice is between the cubic edge and two refinements resulting in quadratic/linear of linear/quadratic approximations for the edge. In general, one has to solve $p_e + 1$ projection problems.

Once the optimal refinements for the preselected edges have been determined, we compute the corresponding projection (interpolation) errors and define the edge error indicators as the rates with which the interpolation errors decrease,

$$\eta_e = \|u_{h/2,p+1} - u^1_{hp} - u^e_{hp}\|^2_e - \|u_{h/2,p+1} - u^1_{hp} - u^e_{opt}\|^2_e \,. \tag{14.2.7}$$

Here u^e_{opt} denotes the edge interpolant corresponding to the optimal refinement of the edge, producing the smallest projection error. We determine next the maximum error $\eta_{max} = \max_e \eta_e$, and identify all edges e for which

$$\eta_e \geq \frac{1}{3}\eta_{max} \,. \tag{14.2.8}$$

These are the edges marked for a refinement. At this point, we know their desired optimal refinement and the corresponding interpolant (projection) u^e_{opt}.

Step 3: Breaking elements. Elements adjacent to edges marked for h-refinement are next h-refined. Triangular elements are just broken into four congruent element sons. In the case of quadrilaterals, we determine first an *isotropy flag* for each element by analyzing the difference

$$u_{h/2,p+1} - u_{hp} \,. \tag{14.2.9}$$

Roughly speaking, anisotropic refinements are selected only if the error function exhibits essentially one-dimensional behavior. This allows for an optimal refinement along boundary layers and edge singularities (3D), see [58] for details.

The 1-irregularity rule for meshes implies some involuntary h refinements of edges. This may apply to edges marked for p-refinement or marked for no refinement at all. We have to revisit those edges and determine optimal orders of approximation for the resulted edge elements. Please see again [58] for details.

In summary, the decision on breaking elements is made exclusively by determining optimal refinements of edges and, for quads, analyzing the possible anisotropy of the error function. If none of element edges is selected for h-refinement, the element may only be p-refined.

Step 4: Determining optimal orders of approximation for refined elements. After the third step we know the topology of the mesh and optimal order of approximation for all element edges that overlap with edges of the coarse mesh. Given this information we set to determine optimal orders of approximation for all involved elements. The algorithm is again local - we consider one element of the coarse grid at the time. If none of the element edges has been changed, we do not change the approximation. If any of the element edges has been modified in any way, the element may have been left without any h-refinement, or it may have been h-refined in one (quads only) or two directions. In either of the cases, we need to determine new order of approximation for the element or its sons. We do it by monitoring the *interpolation error decrease rates*, defined as

$$\eta_K = \frac{|u_{h/2,p+1} - u^1_{hp} - u^2_{opt} - u^3_{hp}|^2_{H^1(K)} - |u_{h/2,p+1} - u^1_{hp} - u^2_{opt} - u^3_{opt}|^2_{H^1(K)}}{\Delta\mathrm{nrdof}} \,. \quad (14.2.10)$$

Here

- $u^2_{opt} = \hat{u}^e_{opt}$ denotes a lift of edge interpolants corresponding to the optimal edge refinements determined in the previous step.

- u^3_{hp} denotes the projection of difference $u_{h/2,p+1} - u^1_{hp} - u^2_{opt}$ onto the space of element shape functions vanishing on element edges, $\mathcal{P}^p_{-1}(K)$,

$$\begin{aligned} & u^3_{hp} \in \mathcal{P}^p_{-1}(K) \\ & |u_{h/2,p+1} - u^1_{hp} - u^2_{opt} - u^3_{hp}|_{H^1(K)} \to \min \,. \end{aligned} \quad (14.2.11)$$

  This is again equivalent to solving a system of equations,

$$\left\{ \begin{array}{l} u^3_{hp} \in \mathcal{P}^p_{-1}(K) \\ \displaystyle\int_K \boldsymbol{\nabla} u^3_{hp} \boldsymbol{\nabla} v_{hp} \, d\boldsymbol{x} = \int_K \boldsymbol{\nabla}(u_{h/2,p+1} - u^1_{hp} - u^2_{opt}) \boldsymbol{\nabla} v_{hp} \, d\boldsymbol{x} \quad \forall v_{hp} \in \mathcal{P}^p_{-1}(K) \,. \end{array} \right. \quad (14.2.12)$$

  Projection u^3_{hp} serves for a reference *element interior interpolant* with respect which the error decrease rate is defined.

- $\Delta\mathrm{nrdof}$ denotes the increase in the number of the element interior d.o.f., compared with the coarse mesh.

Finally, u^3_{opt} is determined by solving again the coarse element interior projection problem with $\mathcal{P}^p_{-1}(K)$ replaced with *space of element bubble functions* $V_{hp}(K)$ defined as the span of all (new coarse grid) basis functions with support in $\bar{K}$. This is done recursively, starting with the order of approximation for the element(s) interior(s) determined by orders for the edges and the *minimum rule*. The projection problem is solved, and local contributions to the projection error analyzed with the order p increased accordingly. [2] We monitor interpolation error decrease rate η_K defined above, and stop the iteration when two conditions are satisfied:

- The error corresponding to the new mesh must not exceed the error corresponding to the existing mesh.
- The rate is greater or equal than a prespecified minimum rate,

$$\eta_K \geq \eta_{min} \, . \tag{14.2.13}$$

In global terms, this is an investment problem. We invest with refinements only in those elements that produce sufficiently good error decrease rates. The minimum 'investment rate' η_{min} is determined by identifying the element for which the edge optimization has resulted in the largest error decrease. We then *presolve* the p-optimization problem for this element, increasing order p all the way to the order of the fine grid, and determining the maximum rate for the element. The minimum rate is then set to 1/3 of the maximum rate for the element.

The stopping criterion. In the end of the fourth step, we have determined the new coarse mesh and we continue the process until a stopping criterion is met. For the stopping criterion we use simply the error between the coarse and fine grid solutions measured in the H^1_0 seminorm, relative to the H^1_0-seminorm of the fine grid solution,

$$\frac{|u_{h/2,p+1} - u_{hp}|_{H^1(\Omega)}}{|u_{h/2,p+1}|_{H^1(\Omega)}} \leq \text{admissible error} \, . \tag{14.2.14}$$

14.3 Example: L-shape domain problem.

As an illustrating example, we present a solution of the classical L-shape domain problem. The Laplace equation was solved with Dirichlet boundary conditions imposed using the exact solution. The initial mesh consisted of four triangles and one quad, all of second order, and the mesh optimization procedure was continued until an error tolerance of 0.001 (.1 percent) was reached. The corresponding optimal hp mesh is shown in Fig. 14.1. Fig. 14.2 presents the corresponding convergence history, compared with the convergence history for just h-adaptivity using quadratic elements

[2] This is really a miniature p-adaptivity problem. Given an approximation on the boundary, determine the best p in the interior...

only. The scale on the horizontal axis corresponds to the optimal, exponential convergence rates predicted by the theory of hp discretizations [113], and confirmed by the algorithm.

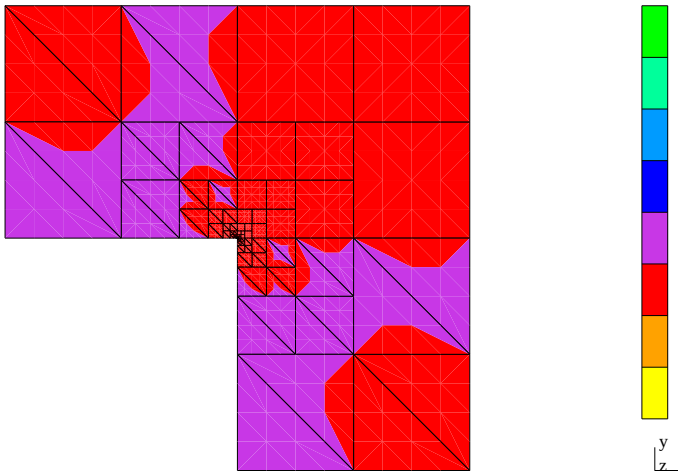


Figure 14.1: L-shape domain example: final, optimal hp mesh

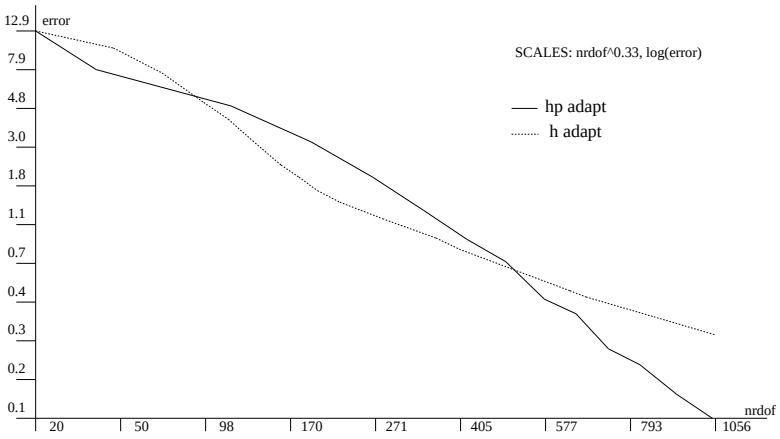


Figure 14.2: L-shape domain example: Convergence history

Note that, in the preasymptotic range, the pure h-adaptive strategy can deliver slightly better results. This is related to the fact that our hp algorithm starts seeing the need for h-refinements only after a sufficient (spectral) information about the solution is available. In simple words, order p has

to go up first ($p = 3$ minimum in this example) before the edges converging at the singular corner are broken. Asymptotically, however, hp-adaptivity always wins, as the fixed order discretization can only deliver algebraic rates of convergence. In problems with smoother solutions, the hp strategy wins earlier, see [58] for more examples.

14.4 Additional remarks

Is the exact solution known? If the exact solution is known, and you are using the code to study the convergence only, set parameter $nexact = 1$ in $adapt/hp_strategy$ (do not forget to recompile the routine...). The exact solution will be used then to evaluate the H_0^1-error and *the exact errors* will be reported in the convergence history. Watch then for the *global effectivity index*,

$$\text{effectivity index} = \frac{\text{error estimate}}{\text{actual error}} \tag{14.4.15}$$

reported by the code.

If the exact solution is unknown, set $nexact = 0$. The error estimate will then be used in place of the actual error in the convergence graphs.

Maximum number of iterations, debugging. When invoking the automatic hp refinements routine, the user is asked to specify the maximum number of iterations and the admissible error level measured in percent of the norm of the solution. After each step of the mesh optimization, the current mesh is saved in file $files/dump$, and the problem can be restarted from the last mesh. This makes it convenient for debugging. A number of printing flags, including graphics control is available in the driver routine $adapt/hp_strategy$, and we invite the user to study the mesh optimization procedure, by displaying all consecutive meshes after various stages of the refinement.

Systems of equations and complex-valued problems. The automatic hp-adaptivity routines share the overall generality of the code. In the case of a system of equations, each quantity discussed in this section is defined by summing over the components of the solution. We remind that all components of the solution are being approximated using the same mesh and no mixed discretizations are supported. In the case of complex-valued problems, all operations are simply done on complex numbers.

Automatic h-adaptivity. For a comparison, a simple h-adaptive strategy has been coded in routine $adapt/h_adapt$. The error estimation and adaptivity is again based on the interaction between the coarse and a fine mesh, this time obtained by performing only a global h-refinement. The element contributions to the global error,

$$\eta_K = \|u_{h/2} - u_h\|_{H_0^1(K)} \,, \tag{14.4.16}$$

are used as error indicator to pick the elements to be refined. The h-adaptivity done on meshes of quadratic elements, frequently delivers slightly better meshes in the preasymptotic range (see the discussion above), and provides itself a valuable tool for solving practical meshes.

Chapter 15

Examples of Applications

The 2D hp technology discussed in the previous chapters, applies to a large class of elliptic problems, including elliptic systems of equations. In this chapter, we present two examples of applications. The first one deals with a class of axisymmetric Maxwell problems that may be reduced to the solution of a single elliptic equation in cylindrical coordinates (r, θ). The second is the classical example of an elliptic system - the linear elasticity equations.

15.1 Linear elasticity

Solving a linear elasticity problem involves determining a displacement vector $\boldsymbol{u} = (u_i)$, symmetric strain tensor ε_{ij} related to the displacement vector components through the Cauchy equation,

$$\varepsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i}) \, ,$$

and stress tensor σ_{ij} related to the strain tensor by a *constitutive law*,

$$\sigma_{ij} = E_{ijkl}\varepsilon_{kl} \, .$$

Here comas denote the partial differentiation, we use the summation convention, and tensor of elasticities E_{ijkl} satisfies the usual symmetry and positive-definitness assumptions,

$$\begin{aligned} E_{ijkl} &= E_{jikl} = E_{klij} \\ E_{ijkl}\varepsilon_{ij}\varepsilon_{kl} &\geq \alpha \, \varepsilon_{ij}\varepsilon_{ij}, \quad \forall \, \varepsilon_{ij} = \varepsilon_{ji} \quad , \end{aligned} \tag{15.1.1}$$

where $\alpha > 0$. In the most general, three-dimensional case of an anisotropic, nonhomogenous material, elasticities E_{ijkl} may involve up to twenty one independent functions of position $\boldsymbol{x}$. For an isotropic, homogeneous solid, the number reduces to just two constants, e.g. the Lamé constants μ, λ,

$$E_{ijkl} = 2\mu\delta_{ik}\delta_{jl} + \lambda\delta ij\delta_{kl} \, ,$$

and the general constitutive law reduces to the Hooke's law,

$$\sigma_{ij} = 2\mu\varepsilon_{ij} + \lambda\varepsilon_{kk}\delta_{ij} \, .$$

In practice we deal with *composite materials*, and μ, λ are piece-wise constant. In the equations above, δ_{ij} denotes the Kronecker's delta.

The stress tensor σ_{ij} represents internal forces developed in the body. Given a point $\boldsymbol{x}$ and an imaginary smooth surface passing through the point and separating the body into two parts, the interaction between the two parts is represented in terms of *internal forces* represented with intensity t_i, the so called stress or traction vector. It follows from the conservation of momentum that the tractions are related to the stress tensor σ_{ij} and a unit vector n_i at point $\boldsymbol{x}$ orthogonal to the sectioning surface through the famous Cauchy tarction-stress relation,

$$t_i = \sigma_{ij} n_j \, . \tag{15.1.2}$$

More precisely, if versor $\boldsymbol{n}$ points from part 1 to part 2 of the body, traction t_i represents the density of internal forces representing the action of part 2 on the part 1. If we reverse vector $\boldsymbol{n}$, we obtain an opposite vector representing the action of part 1 on part 2. The fact that the two vectors are opposite to each other is consistent with the satisfaction of global momentum (or equilibrium) equations.

The stress tensor components must satisfy the momentum equations,

$$\rho\ddot{u}_i - \sigma_{ij,j} = f_i \, .$$

Here ρ denotes the mass density of the solid, dot stands for the differentiation in time, and f_i denote prescribed volume forces, resulting e.g. from gravity. For static problems discussed in this section, the equations of motion reduce to the equilibrium equations,

$$\sigma_{ij,j} = f_i \, .$$

Upon substituting the Hooke's law and the Cauchy geometric relations into the equilibrium equations, we obtain the classical Lamé equations,

$$\mu u_{i,jj} + (\mu + \lambda) u_{j,ji} = f_i \, , \tag{15.1.3}$$

that constitute a system of n elliptic equations, n being the dimension of the problem.

The Lamé equations are solved in a domain Ω occupied by the elastic solid, and they are accompanied by possible different boundary conditions, discussed next.

Prescribed displacements - Dirichlet BC. The displacement vector is prescribed on a part Γ_D of the boundary,

$$u_i = u_i^0 \quad \text{ on } \Gamma_D \, .$$

Prescribed tractions - Neumann BC. The traction vector is prescribed on part Γ_N of the boundary,

$$t_i = \sigma_{ij} n_j = g_i \quad \text{on } \Gamma_N \, .$$

Elastic foundation - Cauchy BC. In this case the body is supported by means of a system of elastic springs, compare the elastic bar example discussed in Part 1 of this book. We have,

$$t_i = \sigma_{ij} n_j = -\beta_{ij} u_j + g_i \quad \text{on } \Gamma_C \, ,$$

where β_{ij} is a positive-definite symmetric matrix. It is also possible to mix different types of the boundary conditions in different directions, compare Exercise 15.3.

Variational formulation - Principle of Virtual Work. Multiplying the equilibrium equations with a vector-valued test function v_i, and integrating over the domain, we obtain,

$$-\int_\Omega \sigma_{ij,j} v_i \, d\boldsymbol{x} + \int_\Omega f_i v_i \, d\boldsymbol{x} \, .$$

As usual, we proceed with the integration by parts of the second order term,

$$-\int_\Omega \sigma_{ij,j} v_i \, d\boldsymbol{x} = \int_\Omega \sigma_{ij} v_{i,j} \, d\boldsymbol{x} - \int_\Gamma \sigma_{ij} n_j \, v_i \, ds \, .$$

The gradient of the test function can be split into its symmetric an antisymmetric parts - the strain tensor ε_{ij} and the tensor of angular displacements ω_{ij},

$$v_{i,j} = \underbrace{\frac{1}{2}(u_{i,j} + u_{j,i})}_{\varepsilon_{ij}} + \underbrace{\frac{1}{2}(u_{i,j} i u_{j,i})}_{\omega_{ij}} \, .$$

We have then (explain why ?),

$$\sigma_{ij} v_{i,j} = \sigma_{ij}(\varepsilon_{ij}(\boldsymbol{v}) + \omega_{ij}(\boldsymbol{v})) = \sigma_{ij} \varepsilon_{ij}(\boldsymbol{v}) \, .$$

On Dirichlet part of the boundary, tractions t_i are unknown a-priori, so we eliminate them by assuming that the test function v_i vanishes on Γ_D. On the remaining parts of the boundary, we replace the stress vector with the prescribed traction and, in the case of the Cauchy BC, the unknown displacement. Separating the terms with the unknown displacement vector form the terms with known *load data*, we obtain the final variational formulation.

$$\begin{cases} u_i = u_i^0 \text{ on } \Gamma_D \\ \displaystyle\int_\Omega \sigma_{ij}(\boldsymbol{u}) \varepsilon_{ij}(\boldsymbol{v}) \, d\boldsymbol{x} + \int_{\Gamma_C} \beta_{ij} u_j v_i \, ds = \int_\Omega f_i v_i \, d\boldsymbol{x} + \int_{\Gamma_N \cup \Gamma_C} g_i v_i \, ds, \\ \qquad\qquad \text{for every test function } v_i, \; v_i = 0 \text{ on } \Gamma_D \end{cases} \tag{15.1.4}$$

The variational formulation is known in Mechanics under the name of the *Principle of Virtual Work: the internal work of the actual stresses on the virtual strains must be equal to the work of the exterior forces on the virtual displacement*. The test functions are known as *virtual displacements*. Upon identifying the $\boldsymbol{H}^1$-space as the finite energy space,

$$\boldsymbol{H}^1(\Omega) = \underbrace{H^1(\Omega) \times \ldots \times H^1(\Omega)}_{n \text{ times}} ,$$

we introduce the space of test functions,

$$\boldsymbol{V} = \{\boldsymbol{v} \in \boldsymbol{H}^1(\Omega) \,:\, \boldsymbol{v} = \boldsymbol{0} \text{ on } \Gamma_D\} ,$$

and assume that the Dirichlet boundary data has been selected in such a way that it admits a finite energy lift (extension),

$$\text{there exists } \tilde{\boldsymbol{u}}^0 \in \boldsymbol{H}^1(\Omega) \text{ such that } \tilde{\boldsymbol{u}}^0 = \boldsymbol{u}^0 \text{ on } \Gamma_D .$$

The variational formulation has then the standard algebraic structure,

$$\begin{cases} \boldsymbol{u} \in \tilde{\boldsymbol{u}}^0 + \boldsymbol{V} \\ b(\boldsymbol{u}, \boldsymbol{v}) = l(\boldsymbol{v}), \quad \forall \boldsymbol{v} \in \boldsymbol{V} , \end{cases}$$

where the bilinear and linear forms correspond to the two sides of the variational formulation,

$$\begin{aligned} b(\boldsymbol{u}, \boldsymbol{v}) &= \int_\Omega (\mu(u_{i,j} + u_{j,i}) + \lambda u_{k,k}\delta_{ij}) v_{i,j} \, d\boldsymbol{x} + \int_{\Gamma_C} \beta_{ij} u_j v_i \, ds , \\ l(\boldsymbol{v}) &= \int_\Omega f_i v_i \, d\boldsymbol{x} + \int_{\Gamma_N \cup \Gamma_C} g_i v_i \, ds . \end{aligned}$$

The case of pure traction boundary conditions. In the case of pure Neumann boundary conditions, the solution can be determined only up to displacement fields for which the strain tensor vanishes - the *linearized rigid body motions*. In the 2D case, $\varepsilon_{11} = u_{1,1} = 0$ implies that the first component must be a function of the second variable only, $u_1 = u_1(x_2)$. Similarly, vanishing of $\varepsilon_{22} = u_{2,2}$ implies that $u_2 = u_2(x_1)$. Consequently, $\varepsilon_{12} = \frac{1}{2}(u_{1,2} + u_{2,1}) = 0$ implies that

$$u_{1,2} = -u_{2,1} .$$

As the left hand side depends only upon x_2 and the right-hand side depends only upon x_1, both sides must be equal to a constant θ, representing an infinitezimal rotation. This results in the final formula for the linearized rigid body motion in 2D,

$$u_1 = c_1 + \theta x_2, \quad u_2 = c_2 - \theta x_1 .$$

In order to fix values of constants c_1, c_2, θ, one has to specify *exactly* three additional, linearly independent conditions. In practice, we set the displacement vector at a specific vertex node, and one (only !) component at some other vertex node to zero.

As the left-hand side vanishes for all virtual displacements corresponding to a linearized rigid body mode, the right-hand side must vanish as well. This implies, compare Exercise 15.4, that the external forces satisfy global equilibrium conditions. Satifaction of the global equilibrium conditions is necessary *and sufficient* for the problem to be well-posed. Conditions, fixing the rigid body mode, should not be confused with Dirichlet boundary conditions, as they produce no reactions at the involved nodes. Imposing even one extra kinematic condition (e.g. taking a short cut, and eliminating both displacement components at two vertex nodes in 2D), produces a discrete solution that has no continuous counterpart - the corresponding problem is mathematically ill posed.

An example: the cracked plate problem. As the code has been designed to solve any system of equations with unknowns that are discretized with continuous elements, very little work needs to be done to solve the elasticity problem. As an example, we will consider the classical "crack problem" illustrated in Fig. 15.1. A cracked elastic, squared plate, is subjected to vertical tractions pulling the plate apart. The problem involves only Neumann boundary conditions, $t_1 = 0, t_2 = q$,

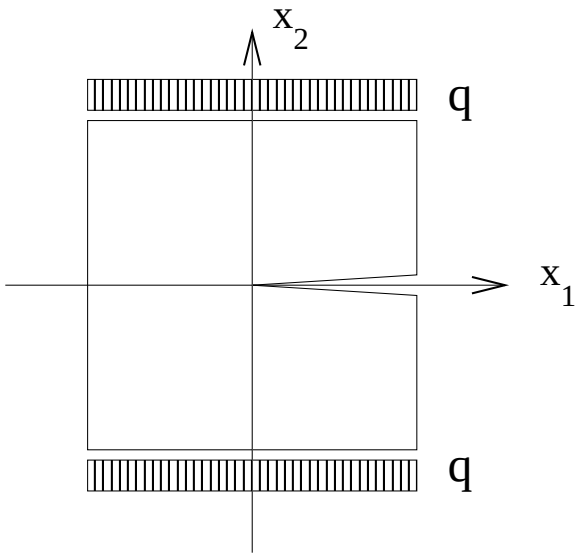


Figure 15.1: Cracked plate problem

along the upper edge, and $t_1 = 0, t_2 = -q$ along the lower edge, with zero tractions along the rest of the boundary including the crack edges. The corresponding input file *files/input_crack* is identical as for a single elliptic equation, except for number of components *NREQNS* = 2. The maximum number of components *MAXEQNS* equals twice the value of *NREQNS*, and it is enforced through the parameter statement in data structure module. Material data are specified in routine *laplace/getmat* and the load data in routines *laplace/getf* and *laplace/getg*. Specification of the material data needs additional comments. The element routine *laplace/elem* has been coded for an arbitrary system of variational equations with the corresponding bilinear and linear forms given by

the formulas,

$$\begin{aligned} b(\boldsymbol{u},\boldsymbol{v}) &= \sum_{I,J=1}^{N} \left\{ \int_\Omega (\sum_{k,l=1}^{2} a_{kl}^{IJ} \frac{\partial u_I}{\partial x_k} \frac{\partial v_J}{\partial x_l} + \sum_{k=1}^{2} b_k^{IJ} \frac{\partial u_I}{\partial x_k} v_J + c^{IJ} u_I v_J)\, d\boldsymbol{x} \right\} \\ l(\boldsymbol{v}) &= \sum_{J=1}^{N} \left\{ \int_\Omega f_J v_J \, d\boldsymbol{x} + \int_{\Gamma_N} g_J v_J \, ds \right\} . \end{aligned}$$

Here N is the number of scalar unknown equals to the number of equations. For elasticity N equals the space dimension but, in general, the two numbers may be different. Setting the elasticity problem involves specifying the material coefficients $a_{kl}^{IJ}, b_k^{IJ} c^{IJ}$ in terms of the Lamé constants μ, λ, comp. Exercise 15.5. The unknowns are organized componentwise, i.e. for a given d.o.f., we list all components first. In other words, the approximate solution is represented in the form,

$$\boldsymbol{u}_{hp} = \sum_k \boldsymbol{u}^k \phi(\boldsymbol{x}) ,$$

where $\boldsymbol{u}^k = (u_1^k, \ldots, u_N^k)$ denote *vector-valued* d.o.f. and $\phi(\boldsymbol{x})$ are *scalar-valued* basis (shape) functions[1]. For solving a linear system of equations, the vector-valued d.o.f. have to be organized into a single vector. This is done in the order,

$$u_1^1, \ldots, u_N^1, u_1^2, \ldots, u_N^2, u_1^3, \ldots, u_N^3, \ldots$$

Consequently, all routines dealing with degrees-of-freedom are organized in such a way that the loop through components is always last. Obviously, for a single equation $N = 1$ and those loops are executed only once.

Elimination of the rigid body motion is done by setting the Dirichlet "boundary condition" flag for the vertex node at the origin, and a special boundary condition flag (= 7) for the node located at $\boldsymbol{x} = (0, 1)$, corresponding to the elimination of the horizontal component only. The appropriate elimination of the horizontal d.o.f. at the vertex node is done in routine *laplace/bcmod*.

Lame constants were set to $\mu = \lambda = 1$, and prescribed traction density $q = 1$.

The results of the hp algorithm are presented in Fig. 15.2 showing the final, optimal hp mesh, and Fig. 15.3 reporting the convergence history. The error tollerance was set at the level of 0.1 percent which was achieved after 10 iterations. The optimal mesh is very similar for the L-shaped domain problem, and the straight line on the algebraic-log scale indicates the exponential convergence. The error has been measured in (problem independent) first order Sobolev seminorm.

[1] For the Maxwell problems discussed in the third part of this volume, the situation will be exactly opposite, d.o.f. will be scalars but the shape functions will be vector-valued

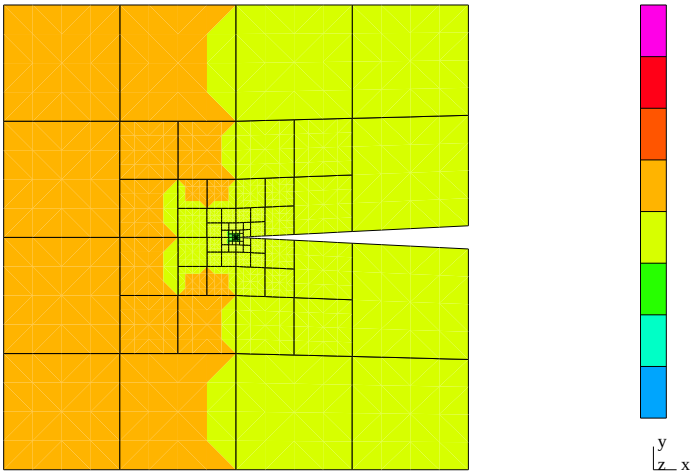


Figure 15.2: The crack problem: optimal (coarse) hp mesh

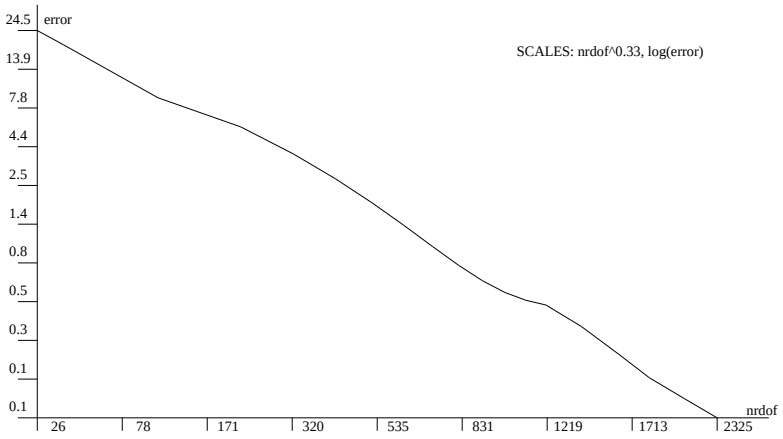


Figure 15.3: The crack problem: convergence history

15.2 An axisymmetric Maxwell problem

We will study the time-harmonic Maxwell equations in the last part of this volume. In the case of axisymmetric problems where the geometry, the load data and, consequently, the solution, are independent of coordinate θ, the Maxwell equations decouple into two separate (systems of) equations corresponding to the so called TE and TM modes.

We shall work with the standard cylindrical coordinates r, θ, z, with the corresponding unit vectors denoted by $\boldsymbol{e}_r, \boldsymbol{e}_\theta, \boldsymbol{e}_z$. Whenever you need to use curvilinear coordinates, we reccomend deriving the variational formulation in Cartesian coordinates and only then switching to the curviliner ones.

For a vector field $\boldsymbol{E} = E_r \boldsymbol{e}_r + E_\theta \boldsymbol{e}_\theta + E_z \boldsymbol{e}_z$ independent of θ, the formula (comp. Exercise 15.6) for curl $\boldsymbol{E}$reduces to,

$$\boldsymbol{\nabla} \times \boldsymbol{E} = \left(-\frac{\partial E_\theta}{\partial z} \right) \boldsymbol{e}_r + \left(\frac{\partial E_r}{\partial z} - \frac{\partial E_z}{\partial r} \right) \boldsymbol{e}_\theta + \left(\frac{\partial E_\theta}{\partial r} + \frac{1}{r} E - \theta \right) \boldsymbol{e}_z \, .$$

The formula for the bilinear form entering the variational formulation for Maxwell equations, see (17.2.7), decouples into two contributions corresponding to components E_r, E_z and transversial component E_θ.

$$\begin{aligned} b(\boldsymbol{E}, \boldsymbol{F}) &= \int_\Omega \left\{ \frac{1}{\mu} (\boldsymbol{\nabla} \times \boldsymbol{E})(\boldsymbol{\nabla} \times \boldsymbol{F}) - (\omega^2 \epsilon - i\omega\sigma) \boldsymbol{E}\boldsymbol{F} \right\} \, d\boldsymbol{x} \\ &= 2\pi \int_{\hat{\Omega}} \left\{ \frac{1}{\mu} (\frac{\partial E_\theta}{\partial r} + \frac{1}{r} E_\theta)(\frac{\partial F_\theta}{\partial r} + \frac{1}{r} F_\theta) - (\omega^2 \epsilon - i\omega\sigma) E_\theta F_\theta \right\} \, r \, dr \, dz \\ &+ 2\pi \int_{\hat{\Omega}} \left\{ \frac{1}{\mu} (\frac{\partial E_r}{\partial z} - \frac{\partial E_z}{\partial r})(\frac{\partial F_r}{\partial z} - \frac{\partial F_z}{\partial r}) - (\omega^2 \epsilon - i\omega\sigma)(E_r F_r + E_z F_z) \right\} \, r \, dr \, dz \, . \end{aligned}$$

Here $\hat{\Omega}$ denotes the "trace" of the axisymmetric three-dimensional domain Ω in the r, z-plane, $\omega, \mu, \epsilon, \sigma$ denote the frequency, permeability, permittivity and conductivity, respectively (see Part 3 for a more complete discussion of Maxwell equations).

A similar decomposition holds for the linear form,

$$\begin{aligned} l(\boldsymbol{F}) &= -i\omega \int_\Omega \boldsymbol{J}^{imp} \boldsymbol{F} \, d\boldsymbol{x} + i\omega \int_{\Gamma_N} \boldsymbol{J}_S^{imp} \boldsymbol{F} \, dS \\ &= 2\pi \left\{ -i\omega \int_{\hat{\Omega}} J_\theta^{imp} F_\theta \, r \, dr \, dz + i\omega \int_{\hat{\Gamma}_N} J_{S,\theta}^{imp} F_\theta \, d\hat{S} \right\} \\ &+ 2\pi \left\{ -i\omega \int_{\hat{\Omega}} (J_r^{imp} F_r + J_z^{imp} F_z) \, r \, dr \, dz + i\omega \int_{\hat{\Gamma}_N} (J_{S,r}^{imp} F_r + J_{S,r}^{imp} F_z) \, d\hat{S} \right\} \, . \end{aligned}$$

Here $\boldsymbol{J}^{imp}$ and $\boldsymbol{J}_S^{imp}$ denote the impressed volume and surface currents, respectively.

Consequently, testing with F_θ, we interact with E_θ only, and test functions F_r, F_z couple with E_r, E_z only. The problem decouples into two separate variational formulations corresponding to components E_r, E_z (the TE mode) or E_θ (the TM mode).

The problem of interest is presented in Fig. 15.4. A single loop coil antenna wrapped around a metallic cylinder radiates into a conductive medium. The data for the problem are as follows:

$$\begin{aligned} \text{domain} &= 2 \times 4 \text{ m} \\ \mu &= 1 \\ \epsilon &= 1 \\ f &= 2 \text{ MHz} \\ \sigma_{\text{metal}} &= 10^7 \text{ S/m} \\ \sigma_{\text{medium}} &= 1 \text{ S/m} \, . \end{aligned}$$

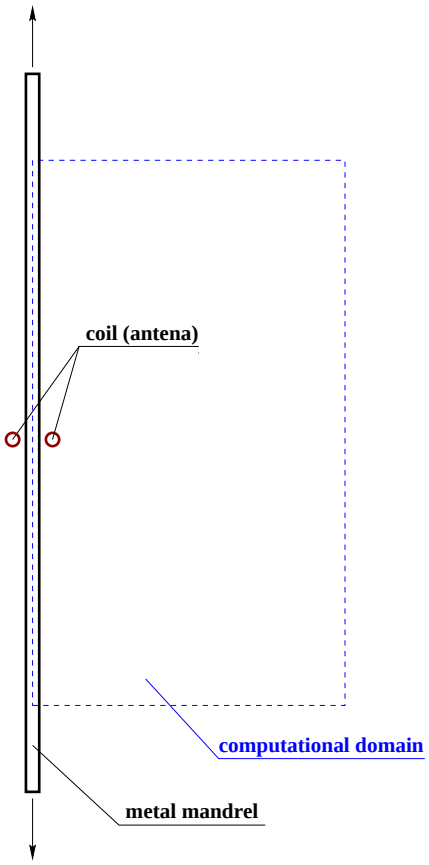


Figure 15.4: Radiation from a coil antenna into a conductive medium

The antenna is modeled with an impressed surface current prescribed on the antenna boundary. As only the $J^{imp}_{S,\theta}$ component is different from zero, the problem reduces to the solution of the equation for the TM mode E_θ.

Due to an exponential decay of the solution, we truncate the infinite domain to the computational rectangular domain indicated above, and apply Perfect Electric Conductor (PEC) boundary condition, $E_\theta = 0$, on the whole boundary of the domain except for the antenna boundary. Notice that the domain includes a portion of the metallic mandrel. We do not model the cylinder with approximate impedance boundary conditions, instead we attempt to resolve the boundary layer (skin effect) directly.

The final variational formulation of the TM problem looks as follows.

$$\begin{cases} E \in V \\ \displaystyle\int_\Omega \left\{ \frac{1}{\mu}[\frac{1}{r^2}(E + r\frac{\partial E}{\partial r})(F + r\frac{\partial F}{\partial r}) + \frac{\partial E}{\partial z}\frac{\partial F}{\partial z}] - (\omega^2\epsilon - i\omega\sigma)EF \right\} rdrdz \\ \displaystyle = i\omega \int_{\Gamma_N} r J^{imp}_S F \, ds, \quad \forall F \in V \, , \end{cases}$$

where

$$V = \{F \in H^1(\Omega) \ : \ F = 0 \text{ on } \Gamma_D\} \ .$$

Here, for simplicity, we have dropped "hats" in the notation for the trace domain and parts of its boundary, and index θ. Γ_D is the outer boundary of the rectangular domain, and Γ_N corresponds to the boundary of the antenna, modeled with a squared crossection. The density of impressed surface current is assumed to be constant, with the value corresponding to a unit lumped current in the antenna.

Figure 15.5 presents the convergence history for the hp algorithm. The first curve was produced beginning with a mesh of linear elements only. The increase in the reported error for meshes around 1000 d.o.f. illustrates a limitation of using the difference between the coarse and fine grid solution for an error estimate. The initial mesh of linear elements "does not see" the boundary layer. Mesh refinements originate around the antenna and propagate in all directions. Around a 1000 d.o.f. mesh, the refinements reach the metal cylinder and the fine mesh solution drastically changes - the mesh begins to "see" the boundary layer. Consequently, the error estimate goes up. Once the boundary layer has been resolved by the fine mesh, the convergence curve stabilizes, and the exponential convergence can be seen. The second curve was produced starting with the same coarse mesh and $p = 1$ except for the elements in the metallic cylinder where the *horizontal* order of approximation was increased to 4. The boundary layer is then seen from the very beginning and the reported convergence curve is monotone. Figure 15.6 presents details of our best mesh. Notice the full

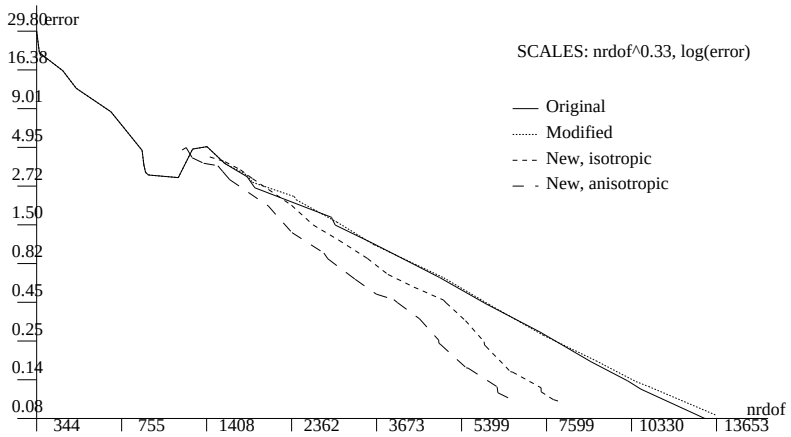


Figure 15.5: Radiation from a coil antenna: convergence history, error estimate/norm vs number of d.o.f.

symmetry of the mesh with respect to the horizontal axis. The corresponding imaginary part of the solution representing the direct radiation from the antenna, and the real part, representing the

secondary radiation resulting from the reflection from the mandrel, are shown in Figures 15.7 and 15.8.

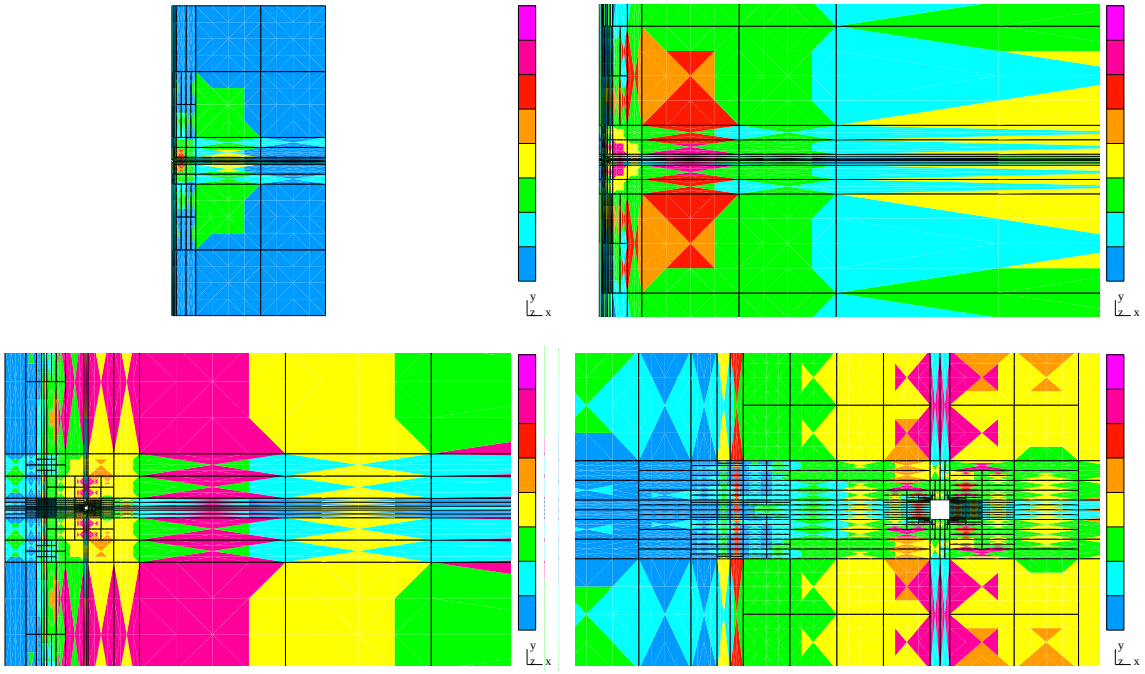


Figure 15.6: Radiation from a coil antenna: the optimal hp mesh. Zooms on the antenna with factors 1,10,100,500

Exercises

Exercise 15.1 Prove that any tensor $E_{ijkl}, i, j, k, l = 1, \ldots, 3$, satisfying symmetry assumptions (15.1.1) has at most 21 different components.

Exercise 15.2 Show that the symmetry assumptions (15.1.1) imply that there exists a homogeneous quadratic form $W = W(\varepsilon_{ij})$ (strain energy density) such that,

$$\sigma_{ij} = \frac{\partial W}{\partial \varepsilon_{ij}} .$$

For an isotropic material, $W(\varepsilon_{ij})$ must be invariant under the change of a Cartesian system of coordinates (group of rotations). More precisely,

$$W(\varepsilon'_{ij}) = W(\varepsilon_{ij}), \qquad \text{where } \varepsilon'_{ij} = \alpha_{ik}\alpha_{jk}\varepsilon_{kl},$$

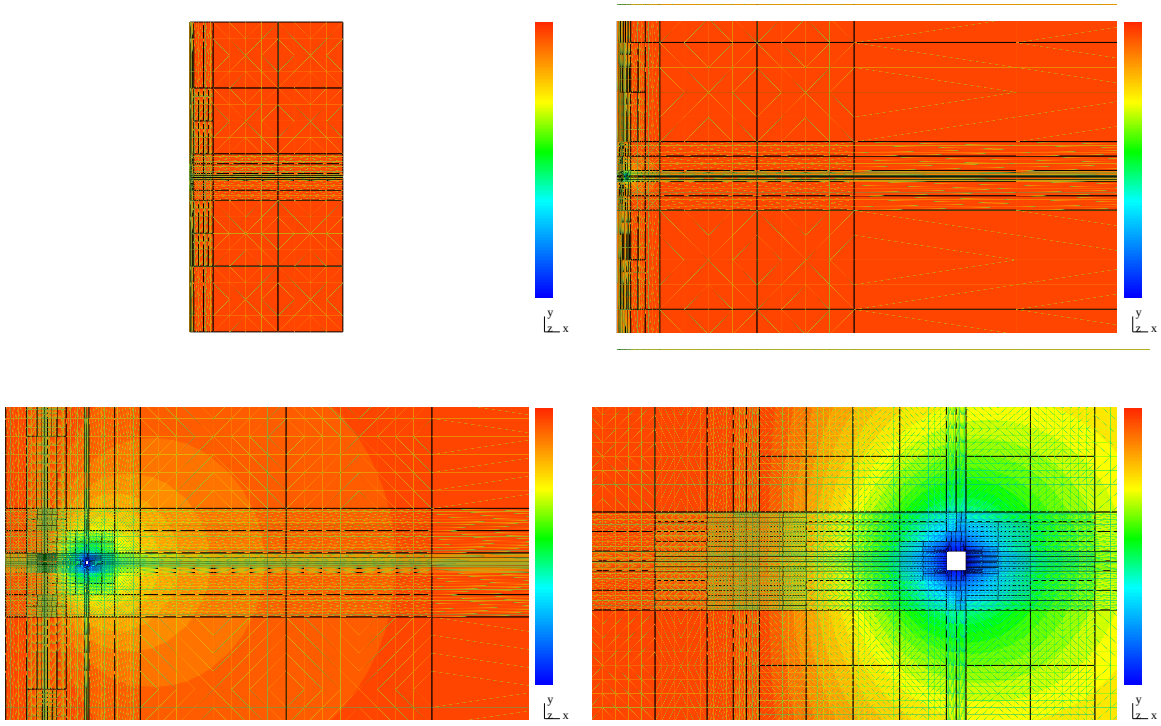


Figure 15.7: Radiation from a coil antenna: zooming on the imaginary part of the solution

for any orthogonal transformation matrix $\alpha^T = \alpha^{-1}$ representing rotation of the system of coordinates. Show that for any strain energy density satisfying the isotropic invariance property, there exists a unique function $\hat{W}(\lambda_1, \lambda_2, \lambda_3)$ (in 3D) such that

$$W = W(\varepsilon_{ij}) = \hat{W}(\lambda_1(\varepsilon_{ij}), \lambda_2(\varepsilon_{ij}), \lambda_3(\varepsilon_{ij})) \,,$$

where $\lambda_i(\varepsilon_{ij})$ denote the algebraic invariants of the strain tensor. Conclude that, for a homogeneous quadratic form, $\hat{W}(\lambda_1, \lambda_2, \lambda_3)$ must be of the form,

$$\hat{W}(\lambda_1, \lambda_2, \lambda_3) = a_1\lambda_1^2 + a_2\lambda_2 \,,$$

and, therefore, the strain energy density for a linear isotropic solid may involve only two material constants. Relate constants a_i to Lamé constants.

Exercise 15.3 Consider the case of mixed boundary conditions, applied on a part Γ_M of the boundary.

$$\sigma_n = -\beta u_n \,, \quad \boldsymbol{t}_t = \boldsymbol{0} \,.$$

Here $\sigma_n = t_i n_i = \sigma_{ij} n_j n_i$ represents the normal component of the stress vector and $\boldsymbol{t}_t$ stands for the remaining tangentail component,

$$\boldsymbol{t} = \sigma_n \boldsymbol{n} + \boldsymbol{t}_t \,.$$

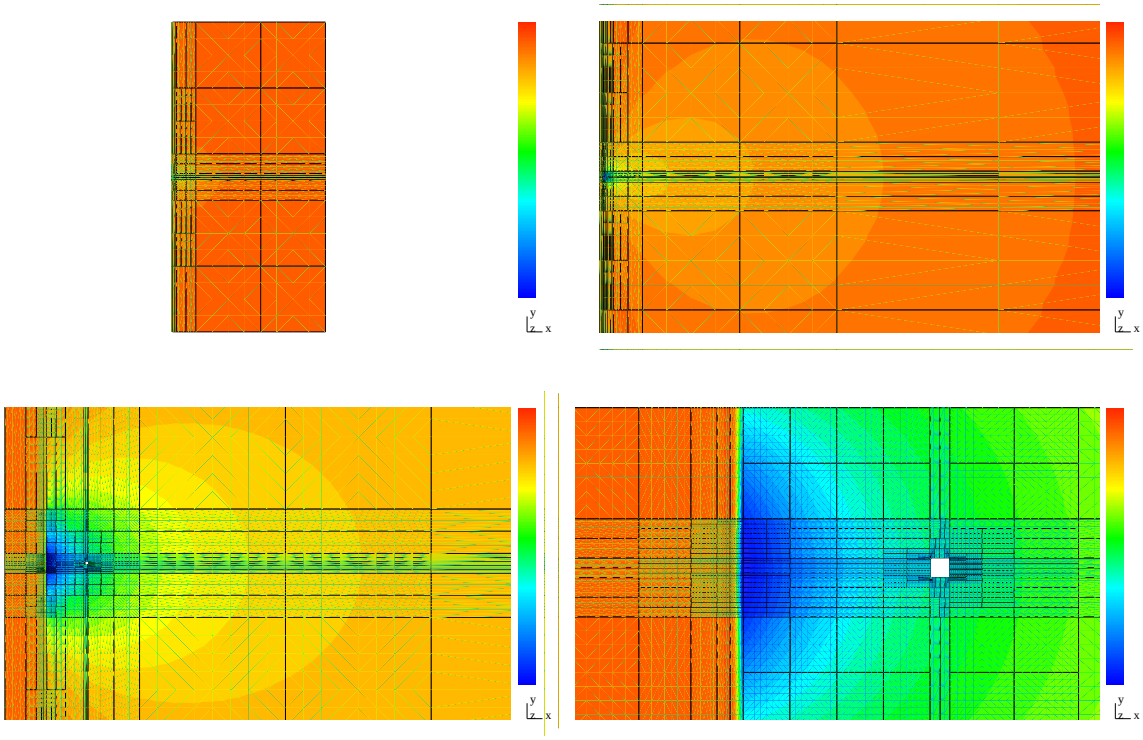


Figure 15.8: Radiation from a coil antenna: zooming on the real part of the solution

A similar decomposition holds for the displacement vector,

$$\boldsymbol{u} = u_n \boldsymbol{n} + \boldsymbol{u}_t \, .$$

Discuss how the boundary condition enters the variational formulation (Principle of Virtual Work), and discuss its implementation in a FE code.

Exercise 15.4 Let $l(\boldsymbol{v})$ be the linear form corresponding to the Principle of Virtual Work. Assume that $l(\boldsymbol{v}) = 0$, for all virtual displacements corresponding to a linearized rigid body mode, $\varepsilon(\boldsymbol{v}) = 0$. Prove that this is equivalent to the *global equilibrium equations* satisfied by the exterior forces,

$$\int_\Omega \boldsymbol{f} \, d\boldsymbol{x} + \int_{\Gamma_N \cup \Gamma_C} \boldsymbol{g} \, ds = \boldsymbol{0},$$
$$\int_\Omega \boldsymbol{x} \times \boldsymbol{f} \, d\boldsymbol{x} + \int_{\Gamma_N \cup \Gamma_C} \boldsymbol{x} \times \boldsymbol{g} \, ds = \boldsymbol{0} \, .$$

Exercise 15.5 Determine material constants $a_{kl}^{IJ}, b_k^{IJ} c^{IJ}$ in terms of the Lamé constants μ, λ.

Exercise 15.6 Let $\boldsymbol{x} = \boldsymbol{x}(u^j), \; j = 1, 2, 3$ be an arbitrary curvilinear system of coordinates with

corresponding basis vectors,

$$\boldsymbol{a}_i = \frac{\partial \boldsymbol{x}}{\partial u^i} = \frac{\partial x_j}{\partial u^i}\boldsymbol{e}_j,\ i = 1, 2, 3\ ,$$

where $\boldsymbol{e}_j$ denote the unit vectors of the Cartesian system of coordinates.

- Prove that the corresponding dual basis vectors are given by formulas,

$$\boldsymbol{a}^j = \frac{\partial u^j}{\partial x_i}\boldsymbol{e}_i\ .$$

- Use the representation above to prove the following formulas for grad, div and curl operators.

$$\boldsymbol{\nabla} w = \frac{\partial w}{\partial u^j}\boldsymbol{a}^j, \quad \boldsymbol{\nabla} \cdot \boldsymbol{v} = \frac{\partial \boldsymbol{v}}{\partial u^j} \cdot \boldsymbol{a}^j, \quad \boldsymbol{\nabla} \times \boldsymbol{E} = -\frac{\partial \boldsymbol{E}}{\partial u^j} \times \boldsymbol{a}^j\ .$$

- Use the general formulas to derive the formulas for grad, div and curl in cylindrical and spherical coordinates. Both systems fall into the class of *orthogonal systems of coordinates* for which it is customary to work with unit vectors, identical for the basis and cobasis vectors.

Chapter 16

Exterior Boundary-Value Problems

If domain Ω in which the differential equation is being solved, coincides with the complement of a bounded domain $\Omega_{int},\ \Omega = \mathbb{R}^2 - \Omega_{int}$, we speak of an *exterior boundary-value problem*. On top of standard boundary conditions imposed on the domain boundary, an additional *decay condition at infinity* must be added.

We shall focus on a class of model rigid scattering problems for acoustics. Given an *incident pressure wave* u^{inc} that satisfies the Helmholtz equation on the whole plane,

$$-\Delta u^{inc} - k^2 u^{inc} = 0 \quad \text{in } \mathbb{R}^2 \ ,$$

we look for a *scattered pressure wave* u that satisfies:

- the same Helmholtz equation, but only in the exterior domain,

$$-\Delta u - k^2 u = 0 \quad \text{in } \Omega = \mathbb{R}^2 - \Omega_{int} \ ,$$

- Neumann boundary condition on boundary $\Gamma = \partial\Omega = \partial\Omega_{int}$,

$$\frac{\partial (u + u^{inc})}{\partial n} = 0 \ ,$$

- Sommerfeld radiation condition at infinity,

$$\frac{\partial u}{\partial r} + iku \in L^2(\Omega) \ .$$

Here $k = \frac{\omega}{c} > 0$ is the wave number with ω representing the angular frequency, and c the speed of sound in the (homogeneous) medium, $\boldsymbol{n}$ is the outward unit vector, and r stands for the radial coordinate corresponding to polar coordinates with the origin at an arbitrary point, typically chosen inside of the interior domain Ω_{int} occupied by the rigid obstacle (scatterer), comp. Exercise 16.1. As the gradient of the pressure is proportional to the velocity vector, the Neumann boundary condition

corresponds to the requirement that the normal component of the total velocity vector must vanish on the boundary of the rigid obstacle. The sum of the incident and scattered pressures represents the total pressure wave. In practice, the incident pressure wave corresponds to a source located away from the obstacle. If the location of the source is far enough, the incident pressure wave may be assumed in the form of a plane wave (source at infinity),

$$u^{inc}(\boldsymbol{x}) = u_0 e^{-ik\boldsymbol{e}\cdot\boldsymbol{x}} ,$$

where u_0 is a scaling factor, and unit vector $\boldsymbol{e}$ represents the direction of the incoming incident wave. As the problem is linear, we can assume $u_0 = 1$. The role of the incident wave is therefore to provide only a driving Neumann load data on boundary Γ,

$$\frac{\partial u}{\partial n} = g := -\frac{\partial u^{inc}}{\partial n} . \tag{16.0.1}$$

The Sommerfeld radiation condition represents a requirement that the scattered wave must be *outgoing* to infinity, and it can be represented in many different but equivalent forms. The form used here has a particular advantage of being independent of space dimension, the same condition will hold in $\mathbb{R}^3$. Notice that the formula for the plane wave, and the sign in the Sommerfeld radiation condition correspond to the $e^{i\omega t}$ ansatz in time.

Of particular interest may be the computation of the monostatic *Echo Area* (EA) defined as,

$$\lim_{r\to\infty} |u(r\hat{\boldsymbol{x}})\,|r^{\alpha} . \tag{16.0.2}$$

Here $\hat{\boldsymbol{x}}$ is a point on the unit circle (sphere in 3D) opposite to the direction of the incident wave, $\boldsymbol{e} = -\hat{\boldsymbol{x}}$, and u is the corresponding scattered wave. Coefficient $\alpha = 1/2$ reflects the decay rate of the leading term (far field pattern) of the solution, in three space dimensions, $\alpha = 1$.

16.1 Variational formulation. Infinite element discretization

The idea of coupled finite/infinite elements is illustrated in Fig.16.1. The scatterer is surrounded with a truncating circle partitioning the exterior domain Ω into the "near-field" domain Ω^a in between the scatterer and the truncating surface, and the remaining "far-field" domain Ω^a. The near-field domain is then meshed with finite elements, and the "far-field" domain is discretized with infinite elements, in a fully conforming way. In practice, the exact geometry of the truncating circle is replaced with its FE isoparametric approximation. The infinite elements are aligned with the coordinate lines of the polar system of coordinates.

We will discuss the variational formulation for the "far-field" part of the domain first.

We introduce a larger circle of radius R, and follow the standard derivation of the variational formulation for the truncated far-field domain,

$$\Omega_a^R = \{\boldsymbol{x} \,:\, a < |\boldsymbol{x}| < R\} .$$

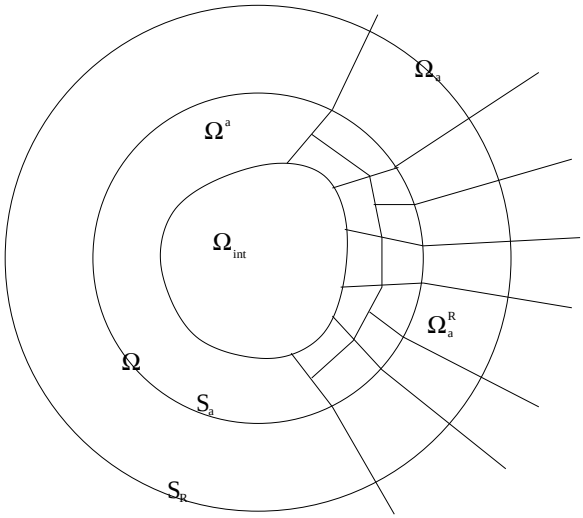


Figure 16.1: FE/IE mesh around a scatterer

Assuming $u, v \in H^1(\Omega_a^R)$, we obtain[1],

$$\int_{\Omega_a^R} \{\boldsymbol{\nabla} u \boldsymbol{\nabla} \bar{v} - k^2 u \bar{v}\}\, d\boldsymbol{x} + ik \int_{S_R} u \bar{v}\, ds = \int_{S_a} \frac{\partial u}{\partial n} \bar{v}\, ds + \int_{S_R} w \bar{v}\, ds, \quad \forall v\,. \tag{16.1.3}$$

Here $w := \frac{\partial u}{\partial r} + iku$, according to the Sommerfeld condition, denotes an unknown function, L^2-integrable in Ω^a, and S_a denotes the truncating circle.

We want to pass with $R \to +\infty$ in (16.1.3). The L^2-integrability of function w implies that the unknown term involving function w will vanish. The first boundary integral on the right-hand side is independent of R, so we need to focus on the two integrals on the left-hand side only.

System of coordinates. We will use a curvilinear system of coordinates, corresponding to an arbitrary parametrization of the truncating circle,

$$\boldsymbol{x}(r, \xi) = r \boldsymbol{x}_a(\xi), \quad r > 1, \quad |\boldsymbol{x}_a| = a\,, \tag{16.1.4}$$

where $\boldsymbol{x}_a(\xi)$ denotes the parametrization of the truncating circle of radius a.

[1] $\bar{v}$ represents the complex conjugate of v.

We have the standard formulas,

$$\begin{aligned}
&\frac{\partial \boldsymbol{x}}{\partial r} = \boldsymbol{x}_a = a\boldsymbol{e}_r, \text{ where } \boldsymbol{e}_r := a^{-1}\boldsymbol{x}_a, \\
&\frac{\partial \boldsymbol{x}}{\partial \xi} = r\,\frac{d\boldsymbol{x}_a}{d\xi} = r|\frac{d\boldsymbol{x}_a}{d\xi}|\,\boldsymbol{e}, \text{ where } \boldsymbol{e} := |\frac{\partial \boldsymbol{x}}{\partial \xi}|^{-1}\frac{\partial \boldsymbol{x}}{\partial \xi}\,, \\
&\boldsymbol{\nabla} u = \frac{1}{a}\frac{\partial u}{\partial r}\boldsymbol{e}_r + \frac{1}{r}\underbrace{\frac{\partial u}{\partial \xi}|\frac{d\boldsymbol{x}_a}{d\xi}|^{-1}\boldsymbol{e}}_{\boldsymbol{\nabla}_{S_a} u} = \frac{1}{a}\frac{\partial u}{\partial r}\boldsymbol{e}_r + \frac{1}{r}\boldsymbol{\nabla}_{S_a} u\,,
\end{aligned} \tag{16.1.5}$$

with $\boldsymbol{\nabla}_{S_a}$ denoting the gradient on the circle S_a.

The sesquilinear form in the variational formulation over domain Ω_a^R is computed now as follows,

$$\begin{aligned}
b(u,v) \;&= a^{-1}\int_{S_a}\int_1^\rho \left(\frac{\partial u}{\partial r}\frac{\partial \bar{v}}{\partial r} - (ka)^2 u\bar{v}\right)\, r^{2\alpha} dr dS_a + a\int_{S_a}\int_1^\rho \frac{1}{r^2}\boldsymbol{\nabla}_{S_a} u \boldsymbol{\nabla}_{S_a}\bar{v}\; r^{2\alpha} dr dS_a \\
&\quad + ik\,\rho^{2\alpha}\int_{S_a} u(\rho,\cdot)\bar{v}(\rho,\cdot)\, dS_a\,,
\end{aligned} \tag{16.1.6}$$

where $\rho a = R$, $dS_a = |\frac{d\boldsymbol{x}_a}{d\xi}|d\xi$.

Incorporating the far-field pattern. Change of dependent variable. Consistent with the far-field pattern, known from the Atkinson-Wilcox expansion (comp. Exercise 16.3), we shall seek the solution u in the form,

$$u = \frac{e^{-ika(r-1)}}{r^\alpha} U\,, \tag{16.1.7}$$

where U is the new unknown, and α is the dimension dependent constant. An identical substitution will be used for test functions. The idea is to substitute these representations into the variational formulation over the truncated far-field domain Ω_a^R, cancel out all Lebesgue non-integrable terms, and then pass with $R \to \infty$.

Substituting (16.1.7) into the second integral in the formula for the sesquilinear form (16.1.6) yields simply,

$$a\int_{S_a}\int_1^\rho \frac{1}{r^2}\boldsymbol{\nabla}_{S_a} U \boldsymbol{\nabla}_{S_a}\bar{V}\; dr dS_a\,, \tag{16.1.8}$$

and we need to focus only on the inner integral of the sum of the first and the third term,

$$a^{-1}\int_1^\rho \left(\frac{\partial u}{\partial r}\frac{\partial \bar{v}}{\partial r} - (ka)^2 u\bar{v}\right)\, r^{2\alpha} dr + ik\rho^{2\alpha} u(\rho)\bar{v}(\rho)\,, \tag{16.1.9}$$

for a particular value of ξ.

We shall proceed with the substitution (16.1.7) in three steps.

Step 1: Taking out the oscillating factor. By setting

$$u(r) = e^{-ika(r-1)}U(r), \quad \bar{v}(r) = e^{ika(r-1)}\bar{V}(r),$$

we obtain,

$$\frac{\partial u}{\partial r} = (\frac{\partial U}{\partial r} - ikaU)e^{-ika(r-1)}, \quad \frac{\partial \bar{v}}{\partial r} = (\frac{\partial \bar{V}}{\partial r} + ika\bar{V})e^{ika(r-1)}.$$

This leads to the following modification of the integrand (16.1.9),

$$a^{-1}\int_1^\rho \left[\frac{\partial U}{\partial r}\frac{\partial \bar{V}}{\partial r} + ika\left(\frac{\partial U}{\partial r}\bar{V} - U\frac{\partial \bar{V}}{\partial r}\right)\right] r^{2\alpha}dr + ik\rho^{2\alpha}U(\rho)\bar{V}(\rho)\,. \tag{16.1.10}$$

Notice that the cancellation obtained so far, does not guarantee Lebesgue integrability of all the remaining terms. With a leading term $1/r$ for both U and V, the first term in the integrand is integrable but the second and third are not. In order to avoid introducing new symbols, we shall trade now symbols U, V back for u, v,

$$a^{-1}\int_1^\rho \left[\frac{\partial u}{\partial r}\frac{\partial \bar{v}}{\partial r} + ika\left(\frac{\partial u}{\partial r}\bar{v} - u\frac{\partial \bar{v}}{\partial r}\right)\right] r^{2\alpha}dr + ik\rho^{2\alpha}u(\rho)\bar{v}(\rho)\,. \tag{16.1.11}$$

Step 2: Taking out the Jacobian. We substitute,

$$u = r^{-\alpha}U, \qquad \bar{v} = r^{-\alpha}\bar{V},$$
$$\frac{\partial u}{\partial r} = (\frac{\partial U}{\partial r} - \frac{\alpha}{r}U)r^{-\alpha}, \quad \frac{\partial \bar{v}}{\partial r} = (\frac{\partial \bar{V}}{\partial r} - \frac{\alpha}{r}\bar{V})r^{-\alpha}\,.$$

This eliminates the Jacobian from the integrand (16.1.11),

$$a^{-1}\int_1^\rho \left[(\frac{\partial U}{\partial r} - \frac{\alpha}{r}U)(\frac{\partial \bar{V}}{\partial r} - \frac{\alpha}{r}\bar{V}) + ika\left(\frac{\partial U}{\partial r}\bar{V} - U\frac{\partial \bar{V}}{\partial r}\right)\right] dr + ikU(\rho)\bar{V}(\rho)\,. \tag{16.1.12}$$

Integration by parts of the last term in the integrand leads to,

$$a^{-1}\int_1^\rho \left[(\frac{\partial U}{\partial r} - \frac{\alpha}{r}U)(\frac{\partial \bar{V}}{\partial r} - \frac{\alpha}{r}\bar{V}) + 2ika\frac{\partial U}{\partial r}\bar{V}\right] dr + ikU(1)\bar{V}(1)\,. \tag{16.1.13}$$

Expanding the first term leads to,

$$a^{-1}\int_1^\rho \left[\frac{\partial U}{\partial r}\frac{\partial \bar{V}}{\partial r} - \frac{\alpha}{r}\frac{\partial U}{\partial r}\bar{V} - \frac{\alpha}{r}U\frac{\partial \bar{V}}{\partial r} + \frac{\alpha^2}{r^2}U\bar{V} + 2ika\frac{\partial U}{\partial r}\bar{V}\right] dr + ikU(1)\bar{V}(1)\,. \tag{16.1.14}$$

Finally, after integrating by parts the third term, we get,

$$a^{-1}\int_1^\rho \left[\frac{\partial U}{\partial r}\frac{\partial \bar{V}}{\partial r} + \frac{\alpha(\alpha-1)}{r^2}U\bar{V} + 2ika\frac{\partial U}{\partial r}\bar{V}\right] dr + \frac{\alpha+ika}{a}U(1)\bar{V}(1) - \frac{\alpha}{\rho a}U(\rho)\bar{V}(\rho)\,. \tag{16.1.15}$$

Upon passing with $\rho \to \infty$, we obtain,

$$a^{-1}\int_1^\infty \left[\frac{\partial U}{\partial r}\frac{\partial \bar{V}}{\partial r} + \frac{\alpha(\alpha-1)}{r^2}U\bar{V} + 2ika\frac{\partial U}{\partial r}\bar{V}\right] dr + \frac{\alpha+ika}{a}U(1)\bar{V}(1)\,. \tag{16.1.16}$$

Step 3: Change of the independent variable. Our final substitution involves mapping a neighborhood of infinity into a neighborhood of zero. We substitute,

$$r = \frac{1}{x}, \quad dr = -\frac{1}{x^2}\,dx, \quad \frac{dU}{dr} = -x^2\frac{dU}{dx}\,. \tag{16.1.17}$$

The final formula for the sesquilinear form is,

$$\begin{aligned} b(u,v) &= \int_{S_a}\int_0^1 \left\{ a^{-1}\left[x^2\frac{\partial U}{\partial x}\frac{\partial \bar{V}}{\partial x} + \alpha(\alpha-1)U\bar{V} - 2ika\frac{\partial U}{\partial x}\bar{V}\right] + a\boldsymbol{\nabla}_{S_a}U\boldsymbol{\nabla}_{S_a}\bar{V}\right\} dxdS_a \\ &\quad + \frac{\alpha+ika}{a}\int_{S_a} U(1,\cdot)\bar{V}(1,\cdot)\,dS_a\,, \end{aligned} \tag{16.1.18}$$

where the combined transformation is:

$$u(r,\xi) = \frac{e^{-ika(r-1)}}{r^\alpha}U(r,\xi) = e^{-ika(x^{-1}-1)}x^\alpha U(x,\xi)\,, \tag{16.1.19}$$

with an identical formula for the test functions.

The ultimate variational formulation is obtained by taking the standard variational formulation for the "near-field" domain,

$$\int_{\Omega^a}\{\boldsymbol{\nabla}u\boldsymbol{\nabla}\bar{v} - k^2u\bar{v}\}d\boldsymbol{x} = \int_\Gamma g\bar{v}\,ds + \int_{S_a}\frac{\partial u}{\partial n}\bar{v}\,ds, \quad \forall v\,,$$

and summing it up with (16.1.3). We obtain the final variational formulation in the standard abstract form,

$$\begin{cases} u \in X \\ b(u,v) = l(v), \quad \forall v \in Y, \end{cases} \tag{16.1.20}$$

where X and Y are some appropriate functional spaces, and the sesquilinear and antilinear forms are given by,

$$\begin{aligned} b(u,v) &= \int_{\Omega^a}\{\boldsymbol{\nabla}u\boldsymbol{\nabla}\bar{v} - k^2u\bar{v}\}d\boldsymbol{x} \\ &= \int_{S_a}\int_0^1 \left\{ a^{-1}\left[x^2\frac{\partial U}{\partial x}\frac{\partial \bar{V}}{\partial x} + \alpha(\alpha-1)U\bar{V} - 2ika\frac{\partial U}{\partial x}\bar{V}\right] + a\boldsymbol{\nabla}_{S_a}U\boldsymbol{\nabla}_{S_a}\bar{V}\right\} dxdS_a \\ &\quad + \frac{\alpha+ika}{a}\int_{S_a} U(1,\cdot)\bar{V}(1,\cdot)\,dS_a \\ l(v) &= \int_\Gamma g\bar{v}\,ds\,. \end{aligned} \tag{16.1.21}$$

16.2 Selection of IE radial shape functions

As usual, shape functions for the infinite elements are constructed by taking tensor products of one-dimensional radial shape functions of coordinate x (or, equivalently, r) and standard FE shape functions corresponding to the edge (face in 3D) lying on the truncating circle (sphere),

$$U(x,\xi) = \sum_{kl} U_{kl}\psi_k(x)e_l(\xi)\ . \tag{16.2.22}$$

The corresponding infinite element space of test functions is the tensor product of polynomials of order N in x and polynomials of order p in ξ, $\mathcal{P}^N \otimes \mathcal{P}^p$. Here N is the (radial) order of the infinite element. The construction extends in a standard way to elements of variable order.

There are many possible choices for the IE radial shape functions.

Same trial and test functions (Bubnov-Galerkin). The leading term (derivative times derivative) in (16.1.18) suggests the use of integrated Jacobi polynomials $P_n^{(0,2)}$ (transformed to interval (0,1)),

$$\psi_j(\xi) = \begin{cases} 1 & j = 0 \\ \displaystyle\int_\xi^1 P_{j-1}^{(0,2)}(2t-1)\,dt & j \geq 1\ . \end{cases} \tag{16.2.23}$$

Only the first IE shape functions "connects" with the adjacent finite element, the remaining ones vanish on the truncating surface and, in terminology of hp methods, they can be classified as IE "bubbles". Shape functions $\psi_j, j \geq 1$, can be normalized to satisfy a desired scaling condition. The same shape functions are used for the test functions. Recall that Jacobi polynomials $P_n^{(0,2)}$ are orthogonal in the weighted L^2 inner product,

$$\int_{-1}^1 (1+t)^2 u'v'\,dt\ . \tag{16.2.24}$$

After rescaling $x = 2t-1$, the corresponding polynomials, defined on the unit interval $(0,1)$, are orthogonal in the inner product,

$$\int_0^1 x^2 u'v'\,dx\ , \tag{16.2.25}$$

corresponding to the first term in the sesquilinear form (16.1.18). The rationale of the choice is that, asymptotically in order N, the leading term is most responsible for conditioning problems.

Notice that, except for the choice of the radial shape functions, the representation for the approximate solution is standard,

$$u(r,\xi) = \frac{e^{-ika(r-1)}}{r^\alpha} \sum_j \underbrace{\sum_l U_{jl}e_l(\xi)}_{u_j(\xi)}\,\psi_j(\frac{1}{r})\ . \tag{16.2.26}$$

The main point is the choice of the *same* test functions, possible only because of the interpretation of the integrals in the CPV sense. Notice also that the construction is identical for both 2D and 3D problems. The difference between 2D and 3D is reflected only by different values of coefficient α in formulas (16.1.19) and (16.1.18).

Different trial and test functions (Petrov-Galerkin). If all the integrals are to be interpreted in the Lebesgue sense only, the leading term for the test functions $V(x)$ must be $x^{2\alpha}$. In other words, the test functions must be of order $1/r^2$ in 2D, and order $1/r^3$ in 3D. Replacing V with $x^{2\alpha}V$ in (16.1.18), we obtain a modified formula for the sesquilinear form,

$$\begin{aligned} b(u,v) \quad &= \int_{S_a}\int_0^1 \{a^{-1}\left[x^{2+2\alpha}\frac{\partial U}{\partial x}\frac{\partial \bar{V}}{\partial x} + \alpha(\alpha-1)x^{2\alpha}U\bar{V} + (2\alpha x^{2\alpha-1} - 2ika)x^{2\alpha}\frac{\partial U}{\partial x}\bar{V}\right] \\ &\quad + ax^{2\alpha}\boldsymbol{\nabla}_{S_a}U\boldsymbol{\nabla}_{S_a}\bar{V}\}\, dx dS_a + \frac{\alpha+ika}{a}\int_{S_a} U(1,\cdot)\bar{V}(1,\cdot)\, dS_a\,, \end{aligned} \tag{16.2.27}$$

The leading term suggests now the use of integrated Jacobi polynomials $P_n^{(0,3)}$ in 2D, and integrated Jacobi polynomials $P_n^{(0,4)}$ in 3D. The choice is no longer dimension independent. In 2D, we have,

$$\psi_j(\xi) = \begin{cases} 1 & j = 0 \\ \displaystyle\int_\xi^1 P_{j-1}^{(0,3)}(2t-1)\, dt & j \geq 1\,. \end{cases} \tag{16.2.28}$$

Notice that the substitution $V := x^{2\alpha}V$ is equivalent with the choice of *different* test shape functions,

$$\phi_j(x) = x^{2\alpha}\psi_j(x)\,. \tag{16.2.29}$$

It is for this reason that we identify it as the Petrov-Galerkin method. With such a choice of test functions, there is no need for the CPV interpretation of the integrals. In terms of spaces, the method is identical with the element of Astley et al. [14].

We will compare the two methods with a few numerical experiments in the next sections.

16.3 Implementaion

Data structure, interface with frontal solver. Conceptually, infinite elements may be treated the same way as finite elements. Shape-wise, in 2D, they fall into the category of quads. The corresponding shape functions are defined as tensor products of 1D shape functions, same way as for the standard quadrilaterals, except for the logic on nodes and the corresponding nodal connectivities. In 2D, each infinite element has only two vertex nodes, one mid-edge node on the truncating circle, the middle node, and two mid-edge nodes on the radial edges originating from the vertex nodes on the truncating circle.

Thus, one way to code the infinite elements, would be to add a new element type to the code, and develop then all necessary data structure, constrained approximation (in 3D only), mesh refinements, and mesh optimization routines.

We choose a simpler way, based on treating the infinite elements as an (implicit) form of an Absorbing Boundary Condition (ABC). Instead of setting up a permanent data structure, we generate a *temporary data* every time, the problem on the FE/IE mesh is solved. The resulting infinite element d.o.f. are then stored in the temporary data structure arrays, and can be used for visualization or postprocessing. The moment, however, any finite element adjacent to the truncating surface is modified, the content of the IE data structure arrays becomes meaningless.

The temporary IE data structure arrays, stored in *module/infinite_elements*, include three arrays. Arrays *IELEMS* and *IENODS* store new objects:

- type *infinite element*,
- type *infinite element node*.

The attributes of an infinite element include,

- the element nodes listed in the order: middle node, left radial mid-edge node, right radial mid-edge node, the mid-edge node on the truncating circle, left vertex, and right vertex nodes,
- the orientation of the mid-edge node on the truncating circle.

The attributes of an infinite element mode are similar to those of the standard nodes. The include node type (*medg* or *mdlq*), order of the node. and the corresponding degrees of freedom *zdofs*.

The third, integer array, *ELTOIE*, is dimensioned with *MAXNODS* - the maximum number of FE higher order nodes. The array is populated with zeros except for entries corresponding to middle nodes of elements adjacent to the truncating surface which store the adjacent infinite element numbers.

Except for the data structure module, all routines related to infinite elements are stored in directory *infel*.

The data structure arrays are allocated and filled up with the relevant information in routine *infel/initiate_infinite_elements*. The edges located on the truncating circle are flagged in the input file with (boundary condition) *BC flag* = 9. The infinite elements are generated then in the order in which the edges with the *BC flag* = 9, are encountered, following the *natural order of elements* supported by routine *datstrs/nelcon*. In other words, in presence of infinite elements, the natural order of elements is modified, by "inserting" infinite elements after the adjacent finite elements.

In presence of infinite elements, the interface with the frontal solver must be modified. The alternative versions of interface routines are stored in files *infel/solve1EM*, *infel/solin2IE*, and *infel/soloutIE*. Please visit the source code for details.

Choice of radial order N. The infinite elements in the initial mesh are assumed to be *isotropic*, i.e. order N in the radial direction is set to the corresponding order p of the edge on the truncating circle. Usually we begin with elements of second order.

During the hp-refinements, edges on the truncating circle get p- or h-refined. Every time, the edge is p-refined, its IE radial order is updated, i.e. N is increased to $N+1$. We also increase the IE order when the edge is h-refined. Therefore, in presence of h-refinements, we encounter infinite elements with radial order N *greater* than the FE order p. This reflects the philosophy that any improvement in the approximation properties on the truncating circle, should be accompanied with the corresponding improvement in the radial direction as well.

Calculation of infinite element stiffness matrix. Routine *infel/eleminf*. As discussed in the previous section, the infinite elements shape functions are tensor products of one-dimensional FE shape functions and IE *radial* shape functions. The parametrization of the truncating circle giving rise to the system of coordinates (16.1.4) is replaced with its isoparametric FE approximation,

$$\boldsymbol{x}_e(\xi_1) = \boldsymbol{x}_1(1-\xi_1) + \boldsymbol{x}_2\xi_1 + \sum_{j=1}^{p-1} \boldsymbol{x}_{3,j}\hat{\chi}_{3,j}(\xi_1)\ .$$

Here $\hat{\chi}_{3,j}(\xi_1)$ denote the mid-edge node (1D master element) shape functions of order p, $\boldsymbol{x}_1, \boldsymbol{x}_2$ are the vertex nodes coordinates, and $\boldsymbol{x}_{3,j}$ denote the mid-edge node geometry d.o.f.

The evaluation of the stiffness matrix follows formula (16.1.18),

$$\int_0^1\int_0^1 \left\{ a^{-1}\left[\xi_2^2 \frac{\partial U}{\partial \xi_2}\frac{\partial \bar{V}}{\partial \xi_2} + \alpha(\alpha-1)U\bar{V} - 2ika\frac{\partial U}{\partial \xi_2}\bar{V}\right] + a\frac{\partial U}{\partial \xi_1}\frac{\partial \bar{V}}{\partial \xi_1}\left(\frac{ds}{d\xi_1}\right)^{-2}\right\}\frac{ds}{d\xi_1}\,d\xi_1 d\xi_2$$
$$+\frac{\alpha+ika}{a}\int_0^1 U(1,\cdot)\bar{V}(1,\cdot)\,\frac{ds}{d\xi_1}\,d\xi_1 d\xi_2\ , \tag{16.3.30}$$

with solution U and test function V replaced with element shape functions, and $x=\xi_2$. In a similar way, formula (16.2.27) is used to evaluate the stiffness matrix for the Petrov-Galerkin case.

In both cases, the stiffness matrix is integrated in the standard way using Gauss-Legendre quadrature points, see the infinite element routine *infel/eleminf*. The infinite element shape functions are evaluated in routine *infel/shape2IE* as tensor products of 1D FE shape functions and IE radial shape fucntions (integrals of Jacobi polynomials) evaluated in routine *infel/shape1IE*.

The remaining routines follow the logic of finite elements. Change of sign due to possible different orientations of the mid-edge node on the truncating circle is taken into account into routine *infel/celeminf* which evaluates the modified stiffness matrix, and local IE d.o.f. are returned by routine *infel/solelmIE*. Finally, routine *infel/displayIE* displays a portion of the infinite elements corresponding to $x=\xi_2>0.5$, allowing for a visualization of the IE mesh and the corresponding solution.

16.4 Calculation of Echo Area

Direct evaluation using the IE solution. The simplest way to evaluate EA is based on using the shape functions (16.2.22). More precisely, if the direction $\hat{\boldsymbol{x}}$ intersects with a finite element edge on the truncating circle, and $\hat{\xi}$ is the value of the corresponding parameter, i.e.,

$$\frac{\boldsymbol{x}_a(\hat{\xi})}{|\boldsymbol{x}_a(\hat{\xi})|} = \hat{\boldsymbol{x}} \, ,$$

the Echo Area (16.0.2) is evaluated by substituting in (16.2.22) $x = 0$,

$$|\sum_{j,l} U_{jl} e_j(\hat{\xi}) \psi_l(0)| \, |\boldsymbol{x}_a(\hat{\xi})|^{\alpha} \, . \tag{16.4.31}$$

Here, the summation extends over all IE shape functions (d.o.f.) denumerated with the double index (j, l).

Evaluation through postprocessing. We shall limit our discussion to the 2D case. Any function u that satisfies the Helmholtz equation in a domain exterior to a closed boundary Γ, along with the Sommerfeld radiation condition, satisfies automatically the *Helmholtz representation formula*,

$$u(\boldsymbol{x}) = \int_{\Gamma} \left\{ -\frac{\partial u}{\partial n}(\boldsymbol{y}) \Phi(\boldsymbol{x}, \boldsymbol{y}) + u(\boldsymbol{y}) \frac{\partial \Phi}{\partial n}(\boldsymbol{x}, \boldsymbol{y}) \right\} \, ds(\boldsymbol{y}) \, . \tag{16.4.32}$$

Here $\boldsymbol{n} = \boldsymbol{n}(\boldsymbol{y})$ is the unit normal directed to infinity, and Φ denotes the fundamental solution to the Helmholtz operator,

$$\Phi(\boldsymbol{x}, \boldsymbol{y}) = \Phi(k|\boldsymbol{x} - \boldsymbol{y}|) \, ,$$

where,

$$\Phi(kr) = \frac{H_0^{(2)}(kr)}{4i} \, . \tag{16.4.33}$$

Here $H_0^{(2)}$ denotes the Hankel function of the second type, of order 0. Notice that changing the ansatz in time to $e^{-i\omega t}$, requires switching to the Hankel function of the first type.

Evaluating the normal derivative of the fundamental solution,

$$\frac{\partial \Phi}{\partial n}(\boldsymbol{x}, \boldsymbol{y}) = k \, \Phi'(k|\boldsymbol{x} - \boldsymbol{y}|) \, \underbrace{\boldsymbol{n}(\boldsymbol{y}) \cdot \frac{\boldsymbol{x} - \boldsymbol{y}}{|\boldsymbol{x} - \boldsymbol{y}|}}_{\cos \beta_y} \, ,$$

we get,

$$u(\boldsymbol{x}) = \int_{\Gamma} \left\{ -\frac{\partial u}{\partial n}(\boldsymbol{y}) \Phi(|\boldsymbol{x} - \boldsymbol{y}|) + k \, u(\boldsymbol{y}) \Phi'(k|\boldsymbol{x} - \boldsymbol{y}|) \cos \beta_y \right\} \, ds(\boldsymbol{y}) \, . \tag{16.4.34}$$

Using the far field approximation,

$$|\boldsymbol{x}-\boldsymbol{y}| = |\boldsymbol{x}| - \boldsymbol{y}\cdot\hat{\boldsymbol{x}}\,,$$

and the asymptotic formula for the Hankel function and its derivative [1],

$$\Phi(kr) = i\sqrt{\frac{2}{\pi kr}}e^{-ikr}, \quad \Phi'(kr) = \sqrt{\frac{2}{\pi kr}}\,e^{-ikr},$$

we obtain,

$$\lim_{|\boldsymbol{x}|\to\infty} |\boldsymbol{x}|^{\frac{1}{2}}|u(\boldsymbol{x})| = \frac{1}{4}\sqrt{\frac{2}{\pi k}}\left|\int_\Gamma \left\{-\frac{\partial u}{\partial n}(\boldsymbol{y}) + ik\,u(\boldsymbol{y})\boldsymbol{n}\cdot\hat{\boldsymbol{x}}\right\} e^{ik(\boldsymbol{y}\cdot\hat{\boldsymbol{x}})}\,ds(\boldsymbol{y})\right| . \qquad (16.4.35)$$

The integration can take place over *any contour* surrounding the scatterer. In context of the IE discretization, it is natural to integrate over the element edges adjacent to the truncating circle.

The two different ways to evaluate the Echo Area have been implemented in routines *infel/far_field* and *infel/far_field2*.

16.5 Numerical experiments

Evaluation of the error. In all reported experiments, the error is computed in the H^1-semi-norm, integrated *only* over the near-field domain. This is in line with treating the infinite element as an implicit implementation of an ABC only. The error is reported in percent of the semi-norm of the solution, defined over the same domain. The exact solution for the cylinder problem is computed using the standard formula involving Hankel functions [83], see also Exercise 16.3.

For the wedge problem, the unknown solution is replaced with the solution on the hp-refined mesh.

Selection of radial shape functions. Conditioning. We do not report any extensive experiments related to conditioning. As a simple illustration supporting the idea of using integrated Jacobi polynomials, we have computed only the condition number for the one-dimensional stiffness matrix corresponding to (16.1.16) and $\alpha = 1/2$, preconditioned with the inverse of the corresponding diagonal matrix. More precisely, we have computed,

$$\mathrm{cond}\boldsymbol{B} = \frac{\lambda^{\frac{1}{2}}_{max}(\boldsymbol{B}\bar{\boldsymbol{B}}^T)}{\lambda^{\frac{1}{2}}_{min}(\boldsymbol{B}\bar{\boldsymbol{B}}^T)}\,, \qquad (16.5.36)$$

where $\boldsymbol{B} = (\mathrm{diag}\boldsymbol{A})^{-1}\boldsymbol{A}$, with $\boldsymbol{A}$ denoting the complex-valued matric corresponding to sesquilinear form (16.1.16), and a particular choice of shape functions. $\mathrm{diag}\boldsymbol{A}$ denotes the diagonal of matrix $\boldsymbol{A}$, and λ_{max} and λ_{min} denote the largest and the smallest eigenvalues of the corresponding

matrices. Three choices of radial shape functions were investigated. Along with the integrated Jacobi polynomials discussed earlier, we have also tried simple monomials, and integrated Legendre polynomials,

$$\psi_j(x) = \begin{cases} 1 & j = 0 \\ x^j - 1 & j > 1 \end{cases} \qquad \psi_j(x) = \begin{cases} 1 & j = 0 \\ \int_x^1 L_{j-1}(t)\, dt & j > 1 \end{cases} \tag{16.5.37}$$

Results presented in Fig. 16.2 support the idea of orthogonalizing the leading term in formula (16.1.16), although they are a little bit surprizing. For higher values of order N, the integrated Legendre polynomials perform slightly better than the integrated Jacobi polynomials. Both Legendre and Jacobi polynomials outperform significantly simple monomials. Preconditioning with the

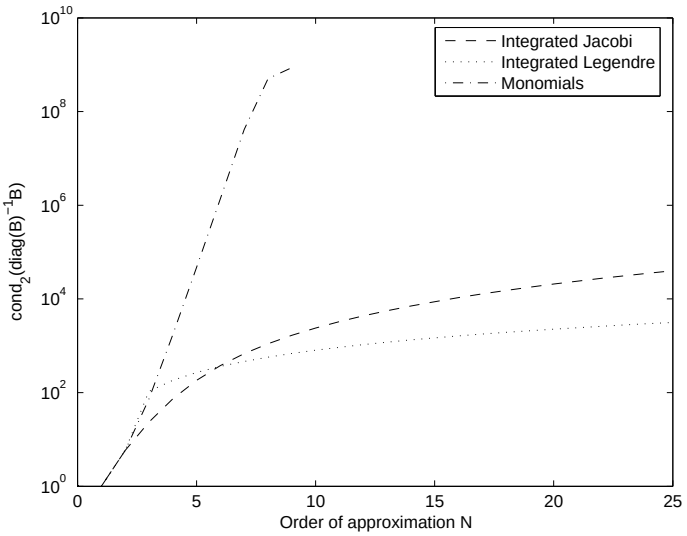


Figure 16.2: Condition number (16.5.36) for three different choices of radial shape functions with $ka = \pi$

inverse of the diagonal matrix does not eliminate the effect of scaling of the shape functions. In the reported experiment, the Legendre polynomials were scaled the standard way: $L_j(1) = 1$, and the *integrated* Jacobi polymials were scaled to 1 at $x = 0$. An optimal scaling may result in an additional improvement in conditioning.

Scattering of a plane wave on the rigid cylinder. We begin with the standard verification test - scattering of a plane wave by a unit cylinder [83]. We set the wave number to $k = \pi$, and truncate the infinite domain with a circle of radius $a = 3$. Fig. 21.2 displays convergence history for p-uniform and hp-adaptive refinements, starting with a mesh of 16 quadratic elements. The horizontal axis corresponds to the number of d.o.f. n displayed on the algebraic scale $n^{1/3}$.

As expected, the uniform p refinements deliver exponential convergence, with the adaptive refinements delivering slightly smaller error but the same rates. Fig. 21.3 shows the optimal hp mesh,

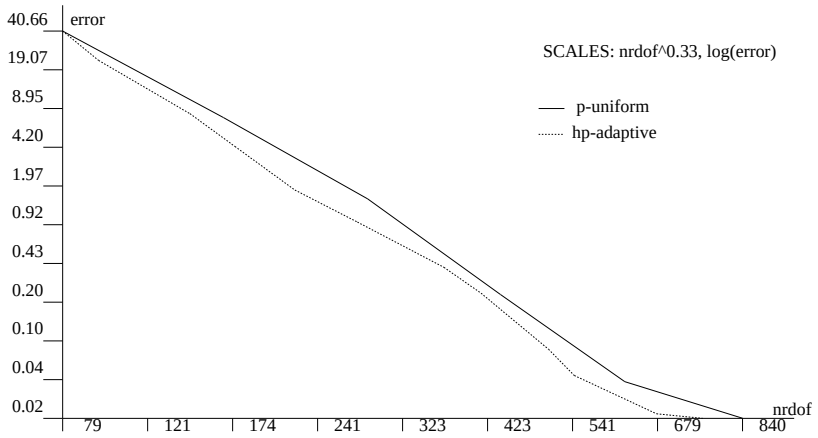


Figure 16.3: Scattering of a plane wave on a cylinder. Convergence history for p-uniform and hp-adaptive refinements

corresponding to an error of 0.025 percent.

Fig. 21.4 presents contour lines of the real part of the error function, for a uniform mesh of quartic elements. The values, indicated by the color scale on the right, range from -0.006 to 0.006. Along with the FE mesh, the graph displays a portion of the infinite elements corresponding to $0.5 < x < 1$. The solution in the IE domain seems to be actually better than in the finite element domain which indicates that lower order infinite elements would have been sufficient. Finally, for the same mesh of quartic elements (1.5 percent error), Fig. 16.6 presents the monostatic EA corresponding to the range of 180 to 0 degrees (left to right), displayed in dB (20 log of the actual value) versus the angle. Both methods for computing EA yield graphically indiscernible results (the first three significant digits are identical). The derivatives in formula (16.4.35) have been averaged over the adjacent finite and infinite elements. With the derivatives evaluated using only the contributions from either finite or infinite elements, the corresponding value of EA is less accurate. The slight variations of the EA for the cylinder, reflect the imperfect approximation of the geometry due to the use of isoparametric elements (for exact geometry elements, the EA would have been constant).

Comparison of Bubnov- and Petrov-Galerkin formulations. Both methods deliver practically identical results. The automatically generated hp meshes are identical and the corresponding convergence graphs are visually indiscernible. Same conclusions apply to the wedge problem discussed

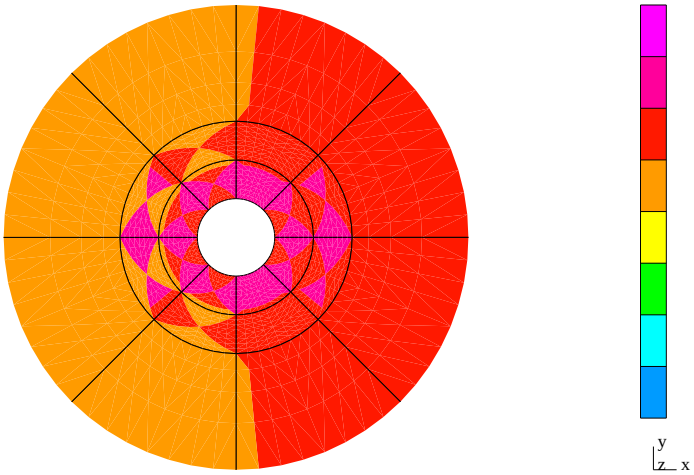


Figure 16.4: Scattering of a plane wave on a cylinder. Optimal hp mesh corresponding to 0.025 percent error

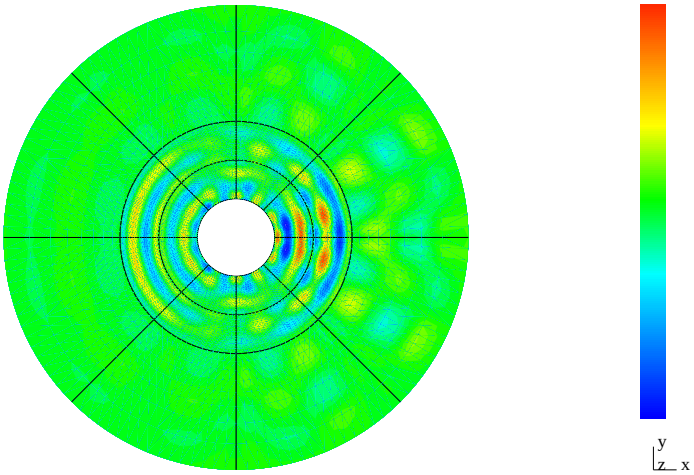


Figure 16.5: Scattering of a plane wave on a cylinder. Real part of the error function for a uniform mesh of quartic elements

next.

Scattering of a plane wave on a wedge. The second and final example deals with the resolution of singularities, and its impact on EA computations. We have divided the cylinder from the previous

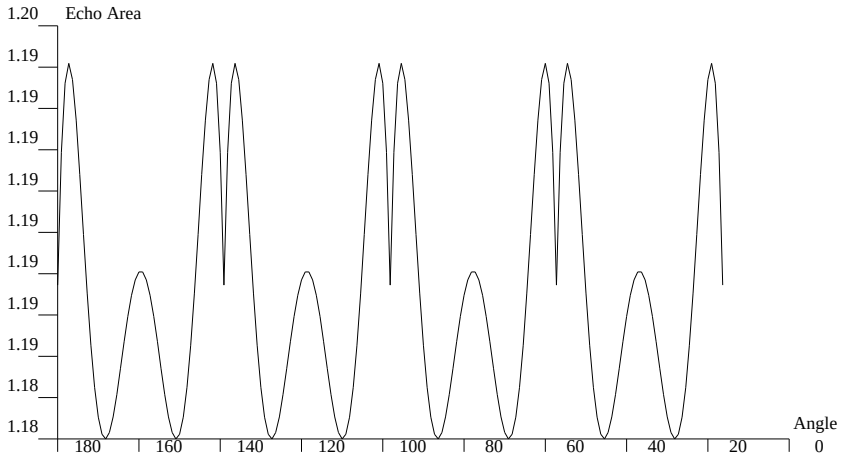


Figure 16.6: Scattering of a plane wave on a cylinder. EA for a uniform mesh of quartic elements (1.5 percent error)

example into four equal quadrants and kept just one of them. We set the wave number $k = \pi$, i.e. the distance between the truncating boundary and the object is equal to one wavelength. We start with an example of a typical, automatically obtained hp mesh corresponding to the incident wave coming from the right (angle θ = 0), and rather academic error level of 0.1 percent. Figures 21.5 and 21.6 present the optimal mesh, with three consecutive zooms showing details of the mesh around the lower corner. Fig. 21.7 presents convergence history for the problem using either Bubnov-

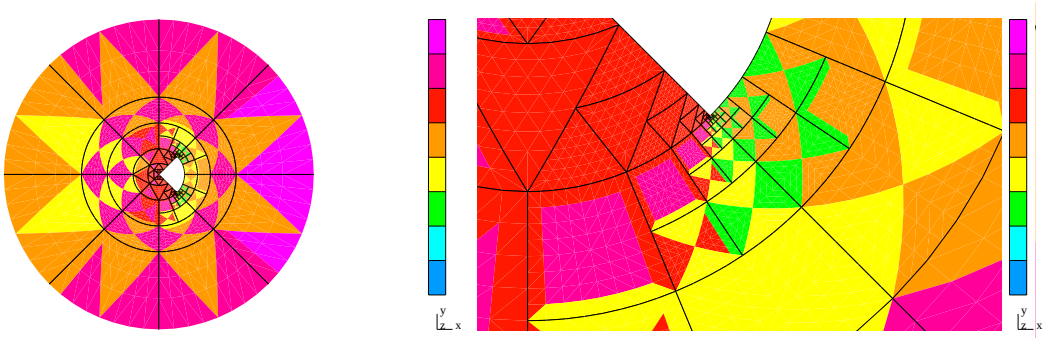


Figure 16.7: Scattering of a plane wave on a wedge, $\theta = 0$. Optimal hp mesh for 0.1 percent error, with a 10 times zoom on the lower corner

and Petrov-Galerkin formulations. Similarly to the cylinder problem, the results are visually indiscernible.

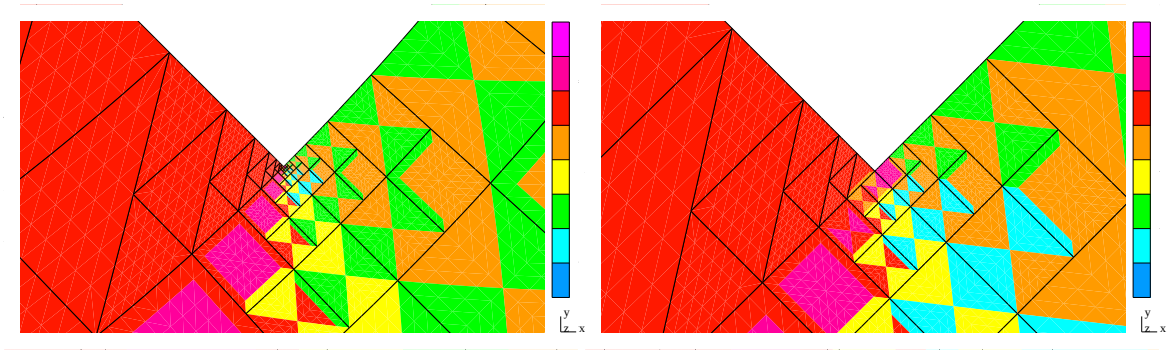


Figure 16.8: Scattering of a plane wave on a wedge, $\theta = 0$. Optimal hp mesh for 0.1 percent error. Zooms on the lower corner with 100 and 1000 magnifications

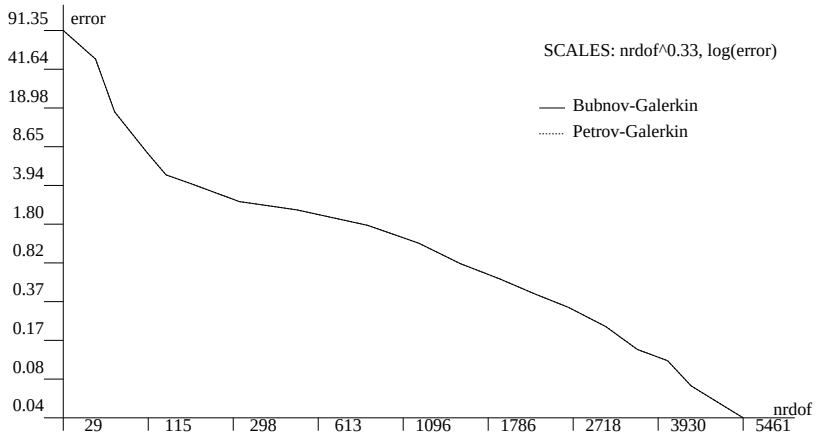


Figure 16.9: Scattering of a plane wave on a wedge. Convergence history for adaptive hp refinements using Bubnov- and Petrov-Galerkin IE formulations

Does the resolution of singularities affect the quality of the Echo Area ? We come to the final experiment reflecting the impact of adaptivity on evaluation of Echo Area. Fig. 16.10 compares EA for the wedge problem evaluated using a uniform mesh of quartic elements and hp-adaptive meshes. The choice of the uniform meshes reflects the usual practice of selecting a mesh that reproduces the wave form of the solution (two quartic elements per wavelength) and delivers an error in the range of 3-4 percent. The hp meshes were obtained by requesting a two percent error level, at which several levels of hp-refinements resolve the structure of the singularities in the solution. For each direction of the incoming wave ($\theta = 180, 179, \ldots, 0$, left to right), the hp-adaptive algorithm was run starting with the optimal mesh for the previous direction, with the optimization procedure restarted from the

initial mesh every 10 degrees. Except for a slight shift in the EA level, the results are practically

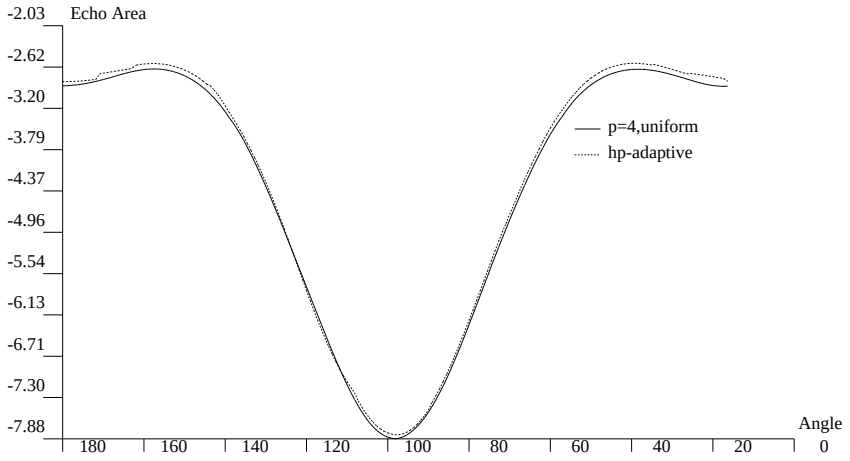


Figure 16.10: Scattering of a plane wave on a wedge. EA for the uniform mesh of quartic elements (3-4 percent error range level) and hp-adaptive mesh (2 percent error)

identical. Resolution of the singularities seems to have no impact on quality of the EA computations.

We do mention, however, that for the same error level of roughly 3.5 percent, and the quality of the corresponding EA, the adaptive algorithm delivers meshes roughly half the size of the uniform meshes. Fig. 16.11 presents an example of such a mesh for $\theta = 90^o$. The optimal distribution of approximation order p is not uniform. The fact that the algorithm has not selected any h-refinements, indicates that the resolution of the wave structure of the solution is more important (in terms of the energy error) than a resolution of the singularities.

Evaluation of EA. As mentioned earlier, both methods of evaluating the Echo Area, delivered practically identical results for uniform meshes with energy error level of 4 to 1 percent. For non-uniform, adaptive meshes, the difference between the two methods is slightly more pronounced. Fig. 16.12 provides a more precise information showing the evolution of the error in EA for the cylinder in terms of the problem size for the sequence of optimal hp meshes. The results are plotted using the algebraic and logarithmic scales, with the error reported on dB scale in percent of the exact value. Comparing with the energy error reported in Fig. 21.2, we see that indeed at the level of two percent energy error, the corresponding error in EA (in dB) for both methods drops below 1 percent (first two digits correct) which makes again the results graphically indiscernible on the EA graph. Stable exponential convergence in terms of energy does not translate, however, into a similar exponential convergence in the quantity of interest - the Echo Area.

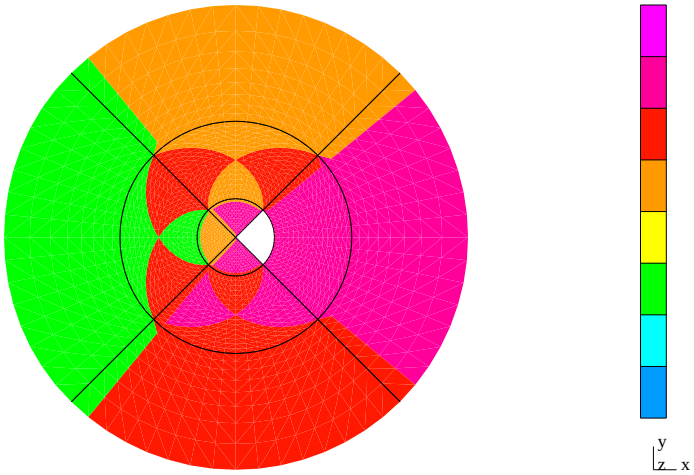


Figure 16.11: Scattering of a plane wave on a wedge, $\theta = 90^o$. Optimal hp mesh delivering 3.5 % error

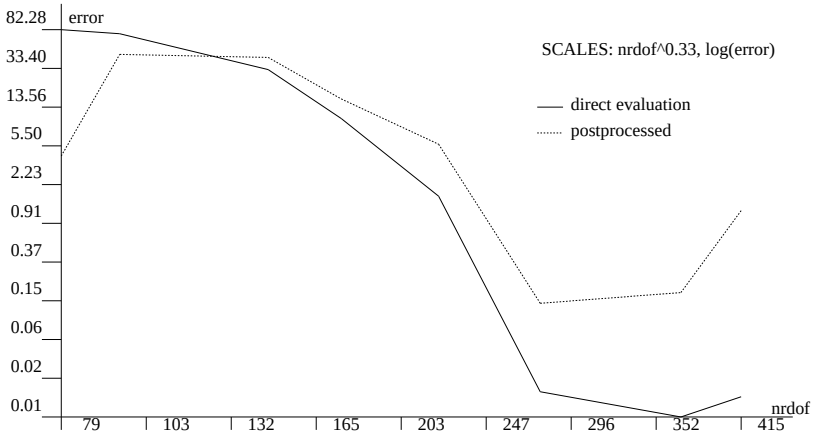


Figure 16.12: Scattering of a plane wave on a cylinder. Error in EA vs. problem size

16.6 Comments

The idea of coupled Finite Element (FE)/Infinite Element (IE) approximations for exterior wave propagation problems dates back to the pioneering contributions of Bettess and Zienkiewicz, see [23] and the literature cited therein.

The works of Astley et al. [14], Cremers *et al.* [34], Givoli [72] and many others recognized the spectral character of the approximation and pointed to the necessity of multipole expansions. Burnett [26] revolutionized the approach from the practical point of view, by introducing a new, *symmetric* unconjugated formulation, and using prolate and oblate spheroidal elements, we refer to [118] for an extensive review on the subject.

Contrary to the concept of Perfectly Matched Layer [22], and other techniques based on Absorbing Boundary Conditions (ABC's), the conjugated element of Astley et al. [14], aimed at obtaining the solution in the whole unbounded domain.

For Laplace equation (coercive problems in general), convergence analysis for coupled FE/IE discretizations reduces to the proof of approximability, see [71]. With the stability granted, the IE discretization in spherical or ellipsoidal coordinates, can be replaced with techniques better suited for more complex geometries, and adaptivity, see e.g. [25] for the idea of *inverted finite elements*. The energy (L^2 norm of the solution gradient) is finite which allows for a standard Bubnov-Galerkin discretization. It is perhaps worth mentioning that, for unbounded domains, L^2-integrability of derivatives *does not* imply the L^2-integrability of the function, solution of the Laplace exterior problem "lives" in a weighted L^2 space only.

Helmholtz equation is dramatically different. The energy of the solution is no longer finite. This is not a surprise; solution to the Helmholtz equation is the steady-state solution corresponding to a time-harmonic forcing term and infinite time. For each finite time interval, the energy "pumped" into the system is finite but, over the infinite time, becomes infinite. This prohibits the use of identical trial and test functions in the standard Bubnov-Galerkin method, if the energy and mass terms in the variational formulation are to be Lebesgue integrable. Petrov-Galerkin formulation of Astley at al. [14] becomes the natural choice.

The unconjugated Bubnov-Galerkin symmetric formulation of Burnett was possible due to the interpretation of the integrals in the Cauchy Principal Value (CPV) sense. The presentation in this chapter follows closely [59] where we have explored a similar idea for conjugated elements.

Convergence analysis for the exterior Helmholtz problem is much more challenging than for the Laplace equation. Due to the unboundedness of the domain, compact perturbation argument does not work. One possibility of analyzing the convergence is to give up on the analysis in the whole exterior domain, and treat the infinite element as an implicit ABC. The exact Dirichlet-to-Neumann (DtN) operator is then replaced with its approximate IE counterpart. This was the idea behind the analysis presented in [47] where sufficient conditions for convergence, expressed in terms of eigenvalues of the approximate DtN operator, were formulated[2]. Those eigenvalues were determined only numerically and, in a strict mathematical sense, the proof of convergence for the infinite elements remains an open problem.

Spectral discretizations are much more sensitive to conditioning problems, and infinite elements

[2]Only the case of spherical infinite elements was analyzed.

of higher order are not an exception. Conditioning problems have been reported by many authors, see e.g. [13, 111]. Similar problems have been encountered in context of using infinite elements for solving exterior Maxwell problems and determining Radar Cross Sections (RCS) [124].

Exercises

Exercise 16.1 Demonstrate that the Sommerfeld radiation condition is *independent* of the choice of the origin of the cylindrical (spherical) system of coordinates. In other words, if a function u satisfies the radiation condition in a system with origin at point $\boldsymbol{x}_1$, it will aso satisfy it in the system with the origin at any other point $\boldsymbol{x}_2$.

Exercise 16.2 Let $w \in L^2(\Omega)$ where Ω is the exterior of a sphere,

$$\Omega = \{\boldsymbol{x} \in I\!R^n \ : \ |\boldsymbol{x}| > a\} \, .$$

Prove that,

$$\lim_{R\to\infty} \int_{S_R} |w|^2 \, dS_R = 0 \, .$$

Exercise 16.3 Derive the exact solution for the problem of scattering a plane wave on a rigid cylinder of radius a. Proceed with the following steps.

- Use separation of variables to derive the general solution to the Helmholtz equation in the exterior cylindrical domain, satisfying the radiation condition,

$$u(r,\theta) = \sum_{n=0}^{\infty} H_n^{(2)}(kr)(C_n \cos n\theta + D_n \sin n\theta) \, . \tag{16.6.38}$$

  Here $H_n^{(2)}$ denotes the Hankel function of the second type, of order n. How the solution would change, if we had used the $e^{-i\omega t}$ ansatz in time ?

- Consult [1] for the following representation of the plane wave incoming from the x direction, in terms of the Bessel functions (the so called *addition theorem*),

$$e^{-ikx_1} = e^{-kr\cos\theta} = \sum_{n=0}^{\infty} (-1)^n \epsilon_n i^n J_n(kr) \cos n\theta \, , \tag{16.6.39}$$

  where,

$$\epsilon_n = \begin{cases} 1 & n = 0 \\ 2 & n > 1 \end{cases}$$

- Use the Neumann boundary condition (16.0.1) to determine the final form of the solution,

$$u(r,\theta) = \sum_{n=0}^{\infty} (-1)^{n+1} \epsilon_n i^n \frac{J_n'(ka)}{H_n^{(2)\prime}(ka)} H_n^{(2)}(kr) \cos n\theta \,. \tag{16.6.40}$$

Exercise 16.4 *Jacobi polynomials* $P_n^{(\alpha,\beta)}$, defined on interval $(-1,1)$ are members of a large class of polynomials forming orthogonal sequences in various weighted L^2 products, see [1], page 778. Orthogonal polynomials and their integrals are natural candidates for hierarchical shape functions. Jacobi polynomials are given by the formula,

$$P_n^{(\alpha,\beta)}(x) = \frac{1}{2^n} \sum_{m=0}^{n} \left(\begin{array}{c} n+\alpha \\ m \end{array} \right) \left(\begin{array}{c} n+\beta \\ n-m \end{array} \right) (x+1)^m (x-1)^{n-m} \,.$$

Here $\left(\begin{array}{c} n \\ m \end{array} \right)$ denotes the Newton symbol,

$$\left(\begin{array}{c} n \\ m \end{array} \right) = \frac{n!}{m!(n-m)!}, \quad m = 0, \ldots, n \,,$$

and coefficients $\alpha, \beta > -1$.

Prove that Jacobi polynomials are orthogonal in the weighted L^2 product,

$$(f,g) := \int_{-1}^{1} (1+x)^\beta (1-x)^\alpha f(x) g(x)\, dx \,.$$

After the change of coordinates $x \stackrel{'}{=} (2x-1)$, the formula for the Jacobi polynomials and the corresponding inner product transform into,

$$P_n^{(\alpha,\beta)}(x) = \sum_{m=0}^{n} \left(\begin{array}{c} n+\alpha \\ m \end{array} \right) \left(\begin{array}{c} n+\beta \\ n-m \end{array} \right) x^m (x-1)^{n-m} \,,$$

$$(f,g) := 2 \int_{0}^{1} x^\beta (1-x)^\alpha f(x) g(x)\, dx \,.$$

Part III

TWO-DIMENSIONAL MAXWELL PROBLEMS

Chapter 17

Two-Dimensional Maxwell Equations

In the third part of this volume, we focus on two-dimenional Maxwell equations. Fascinating for themselves, Maxwell's equations provide probably the most important example of a boundary-value problem for which the natural energy space is the space $H(\mathrm{curl}, \Omega)$ - the space of all square integrable functions for which only *some* combinations of first derivatives, namely the components of curl, are square integrable. We shall begin with a short introduction to Maxwell's equations and discuss possible variational formulations.

Maxwell's equations involve the curl operator,

$$\boldsymbol{\nabla} \times \boldsymbol{E} = \left(\frac{\partial E_3}{\partial x_2} - \frac{\partial E_2}{\partial x_3}, \frac{\partial E_1}{\partial x_3} - \frac{\partial E_3}{\partial x_1}, \frac{\partial E_2}{\partial x_1} - \frac{\partial E_1}{\partial x_2} \right) .$$

For a vector-valued two-dimensional field $\boldsymbol{E} = (E_1, E_2, 0)$, with components depending only upon x_1, x_2, only the third component of the three-dimensional curl operator is different from zero,

$$\boldsymbol{\nabla} \times \boldsymbol{E} = \left(0, 0, \frac{\partial E_2}{\partial x_1} - \frac{\partial E_1}{\partial x_2} \right) .$$

We shall identify it as a two-dimensional scalar-valued curl operator,

$$\mathrm{curl}\, \boldsymbol{E} = \frac{\partial E_2}{\partial x_1} - \frac{\partial E_1}{\partial x_2} .$$

Similarly, for a field with only the third component different from zero, $\boldsymbol{E} = (0, 0, E_3)$, we get,

$$\boldsymbol{\nabla} \times \boldsymbol{E} = \left(\frac{\partial E_3}{\partial x_2}, -\frac{\partial E_3}{\partial x_1}, 0 \right) .$$

This leads to the definition of the two-dimensional, vector-valued curl operator acting on a scalar argument,

$$\boldsymbol{\nabla} \times E = \left(\frac{\partial E}{\partial x_2}, -\frac{\partial E}{\partial x_1} \right) .$$

In 2D, therefore, we have two curl operators, operator curl setting vectors into scalars, and operator $\boldsymbol{\nabla}\times$, setting scalars into vectors. Noice that the vector-valued curl can be obtained by rotating the gradient operator by 90 degrees. More precisely,

$$\boldsymbol{\nabla}\times E = \boldsymbol{\nabla}\tilde{E}, \quad \text{where } \tilde{E}(x_1,x_2) = E(x_2,-x_1)\,.$$

17.1 Introduction to Maxwell's equations

We shall discuss the simplest (linear, isotropic) version of Maxwell's equations. It is more convenient from the algebraic point of view, to discuss the three-dimensional case first, and then specialize it to its two-dimensional version.

Given a domain $\Omega \subset \mathbb{R}^3$, we wish to determine electric field $\boldsymbol{E}(\boldsymbol{x})$ and magnetic field $\boldsymbol{H}(\boldsymbol{x})$ that satisfy:

- Faraday's law (1831),
$$\boldsymbol{\nabla}\times\boldsymbol{E} = -\frac{\partial}{\partial t}(\mu\boldsymbol{H})\,,$$
- Ampere's law (1820) with Maxwell's correction (1856),
$$\boldsymbol{\nabla}\times\boldsymbol{H} = \boldsymbol{J}^{imp} + \sigma\boldsymbol{E} + \frac{\partial}{\partial t}(\epsilon\boldsymbol{E})\,.$$

Here μ, σ, ϵ denote the material data: permeability, conductivity and permittivity, assumed to be piecewise constant. $\boldsymbol{B} := \mu\boldsymbol{H}$ is the magnetic flux, $\boldsymbol{D} := \epsilon\boldsymbol{E}$ is the electric flux, $\sigma\boldsymbol{E}$ is the Ohm current, and $\boldsymbol{J}^{imp}$ denotes a prescribed, given impressed current, with $\boldsymbol{J} := \boldsymbol{J}^{imp} + \sigma\boldsymbol{E}$ identified as the total current.

Once the total current $\boldsymbol{J}$ has been determined, we can use the continuity (conservation of free charge) equation, to determine the corresponding free charge density ρ,

$$\boldsymbol{\nabla}\cdot\boldsymbol{J} + \frac{\partial\rho}{\partial t} = 0\,.$$

The first order Maxwell system is accompanied with initial, boundary, and interface (across material discontinuities) conditions. In the case of exterior domains, the system is completed with the Silver - Müller radiation condition, a generalization of the Sommerfeld radiation condition, expressing the fact that waves must propagate toward infinity only.

Taking divergence of both sides in Faraday's equation, we learn that the magnetic field $\boldsymbol{H}$ has to satisfy (automatically) the equation,

$$\frac{\partial}{\partial t}\left(\boldsymbol{\nabla}\cdot(\mu\boldsymbol{H})\right) = 0\,.$$

Assuming that the initial value $\boldsymbol{H}(0)$ satisfies the Gauss law for the magnetic flux,

$$\boldsymbol{\nabla} \circ (\mu \boldsymbol{H}) = 0 \,,$$

we learn that the law is satisfied automatically for all times t.

Similarly, taking the divergence of both sides of the Ampere equation, and utilizing the continuity law, we learn that the electric field $\boldsymbol{E}$ satisfies,

$$\frac{\partial}{\partial t} \left(\boldsymbol{\nabla} \circ (\epsilon \boldsymbol{E}) - \rho \right) = 0 \,.$$

Assuming that the initial value $\boldsymbol{E}(0)$ satisfies the Gauss law for the electric flux,

$$\boldsymbol{\nabla} \circ (\epsilon \boldsymbol{E}) = \rho \,,$$

we learn that the electric field satisfies the law at all times t.

In the steady state, the Maxwell system degenerates and decouples,

$$\begin{cases} \boldsymbol{\nabla} \times \boldsymbol{E} = 0 \\ \boldsymbol{\nabla} \times \boldsymbol{H} = \boldsymbol{J}^{imp} + \sigma \boldsymbol{E} \,. \end{cases}$$

The closing equation for electrostatics is provided either by the continuity equation, or by the Gauss law. In the case of a perfect dielectric, $\sigma = 0$, the distribution of free charge ρ must be prescribed. We can determine the corresponding electric field by solving the electrostatics equations,

$$\begin{cases} \boldsymbol{\nabla} \times \boldsymbol{E} = 0 \\ \boldsymbol{\nabla} \cdot (\epsilon \boldsymbol{E}) = \rho \,. \end{cases}$$

In the case of a conductor, $\sigma > 0$, the free charges move, and we cannot prescribe them. In the steady state case, ρ is independent of time, and the continuity equation provides a closing equation for the Faraday law,

$$\begin{cases} \boldsymbol{\nabla} \times \boldsymbol{E} = 0 \\ -\boldsymbol{\nabla} \cdot (\sigma \boldsymbol{E}) = \boldsymbol{\nabla} \cdot \boldsymbol{J}^{imp} \,. \end{cases}$$

Once the electric field is known, the resulting free charge density can be determined from the Gauss law. In view of the first equation, either set of the electrostatics equations is usually solved in terms of a scalar potential for the electric field.

In the case of a perfect conductor, $\sigma = \infty$, the corresponding electric field $\boldsymbol{E}$ vanishes, and the volume occupied by the perfect conductor is eliminated from the (computational) domain Ω.

The magnetostatics equations can be obtained by complementing the Ampere law with the Gaussian law for the magnetic flux.

$$\begin{cases} \boldsymbol{\nabla} \times \boldsymbol{H} = \boldsymbol{J}^{imp} + \sigma \boldsymbol{E} \\ \boldsymbol{\nabla} \cdot (\mu \boldsymbol{H}) = 0 \end{cases}$$

Once the electric field is known, the corresponding magnetic field is obtained by solving the magnetostatics equations. Because of the second equation, the problem is usually formulated in terms of a vector potential for the magnetic field.

Wave equation. In order to reduce the number of unknowns, the first order Maxwell system is usually reduced to a single (vector-valued) 'wave equation', expressed either in terms of $\boldsymbol{E}$, or $\boldsymbol{H}$. The choice is usually dictated by the boundary conditions, and the analysis of both systems is fully analogous. We shall focus on the electric field formulation,

$$\boldsymbol{\nabla} \times (\frac{1}{\mu}\boldsymbol{\nabla} \times \boldsymbol{E}) + \epsilon\frac{\partial^2 \boldsymbol{E}}{\partial t^2} + \sigma\frac{\partial \boldsymbol{E}}{\partial t} = -\frac{\partial \boldsymbol{J}^{imp}}{\partial t} \,. \tag{17.1.1}$$

Once the electric field has been determined, the Faraday equation can be integrated to find the corresponding magnetic field.

Time-harmonic wave equation. Assuming the anzatz,

$$\boldsymbol{E}(\boldsymbol{x}, t) = \Re(\boldsymbol{E}(\boldsymbol{x})e^{i\omega t}) \,, \tag{17.1.2}$$

we convert the wave equation into the 'reduced wave equation',

$$\boldsymbol{\nabla} \times (\frac{1}{\mu}\boldsymbol{\nabla} \times \boldsymbol{E}) - (\omega^2\epsilon - i\omega\sigma)\boldsymbol{E} = -i\omega\boldsymbol{J}^{imp} \,, \tag{17.1.3}$$

to be solved for the complex-valued *phasor* $\boldsymbol{E}(\boldsymbol{x})$. Alternatively, (17.1.3) can be obtained by applying Fourier transform to (17.1.1). The solution to the wave equation can then be obtained by applying the inverse Fourier transform,

$$\boldsymbol{E}(\boldsymbol{x}, t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i\omega t}\boldsymbol{E}(\boldsymbol{x}, \omega)\, d\omega \,.$$

Notice that the sign in the exponential in the inverse Fourier transform is consistent with that in the anzatz (17.1.2). In the EE literature, frequently, the opposite sign is assumed,

$$\boldsymbol{E}(\boldsymbol{x}, t) = \Re(\boldsymbol{E}(\boldsymbol{x})e^{-j\omega t}) \,, \tag{17.1.4}$$

with j denoting the imaginary unit. The sign in the anzatz affects the sign in impedance and radiation boundary conditions, and one has to always remember which anzatz is being used. Substituting $j = -i$, we can easily switch in between the two formulations.

Once the electric field has been determined, the corresponding magnetic field is computed from the time-harmonic version of the Faraday law:

$$\boldsymbol{\nabla} \times \boldsymbol{E} = -i\omega\mu\boldsymbol{H} \,. \tag{17.1.5}$$

We postpone the discussion on boundary conditions to the next section, after we study the fundamental "integration by parts formula" for Maxwell's equations.

17.2 Variational formulation

From now on, we shall focus on the time-harmonic Maxwell equations only. We begin with the fundamental "integration by parts" formula,

$$\int_\Omega (\boldsymbol{\nabla} \times \boldsymbol{E}) \boldsymbol{F}\, d\boldsymbol{x} = \int_\Omega \boldsymbol{E} (\boldsymbol{\nabla} \times \boldsymbol{F})\, d\boldsymbol{x} + \int_{\partial\Omega} (\boldsymbol{n} \times \boldsymbol{E}) \boldsymbol{F}_t \, .$$

Here $\boldsymbol{n}$ is the outward normal unit vector for boundary $\partial\Omega$, $\boldsymbol{F}_t = \boldsymbol{F} - (\boldsymbol{F} \cdot \boldsymbol{n})\boldsymbol{n}$ is the tangential component of vector $\boldsymbol{F}$ on the boundary and,

$$\boldsymbol{n} \times \boldsymbol{E} = \boldsymbol{n} \times \boldsymbol{E}_t \, ,$$

is the 'rotated' tangential component of $\boldsymbol{E}$. Obviously, $(\boldsymbol{n} \times \boldsymbol{E})\boldsymbol{F}_t = \boldsymbol{E}_t (\boldsymbol{n} \times \boldsymbol{F})$. Notice that there is no '-' sign typical for other Green's formulas.

Finite energy considerations lead us naturally to the assumption that both electric field and magnetic field are square integrable. In view of (17.1.5), and under assumption on boundednes of material data,

$$0 < \epsilon_{min} \leq \epsilon \leq \epsilon_{max} < \infty, \quad 0 < \mu_{min} \leq \mu \leq \mu_{max} < \infty, \quad 0 \leq \sigma \leq \sigma_{max} < \infty \, ,$$

this implies that electric field $\boldsymbol{E}$ comes from the $H(\mathrm{curl})$ space,

$$H(\mathrm{curl}, \Omega) = \{ \boldsymbol{E} \in \boldsymbol{L}^2(\Omega) \, : \, \boldsymbol{\nabla} \times \boldsymbol{E} \in \boldsymbol{L}^2(\Omega) \} \, .$$

The Green formula is telling us immediately the right type of continuity across material interfaces and interelement boundaries for a conforming FE discretization. Assuming that domain Ω consists of two disjoint parts $\Omega_i, i = 1, 2$ with an interface Γ, with a C^1 field $\boldsymbol{E}$ in either of the subdomains, we use the integration by parts formula to obtain,

$$\int_\Omega (\boldsymbol{\nabla} \times \boldsymbol{E}) \boldsymbol{\phi}\, d\boldsymbol{x} = \int_\Omega \boldsymbol{E} (\boldsymbol{\nabla} \times \boldsymbol{\phi})\, d\boldsymbol{x} + \int_\Gamma [\boldsymbol{n} \times \boldsymbol{E}] \boldsymbol{\phi}_t \, ,$$

for every C^1 test function $\boldsymbol{\phi}$ vanishing on $\partial\Omega$. Here $[\boldsymbol{n} \times \boldsymbol{E}]$ denotes the jump of the tangential component of $\boldsymbol{E}$ across the interface Γ. Consequently, the field $\boldsymbol{\nabla} \times \boldsymbol{E}$ is a function (regular distribution) if an only if the tangential component of $\boldsymbol{E}$ is continuous across the interface. Similar considerations for the magnetic field lead to the observation that also the tangential component of $\boldsymbol{H}$ must be continuous across the material interfaces. In view of (17.1.5) this implies the second interface condition for the electric field,

$$[\boldsymbol{n} \times \frac{1}{\mu} (\boldsymbol{\nabla} \times \boldsymbol{E})] = 0 \, .$$

Multiplying (17.1.3) with a (conjugated) test function $\boldsymbol{F}$, and integrating by parts, we obtain,

$$\int_\Omega \left\{ \frac{1}{\mu} (\boldsymbol{\nabla} \times \boldsymbol{E})(\boldsymbol{\nabla} \times \bar{\boldsymbol{F}}) - (\omega^2 \epsilon - i\omega\sigma) \boldsymbol{E} \bar{\boldsymbol{F}} \right\} d\boldsymbol{x} + \int_{\partial\Omega} \boldsymbol{n} \times \frac{1}{\mu} (\boldsymbol{\nabla} \times \boldsymbol{E}) \bar{\boldsymbol{F}}_t dS = -i\omega \int_\Omega \boldsymbol{J}^{imp} \bar{\boldsymbol{F}} d\boldsymbol{x}. \tag{17.2.6}$$

Recall that working with conjugated test functions that result in sesquilinear rather than bilinear forms in the variational formulation, is typical for complex-valued wave propagation problems, and consistent with the definition of inner product in a complex Hilbert space. It is only a matter of choice though and, as long as shape functions are real-valued, the bilinear and sesquilinear formulations yield identical systems of discrete equations.

We are ready now to discuss the most common boundary conditions.

Perfect Electric Conductor (PEC). As the electric field in a perfect conductor vanishes, and the tangential component of $\boldsymbol{E}$ must be continuous across material interfaces, the tangential component of $\boldsymbol{E}$ on a boundary adjacent to a perfect conductor must vanish,

$$\boldsymbol{n} \times \boldsymbol{E} = 0 \,,$$

compare Exercise 17.2. For scattering problems, the electric field is the sum of a known incident field $\boldsymbol{E}^{inc}$ and (to be determined) scattered field $\boldsymbol{E}^{s}$. The condition above then leads to a nonhomogeneous Dirichlet condition for the scattered field,

$$\boldsymbol{n} \times \boldsymbol{E}^{s} = -\boldsymbol{n} \times \boldsymbol{E}^{inc} \,.$$

Impressed surface current. The simplest way to model an antenna, is to prescribe an impressed surface current on the boundary of the domain occupied by the antenna,

$$\boldsymbol{n} \times \boldsymbol{H} = \boldsymbol{J}_S^{imp} \,.$$

In conjuction with (17.1.5), this leads to the Neumann (natural boundary) condition,

$$\boldsymbol{n} \times \frac{1}{\mu}(\boldsymbol{\nabla} \times \boldsymbol{E}) = -i\omega \boldsymbol{J}_S^{imp} \,.$$

Notice that the impressed surface current is always tangent to the boundary. For a particular case of $\boldsymbol{J}_S^{imp} = 0$ we speak of a magnetic symmetry wall condition .

Impedance boundary condition. This is the simplest, first order approximation to model reflection of the electric field from an interface with a conductor with large but nevertheless finite conductivity, comp. [115],

$$\boldsymbol{n} \times \frac{1}{\mu}(\boldsymbol{\nabla} \times \boldsymbol{E}) - i\omega\gamma \boldsymbol{E}_t = -i\omega \boldsymbol{J}_S^{imp} \,.$$

Here $\boldsymbol{E}_t = -\boldsymbol{n} \times (\boldsymbol{n} \times \boldsymbol{E})$ denotes the tangential component of $\boldsymbol{E}$.

Denoting by $\Gamma_D, \Gamma_N, \Gamma_C$ three disjoint components of boundary $\partial\Omega$ on which the Dirichlet, Neumann and Cauchy boundary conditions have been prescribed, we limit the test functions in (17.2.6) to those that satisfy the homogeneous PEC condition on Γ_D, and use Neumann and Cauchy

conditions to build the impressed surface currents into the formulation. Our final variational formulation reads as follows.

$$\left\{\begin{array}{l} \boldsymbol{E} \in H(\mathrm{curl},\Omega),\ \boldsymbol{n}\times\boldsymbol{E} = \boldsymbol{n}\times\boldsymbol{E}_0 \text{ on } \Gamma_D \\ \displaystyle\int_\Omega \left\{\frac{1}{\mu}(\boldsymbol{\nabla}\times\boldsymbol{E})(\boldsymbol{\nabla}\times\bar{\boldsymbol{F}}) - (\omega^2\epsilon - i\omega\sigma)\boldsymbol{E}\bar{\boldsymbol{F}}\right\} d\boldsymbol{x} + i\omega\int_{\Gamma_C}\gamma\boldsymbol{E}_t\bar{\boldsymbol{F}}_t\, dS \\ = -i\omega\displaystyle\int_\Omega \boldsymbol{J}^{imp}\bar{\boldsymbol{F}}\, d\boldsymbol{x} + i\omega\int_{\Gamma_N\cup\Gamma_C}\boldsymbol{J}_S^{imp}\bar{\boldsymbol{F}}\, dS \quad \text{for every } \boldsymbol{F}\in H(\mathrm{curl},\Omega),\ \boldsymbol{n}\times\boldsymbol{F} = 0 \text{ on } \Gamma_D\,. \end{array}\right. \tag{17.2.7}$$

We follow the classical lines to show that, conversely, any sufficiently regular solution to the variational problem satisfies the reduced wave equation, and the natural boundary conditions.

Weak form of the continuity equation. Employing a special test function, $\boldsymbol{F} = \boldsymbol{\nabla} q, q \in H^1(\Omega), q = 0$ on Γ_D, we learn that the solution to the variational problem satifies automatically the weak form of the continuity equation,

$$\begin{aligned} &\int_\Omega -(\omega^2\epsilon - i\omega\sigma)\boldsymbol{E}\boldsymbol{\nabla}\bar{q}\, d\boldsymbol{x} + i\omega\int_{\Gamma_C}\gamma\boldsymbol{E}_t\boldsymbol{\nabla}\bar{q}\, dS \\ &\quad = -i\omega\int_\Omega \boldsymbol{J}^{imp}\boldsymbol{\nabla}\bar{q}\, d\boldsymbol{x} + i\omega\int_{\Gamma_N\cup\Gamma_C}\boldsymbol{J}_S^{imp}\boldsymbol{\nabla}\bar{q}\, dS \qquad \text{for every } q\in H^1(\Omega),\ q = 0 \text{ on } \Gamma_D\,. \end{aligned} \tag{17.2.8}$$

Upon integrating by parts, we learn that solution $\boldsymbol{E}$ satifies the continuity equation,

$$\mathrm{div}\left((\omega^2\epsilon - i\omega\sigma)\boldsymbol{E}\right) = i\omega\,\mathrm{div}\boldsymbol{J}^{imp} \qquad (=\omega^2\rho)\,,$$

plus additional boundary conditions on Γ_N, Γ_C, and interface conditions across material interfaces.

Maxwell eigenvalue problem. Related to the time-harmonic problem (17.2.7) is the eigenvalue problem,

$$\left\{\begin{array}{l} \boldsymbol{E}\in H(\mathrm{curl},\Omega),\ \boldsymbol{n}\times\boldsymbol{E} = 0 \text{ on } \Gamma_D, \quad \lambda\in\mathbb{R} \\ \displaystyle\int_\Omega \frac{1}{\mu}(\boldsymbol{\nabla}\times\boldsymbol{E})(\boldsymbol{\nabla}\times\bar{\boldsymbol{F}})\, d\boldsymbol{x} = \lambda\int_\Omega \epsilon\boldsymbol{E}\bar{\boldsymbol{F}}\, d\boldsymbol{x} \quad \text{for every } \boldsymbol{F}\in H(\mathrm{curl},\Omega),\ \boldsymbol{n}\times\boldsymbol{F} = 0 \text{ on } \Gamma_D\,. \end{array}\right. \tag{17.2.9}$$

The curl-curl operator is self-adjoint, its spectrum consists of $\lambda = 0$ with an infinite-dimensional eigenspace consisting of all gradients $\boldsymbol{\nabla} p, p \in H^1(\Omega), p = 0$ on Γ_D, and a sequence of positive eigenvalues $\lambda_1 < \lambda_2 < \ldots \lambda_n \to \infty$ with corresponding eigenspaces of finite dimension. Only the eigenvectors corresponding to positive eigenvalues are physical. Repeating the reasoning with the substitution $\boldsymbol{F} = \boldsymbol{\nabla} q$, we learn that they satisfy automatically the continuity equation.

Stabilized variational formulation. The standard variational formulation (17.2.7) is *not* uniformly stable with respect to frequency ω. As $\omega \to 0$, we loose the control over gradients. This

corresponds to the fact that, in the limiting case $\omega = 0$, the problem is ill-posed as the gradient component remains undetermined. A remedy to this problem is to enforce the continuity equation explicitly at the expense of introducing a Lagrange multiplier p. The so called *stabilized variational formulation* looks as follows.

$$\left\{\begin{array}{l} \boldsymbol{E} \in H(\mathrm{curl},\Omega), p \in H^1(\Omega), \quad \boldsymbol{n}\times\boldsymbol{E} = \boldsymbol{n}\times\boldsymbol{E}_0, p = 0 \text{ on } \Gamma_D, \\ \displaystyle\int_\Omega \frac{1}{\mu}(\boldsymbol{\nabla}\times\boldsymbol{E})(\boldsymbol{\nabla}\times\bar{\boldsymbol{F}})d\boldsymbol{x} - \int_\Omega(\omega^2\epsilon - i\omega\sigma)\boldsymbol{E}\cdot\bar{\boldsymbol{F}}d\boldsymbol{x} + i\omega\int_{\Gamma_C}\gamma\boldsymbol{E}_t\bar{\boldsymbol{F}}_t\,dS \\ \displaystyle -\int_\Omega(\omega^2\epsilon - i\omega\sigma)\boldsymbol{\nabla}p\cdot\bar{\boldsymbol{F}}d\boldsymbol{x} = -i\omega\int_\Omega \boldsymbol{J}^{imp}\cdot\bar{\boldsymbol{F}}d\boldsymbol{x} + i\omega\int_{\Gamma_N\cup\Gamma_C}\boldsymbol{J}_S^{imp}\cdot\bar{\boldsymbol{F}}dS \\ \qquad\qquad \forall \boldsymbol{F}\in H(\mathrm{curl},\Omega),\ \boldsymbol{n}\times\boldsymbol{F} = 0 \text{ on } \Gamma_D, \\ \displaystyle -\int_\Omega(\omega^2\epsilon - i\omega\sigma)\boldsymbol{E}\cdot\boldsymbol{\nabla}\bar{q}\,d\boldsymbol{x} + i\omega\int_{\Gamma_C}\gamma\boldsymbol{E}_t\boldsymbol{\nabla}\bar{q}\,dS = \\ \qquad\qquad -i\omega\displaystyle\int_\Omega \boldsymbol{J}^{imp}\cdot\boldsymbol{\nabla}\bar{q}\,d\boldsymbol{x} + i\omega\int_{\Gamma_N}\boldsymbol{J}_S^{imp}\cdot\boldsymbol{\nabla}\bar{q}dS \\ \qquad\qquad \forall q\in H^1(\Omega),\ q = 0 \text{ on } \Gamma_D\,. \end{array}\right. \tag{17.2.10}$$

By repeating the reasoning with the substitution $\boldsymbol{F} = \boldsymbol{\nabla}q$ in the first equation, we learn that the Lagrange multiplier p satisfies the weak form of a Laplace-like equation with homogeneous boundary conditions and, therefore, it *identically vanishes*. For that reason, it is frequently called the *hidden variable*. The stabilized formulation has improved stability properties for small ω. In the case of $\sigma = 0$ and right hand side of (17.2.8) vanishing, we can rescale the Lagrange multiplier, $p = \omega^2 p, q = \omega^2 q$, to obtain a symmetric mixed variational formulation with stability constant converging to one as $\omega \to 0$, comp. [60]. In the general case we cannot avoid a degeneration as $\omega \to 0$ but we can still rescale the Lagrange multiplier with ω ($p = \omega p, q = \omega q$), to improve the stability of the formulation for small ω. The stabilized formulation is possible because gradients of the scalar-valued potentials from $H^1(\Omega)$ form precisely the null space of the curl-curl operator.

From the algebraic point of view, the stabilized variational formulation falls into the category of *mixed problems*. Mixed formulations originate from problems with constraints and have the following abstract algebraic structure.

$$\left\{\begin{array}{lll} \boldsymbol{E}\in\tilde{\boldsymbol{E}}_D + \boldsymbol{Q},\ p\in W & & \\ b(\boldsymbol{E},\boldsymbol{F}) + d(p,\boldsymbol{F}) & = l(\boldsymbol{F}), & \forall \boldsymbol{F}\in\boldsymbol{Q} \\ d^*(\boldsymbol{E},q) & = m(q), & \forall q\in W\,, \end{array}\right. \tag{17.2.11}$$

where $d^*(\boldsymbol{E},q) = \overline{d(q,\boldsymbol{E})}$. In our case,

$$\begin{aligned} \boldsymbol{Q} &= \{\boldsymbol{F}\in H(\mathrm{curl},\Omega)\ :\ \boldsymbol{n}\times\boldsymbol{F} = \boldsymbol{0} \text{ on } \Gamma_D\} \\ W &= \{q\in H^1(\Omega)\ :\ q = 0 \text{ on } \Gamma_D\}\,, \end{aligned}$$

and, as usual, $\tilde{\boldsymbol{E}}_D$ denotes a lift of the Dirichlet boundary condition. The sesquilinear and antilinear forms are defined by appropriate terms in (17.2.10).

The point about the stabilized (mixed) formulation is that, whether we use it or not in the actual computations (the improved stability is one good reason to do it...), the original variational problem for the Maxwell equations *is equivalent* to the mixed problem. This suggests that we cannot escape from the theory of mixed formulations when analysing the problem.

17.2.1 Non-dimensionalization of Maxwell's equations

Material scales. For free space, $\epsilon = \epsilon_0 \approx \frac{1}{36\pi}10^{-9}$ [C^2/Nm^2 = F/m], $\mu = \mu_0 = 4\pi 10^{-7}$ [N/A^2 = h/m], $\sigma = 0$, and (17.1.3) reduces to :

$$\boldsymbol{\nabla} \times (\boldsymbol{\nabla} \times \boldsymbol{E}) - \left(\frac{\omega}{c}\right)^2 \boldsymbol{E} = -i\omega\mu_0 \boldsymbol{J}^{imp} \,,$$

where $c = (\epsilon_0\mu_0)^{-\frac{1}{2}} = 3 \times 10^8$ [m/s] is the speed of light in free space, and $k_0 = \omega/c$ [1/m] is the free space wave number. We shall represent the actual values of ϵ and μ in terms of relative permittivity, and relative permeability,

$$\epsilon_r = \frac{\epsilon}{\epsilon_0}, \quad \mu_r = \frac{\mu}{\mu_0} \,.$$

Spatial scale. With a [m] denoting a characteristic dimension of the computational domain, we introduce dimensionless coordinates,

$$x'_j = \frac{x_j}{a} \,,$$

and the corresponding curl operator,

$$(\boldsymbol{\nabla}' \boldsymbol{E})_i = \epsilon_{ijk} \frac{\partial E_k}{\partial x'_j} \,,$$

(ϵ_{ijk} denotes the Rizzi symbol).

Solution scale. Finally, identifying a characteristic value of the electric field E_0, we introduce a non-dimensional electric field vector,

$$\boldsymbol{E}' = \frac{\boldsymbol{E}}{E_0} \,.$$

Switching to the non-dimensional variables in (17.2.7) and multiplying both sides by factor $\frac{\mu_0}{E_0^2 a}$, we obtain,

material	μ_r	ϵ_r	$\sqrt{\mu_0/\epsilon_0}a\sigma$
Alumina (10GHz)	1	10	0.63
Polyethylene (10GHz)	1	2.25	0.19
copper	1	1	2.2E+9
seawater	1	1	1.5E+3
human muscle (900MHz)	1	58	4.56E+2

Table 17.1: Material constants for selected materials

$$
\left\{
\begin{array}{l}
\boldsymbol{E}' \in H(\mathrm{curl},\Omega'),\ \boldsymbol{n}\times\boldsymbol{E}' = \boldsymbol{n}\times\boldsymbol{E}'_0 \text{ on } \Gamma'_D \\
\displaystyle\int_{\Omega'} \left\{ \frac{1}{\mu_r}(\boldsymbol{\nabla}'\times\boldsymbol{E}')(\boldsymbol{\nabla}'\times\bar{\boldsymbol{F}}') - (\omega^2 a^2\epsilon_0\mu_0 a^2\epsilon_r - i\omega\sigma\mu_0 a^2)\boldsymbol{E}'\bar{\boldsymbol{F}}' \right\} d\boldsymbol{x}' \\
-i\omega\mu_0 a\int_{\Gamma'_C}\gamma(\boldsymbol{n}\times\boldsymbol{E})\bar{\boldsymbol{F}}\,dS = -i\omega\dfrac{\mu_0 a^2}{E_0}\displaystyle\int_\Omega \boldsymbol{J}^{imp}\bar{\boldsymbol{F}}\,d\boldsymbol{x} + i\omega\frac{\mu_0 a}{E_0}\int_{\Gamma'_N\cup\Gamma'_C}\boldsymbol{J}_S^{imp}\bar{\boldsymbol{F}}\,dS \\
\qquad \text{for every } \boldsymbol{F}' \in H(\mathrm{curl},\Omega'),\ \boldsymbol{n}\times\boldsymbol{F}' = 0 \text{ on } \Gamma'_D\,.
\end{array}
\right. \tag{17.2.12}
$$

Here primes denote the non-dimensional quantities including the domain and parts of its boundaries.

This leads to a consistent non-dimensionalization of frequency, conductivity, impedance constant, and impressed volume and surface currents,

$$
\begin{aligned}
\omega' &= \omega\sqrt{\epsilon_0\mu_0}\,a \\
\sigma' &= \sigma\sqrt{\frac{\mu_0}{\epsilon_0}}\,a \\
\gamma' &= \sqrt{\frac{\mu_0}{\epsilon_0}}\gamma \\
\boldsymbol{J}^{imp,'} &= \frac{\boldsymbol{J}^{imp}\sqrt{\frac{\mu_0}{\epsilon_0}}\,a}{E_0} \\
\boldsymbol{J}^{imp,'} &= \frac{\boldsymbol{J}^{imp}\sqrt{\frac{\mu_0}{\epsilon_0}}}{E_0}\,.
\end{aligned}
$$

Notice that the non-dimensional frequency coincides with the non-dimensional free-space wave number.

With the non-dimensional quantities, the variational formulation recovers its original form. We drop all the primes in the notation with the understanding that the actual computations take place always in terms of the non-dimensional quantities.

A few sample values of relative permeability μ_r, relative permittivity ϵ_r, and scaled conductivity $\sqrt{\frac{\mu_0}{\epsilon_0}}\,a\sigma$ with a = 1 [m] are summarized in Table 17.1.

In reality, the material constants depend upon frequency ω, temperature, and other factors. The discussed equations apply only to isotropic materials, many materials, e.g. ferromagnetics, are strongly anisotropic.

Exercises

Exercise 17.1 Consider a system of first order differential equations,

$$\dot{\boldsymbol{U}} + \boldsymbol{A}_i \boldsymbol{U}_{,i} = \boldsymbol{0} \,,$$

where $\boldsymbol{U}$ is a *group variable* representing several scalar quantities, dot denotes differentiation in time, and comas stand for derivatives in space coordinates. *Jacobian matrices* $\boldsymbol{A}_i$ are, in general, functions of the unknown $\boldsymbol{U}$. If not, the system is linear. Summation convention is used. Let $\boldsymbol{n} = (n_i)$ denote a unit vector, i.e. $n_i \, n_i = 1$. The system is called *hyperbolic* if the *normal Jacobian matrix*,

$$\boldsymbol{A}_n = \boldsymbol{A}_i n_i \,,$$

has only real eigenvalues and a corresponding complete system of eigenvectors, i.e. for each of multiple eigenvalues, the corresponding eigenspace has the dimension equal to the algebraic multiplicity of the eigenvalue. Prove that the Maxwell system consisting of the Ampere and Faraday equations, and $\boldsymbol{U}$ composed of components of electric and magnetic fields, in the lossless case $\sigma = 0$, is hyperbolic. Determine the eigenpairs of the corresponding normal Jacobian matrix. Follow the following steps.

Symmetric case. Assume $\epsilon = \mu = 1$. Demonstrate that the normal Jacobian matrix has the form,

$$\boldsymbol{A}_n = \begin{pmatrix} & \boldsymbol{B} \\ -\boldsymbol{B} & \end{pmatrix} \,,$$

where $\boldsymbol{B}$ is 3×3 antisymmetric matrix corresponding to the curl operator, $\boldsymbol{B}^T = -\boldsymbol{B}$. Conclude that $\boldsymbol{A}_n$ is, therefore, a symmetric matrix and draw conclusions about the corresponding eigenpairs.

Calculation of eigenpairs. Notice that,

$$\boldsymbol{A}_n^2 = \begin{pmatrix} \boldsymbol{B}\boldsymbol{B}^T & \\ & \boldsymbol{B}^T\boldsymbol{B} \end{pmatrix} \,,$$

and utilize the decoupling for the eigenvalues computation.

General case. Symmetrize the system, i.e. determine such a matrix $\boldsymbol{A}_0$ (a symmetrizer) that the original non-symmetric system turns into a new system of the form

$$\boldsymbol{A}_0 \dot{\boldsymbol{U}} + \boldsymbol{A}_0 \boldsymbol{A}_i \boldsymbol{U}_{,i} = \boldsymbol{0} \,,$$

where the products $\boldsymbol{A}_0\boldsymbol{A}_i$ are symmetric. Symmetrization for Maxwell equations results from a simple scaling, i.e. matrix $\boldsymbol{A}_0$ is diagonal. Use results of the previous two steps.

Exercise 17.2 Consider the following simple elliptic model problem,

$$\begin{aligned} -\sigma_1 u_1'' &= 0 \qquad x \in (-1,0) \\ -\sigma_2 u_2'' &= 1 \qquad x \in (0,1)\,, \end{aligned}$$

accompanied by boundary and interface conditions,

$$u_1(-1) = 0, \quad u_2(1) = 0, \quad u_1(0) = u_2(0), \quad \sigma_1 u_1'(0) = \sigma_2 u_2'(0)\,.$$

Determine the solution. Consider then limit $\sigma_1 \to \infty$ and show that both $u_1, \sigma_1 u_1' \to 0$ in $(-1,0)$ but $\sigma_1 u_1'(0)$ converges to a finite value. Consequently, show that u_2 converges to the solution of the Dirichlet problem,

$$-\sigma_2 u_2'' = 1 \text{ in } (0,1), \quad u(0) = u(1) = 0\,,$$

with the non-zero flux $\sigma_2 u_2''(0)$. The situation is similar in the case of a PEC limit. The domain occupied by the perfect conductor is eliminated from the computational domain, with the PEC boundary condition imposed on the resulting boundary, and a resulting non-zero impressed surface current.

Exercise 17.3 Show that the mixed problem,

$$\left\{ \begin{array}{ll} E \in Q, p \in W & \\ b(E,F) + d(p,F) = l(F), & \forall F \in Q \\ d^*(E,q) \qquad\quad = m(q), & \forall q \in W\,, \end{array} \right.$$

where $b(E,F), d(p,F), d^*(E,q) = \overline{d(q,E)}$ are appropriately defined bilinear (sesquilinear) forms, can be interpreted as a standard variational problem for the group variable $U = (E,p) \in Q \times W$,

$$\left\{ \begin{array}{l} U \in Q \times W \\ B(U,V) = L(V), \quad \forall\, V \in Q \times W\,, \end{array} \right.$$

where $B(U,V)$ is a "big" bilinear (sesquilinear) form defined as,

$$B(U,V) = b(E,F) + d(p,F) + d^*(E,q)\,.$$

Is such a big form unique, i.e. can you define another $B(U,V)$ that yields a problem equivalent to the original mixed problem ?

Demonstrate that the Babuška's inf-sup stability condition for the big bilinear form,

$$\inf_{U\neq 0} \sup_{V \neq 0} \frac{|B(U,V)|}{\|U\|\,\|V\|} \geq \gamma > 0\,,$$

implies the Brezzi inf-sup condition,

$$\inf_{p\neq 0}\sup_{F\neq 0}\frac{|b(p,F)|}{\|p\|\,\|F\|}\geq\beta>0\,,$$

for some $\beta > 0$. You may assume the "composite norm" in the form, $\|U\|_{Q\times W} = \|E\|_Q + \|p\|_W$.

For a discussion on equivalence of Babuška and Brezzi conditions, consult [42].

Chapter 18

Edge Elements and de Rham Diagram

Positive definite problems enjoy an unconditional discrete stability - Galerkin method delivers the best approximation in terms of the energy norm. Helmholtz-like equations representing wave propagation and vibration problems are only asymptotically stable. Nevertheless, for meshes that are fine enough, Galerkin method is again (asymptotically) optimal in the energy norm. The situation for mixed problems is quite different. Discrete stability is not longer implied by the continuous stability. Necessary for the discrete stability is the so-called Babuška-Brezzi or $\inf - \sup$ condition relating spaces $\boldsymbol{Q}$ and W, and the constraint sesquilinear form $d(q, \boldsymbol{F})$, compare Exercise 17.3,

$$\sup_{\boldsymbol{F} \in \boldsymbol{Q}} \frac{|d(q, \boldsymbol{F})|}{\|\boldsymbol{F}\|_{\boldsymbol{Q}}} \geq \beta \|q\|_W, \quad \forall q \in W \, ,$$

where constant β is mesh independent, i.e. it depends neither on element size h nor element order p. The condition is trivially satisfied at the continuous level, because the space W is a subspace of space $\boldsymbol{Q}$ (the supremum is attained by $\boldsymbol{F} = \boldsymbol{\nabla} q$); gradients of potentials from H^1 are automatically elements of $H(\mathrm{curl})$. Indeed, the components of the gradient are square integrable by assumption, and the curl of a gradient simply vanishes, so the condition on the square integrability of the curl of a gradient, is trivially satisfied.

The inclusion property, $\boldsymbol{\nabla} W \subset \boldsymbol{Q}$, is part of a bigger picture expressed in terms of an *exact sequence* of spaces that we will study in this chapter. Construction of stable elements for Maxwell's equations, the so-called *edge elements* involves reproducing the exact sequence on the discrete level, for polynomial spaces. Finally, the two exact sequences will be coupled together in a commuting diagram using the projection-based interpolation operator discussed in the previous chapters and its generalizations for space $H(\mathrm{curl})$ and other spaces entering the exact sequence.

18.1 Exact sequences

For a *simply-connected* domain $\Omega \subset \mathbb{R}^3$, the gradient and curl operators, along with the divergence operator, form an *exact sequence*,

$$\mathbb{R} \longrightarrow H^1 \xrightarrow{\boldsymbol{\nabla}} \boldsymbol{H}(\mathrm{curl}) \xrightarrow{\boldsymbol{\nabla}\times} \boldsymbol{H}(\mathrm{div}) \xrightarrow{\boldsymbol{\nabla}\circ} L^2 \longrightarrow 0\,.$$

In an exact sequence of operators, the range of each operator coincides with the null space of the operator next in the sequence. Simply speaking, the gradient of a function vanishes if and only if the function is constant, the curl of a vector- valued function is zero, if and only if the function is the gradient of a scalar potential, etc.

The spaces above may incorporate homogeneous essential boundary conditions. Introducing,

$$\begin{aligned} W &= \{q \in H^1(\Omega) \,:\, q = 0 \text{ on } \Gamma_D\} \\ \boldsymbol{Q} &= \{\boldsymbol{E} \in H(\mathrm{curl},\Omega), \boldsymbol{E}_t = \boldsymbol{0} \text{ on } \Gamma_D\} \\ \boldsymbol{V} &= \{\boldsymbol{H} \in H(\mathrm{div},\Omega), H_n = 0 \text{ on } \Gamma_D\} \\ Y &= L^2(\Omega)\,, \end{aligned}$$

we have the exact sequence:

$$W \xrightarrow{\boldsymbol{\nabla}} \boldsymbol{Q} \xrightarrow{\boldsymbol{\nabla}\times} \boldsymbol{V} \xrightarrow{\boldsymbol{\nabla}\circ} Y \longrightarrow 0\,.$$

The presence of $\mathbb{R}$ in the original sequence signifies the fact that the null space of the gradient consists of constant fields. With the Dirichlet boundary condition on boundary Γ_D, the constant must be zero, and the space of constant fields is eliminated. Similarly, presence of the trivial space in the end of the sequence signifies the fact that the preceeding operator is a surjection. If Γ_D coincides with the *whole boundary*, the L^2 space must be replaced with the space of fields of zero average,

$$L^2_0 = \{u \in L^2(\Omega) \,:\, \int_\Omega u = 0\}\,.$$

In order to simplify the notation, we shall drop $\mathbb{R}$ and the trivial space from our discussion. We shall also remember that symbol L^2 is interpreted as L^2_0 when the Dirichlet boundary conditions are imposed on the whole boundary.

In two space dimensions, the 3D exact sequence gives rise to two sequences,

$$\mathbb{R} \longrightarrow H^1 \xrightarrow{\boldsymbol{\nabla}} \boldsymbol{H}(\mathrm{curl}) \xrightarrow{curl} L^2\,,$$

and,

$$\mathbb{R} \longrightarrow H^1 \xrightarrow{\boldsymbol{\nabla}\times} \boldsymbol{H}(\mathrm{div}) \xrightarrow{\boldsymbol{\nabla}\circ} L^2\,.$$

Notice the difference between the two curl operators discussed in the previous chapter. The second exact sequence can be obtained by "rotating" the first sequence by 90 degrees.

The exact sequence property is crucial in proving the stability result for the regularized variational formulation for the time- harmonic Maxwell equations, see [60]. This suggests to construct (piecewice) polynomial Finite Element discretization of the $H(\mathrm{curl})$ space in such a way that the exact sequence property is also satisfied at the discrete level. Two fundamental families of elements introduced by Nédélec in [92, 93] satisfy such a condition.

In this volume we shall discuss only the 2D versions of those elements.

18.1.1 Nédélec's Triangular Element of the Second Type

All polynomial spaces are defined on the master triangle,

$$T = \{(x_1, x_2) \,:\, x_1 > 0, x_2 > 0, x_1 + x_2 < 1\} \,.$$

We have the following exact sequence,

$$\mathcal{P}^p \quad \xrightarrow{\boldsymbol{\nabla}} \quad \boldsymbol{P}^{p-1} \quad \xrightarrow{\boldsymbol{\nabla}\times} \quad \mathcal{P}^{p-2} \,.$$

Here $\mathcal{P}^p$ denotes the space of polynomials of (group) order less or equal p, e.g. $x_1^2 x_2^3 \in \mathcal{P}^5$, and $\boldsymbol{P}^p = \mathcal{P}^p \times \mathcal{P}^p$.

Obviously, the construction starts with $p \geq 2$, i.e. the $\boldsymbol{H}(\mathrm{curl})$-conforming elements are at least of first order.

The construction can be generalized to triangles of *variable order*. With each triangle's edge we associate the corresponding edge order p_e. We assume that,

$$p_e \leq p \,\forall \text{ edge } e \,.$$

The assumption is satisfied in practice by enforcing the *minimum rule* discussed in previous chapters, i.e. setting the edge orders to the minimum of the orders of the adjacent elements. We introduce now the following polynomial spaces.

- The space of scalar-valued polynomials of order less or equal p, whose traces on edges e reduce to polynomials of (possibly smaller) order p_e,

$$\mathcal{P}^p_{p_e} = \{u \in \mathcal{P}^p \,:\, u|_e \in \mathcal{P}^{p_e}(e)\}.$$

- The space of vector-valued polynomials of order less or equal p, whose *tangential traces* on edges e reduce to polynomials of order p_e,

$$\boldsymbol{P}^p_{p_e} = \{\boldsymbol{E} \in \boldsymbol{P}^p \,:\, E_t|_e \in \mathcal{P}^{p_e}(e)\}.$$

- The space of scalar-valued polynomials of order less or equal p, with zero average

$$\mathcal{P}_0^p = \{u \in \mathcal{P}^p \ : \ \int_T u \ = 0\}.$$

We have then the exact sequence,

$$\mathcal{P}^p_{p_e} \quad \stackrel{\boldsymbol{\nabla}}{\longrightarrow} \quad \boldsymbol{P}^{p-1}_{p_e-1} \quad \stackrel{\boldsymbol{\nabla}\times}{\longrightarrow} \quad \mathcal{P}^{p-2}(\mathcal{P}_0^{p-2}) \ . \tag{18.1.1}$$

The case $p_e = -1$ corresponds to the homogeneous Dirichlet boundary condition. In the case of the homogeneous Dirichlet boundary conditions imposed on *all* edges e, the last space in the sequence, polynomials of order $p-2$, must be replaced with the polynomials with zero average.

18.1.2 Nédélec's Rectangular Element of the First Type

All spaces are defined on a unit square. We introduce the following polynomial spaces.

$$\begin{aligned} W_p &= Q^{(p,q)} \\ \boldsymbol{Q}_p &= Q^{(p-1,q)} \times Q^{(p,q-1)} \\ Y_p &= Q^{(p-1,q-1)} \ . \end{aligned}$$

Here $Q^{p,q} = \mathcal{P}^p \otimes \mathcal{P}^q$ denotes the space of polynomials of order less or equal p, q with respect to x, y, respectively. For instance, $2x^2y^3 \in Q^{(2,3)}$. The polynomial spaces form again the exact sequence,

$$W_p \quad \stackrel{\boldsymbol{\nabla}}{\longrightarrow} \quad \boldsymbol{Q}_p \quad \stackrel{\boldsymbol{\nabla}\times}{\longrightarrow} \quad Y_p \ . \tag{18.1.2}$$

The generalization to variable order elements is a little less straightforward than for the triangles. For each horizontal edge e, we introduce order p_e, and with each vertical edge e, we associate order q_e. We assume again that the minimum rule holds, i.e.

$$p_e \leq p, \quad q_e \leq q \ .$$

By $Q^{(p,q)}_{p_e,q_e}$ we understand the space of polynomials of order less or equal p with respect to x, and order less or equal q with respect to y, such that their traces to horizontal edges e reduce to polynomials of (possibly smaller) degree p_e, and restrictions to vertical edges reduce to polynomials of (possibly smaller) order q_e,

$$\begin{aligned} Q^{(p,q)}_{p_e,q_e} = \{u \in Q^{(p,q)} \ : \quad & u(\cdot,0) \in \mathcal{P}^{p_1}(0,1), u(\cdot,1) \in \mathcal{P}^{p_2}(0,1), \\ & u(0,\cdot) \in \mathcal{P}^{q_1}(0,1), u(1,\cdot) \in \mathcal{P}^{q_2}(0,1)\} \ . \end{aligned}$$

With spaces,

$$\begin{aligned} W_p &= Q^{(p,q)}_{p_e,q_e} \\ \boldsymbol{Q}_p &= Q^{(p-1,q)}_{p_e-1} \times Q^{(p,q-1)}_{q_e-1} \\ Y_p &= Q^{(p-1,q-1)} \,, \end{aligned}$$

we have the exact sequence,

$$W_p \quad \stackrel{\boldsymbol{\nabla}}{\longrightarrow} \quad \boldsymbol{Q}_p \quad \stackrel{\boldsymbol{\nabla}\times}{\longrightarrow} \quad Y_p \,.$$

Notice that space $\boldsymbol{Q}_p$ cannot be obtained by merely differentiating polynomials from $Q^{(p,q)}_{p_e,q_e}$. For the derivative in x, this would lead to space $Q^{(p-1,q)}_{p_e-1,q_e}$ for the first component, whereas in our definition above q_e has been increased to q. This is motivated with the fact that the traces of E_1 along the *vertical edges* are interpreted as *normal* components of the $\boldsymbol{E}$ field. The $\boldsymbol{H}(\mathrm{curl})$-conforming fields 'connect' only through tangential components, shape functions corresponding to the normal components on the boundary are classified as interior modes, and they should depend only upon the order of the element, and not the order of neighboring elements.

18.1.3 Nédélec's Triangular Elements of the First Type

There is a significant difference between the triangular and square elements presented so far. For the triangle, the order p drops in the diagram from p to $p-2$. This merely reflects the fact that the differentiation lowers always the polynomial order by one. In the case of the Q-spaces, however, the order in the diagram has dropped only by one, from (p,q) to $(p-1,q-1)$. A similar effect can be obtained for triangles.

The goal is to switch from $p-2$ to $p-1$ in (18.1.1) *without* increasing the order p in the space on the left. We begin by rewriting (18.1.1) with p increased by one.

$$\mathcal{P}^{p+1}_{p_e} \quad \stackrel{\boldsymbol{\nabla}}{\longrightarrow} \quad \mathbf{P}^{p}_{p_e-1} \quad \stackrel{\boldsymbol{\nabla}\times}{\longrightarrow} \quad \mathcal{P}^{p-1} \,.$$

Notice that we have not increased the order along the edges. This is motivated with the fact that the edge orders do not affect the very last space in the diagram [1]. Next, we decompose the space of potentials into the previous space of polynomials $\mathcal{P}^p_{p_e}$, and *an algebraic complement* $\tilde{\mathcal{P}}^{p+1}_{p_e}$,

$$\mathcal{P}^{p+1}_{p_e} = \mathcal{P}^{p}_{p_e} \oplus \tilde{\mathcal{P}}^{p+1}_{p_e} \,.$$

The algebraic complement is *not unique*, it may be constructed in (infinitely) many different ways. The decomposition in the space of potentials implies a corresponding decomposition in the $\boldsymbol{H}(\mathrm{curl})$-conforming space,

$$\boldsymbol{P}^{p}_{p_e-1} = \boldsymbol{P}^{p-1}_{p_e-1} \oplus \boldsymbol{\nabla}(\tilde{\mathcal{P}}^{p+1}_{p_e}) \oplus \tilde{\boldsymbol{P}}^{p}_{p_e-1} \,.$$

[1] Except for the case of the homogeneous Dirichlet boundary condition imposed on the whole boundary which forces the use of polynomials of zero average for the last space in the diagram

The algebraic complement $\tilde{\boldsymbol{P}}^p_{p_e-1}$ is again *not unique*. The desired extension of the original sequence can now be constructed by remowing the gradients of order $p+1$,

$$\mathcal{P}^p_{p_e} \quad \xrightarrow{\boldsymbol{\nabla}} \quad \mathbf{P}^{p-1}_{p_e-1} \oplus \tilde{\boldsymbol{P}}^p_{p_e-1} \quad \xrightarrow{\boldsymbol{\nabla}\times} \quad \mathcal{P}^{p-1} \,.$$

Note the following facts.

- The construction enables the $\boldsymbol{H}(\mathrm{curl})$-conforming discretization of lowest order on triangles. For $p = p_e = 1$,
$$\mathbf{P}^0_0 \oplus \tilde{\boldsymbol{P}}^1 = \boldsymbol{P}^1_0, \quad \dim \boldsymbol{P}^1_0 = 3 \,.$$
The complement $\tilde{\mathcal{P}}^2_1$ *is empty* and, therefore, in this case, the resulting *Whitney space* $\boldsymbol{P}^1_0 = \mathbf{P}^0_0 \oplus \tilde{\boldsymbol{P}}^1_0$ *is unique*. This is the smallest space to enforce the continuity of the (constant) tangential component of $\boldsymbol{E}$ across the interelement boundaries.

- It is not necessary but natural to construct the complements using scalar and vector *bubble functions*. In this case the notation $\tilde{\mathcal{P}}^{p+1}_{-1}$ and $\tilde{\boldsymbol{P}}^p_{-1}$ is more appropriate. The concept is especially natural if one uses *hierarchical shape functions*. We can always enforce the zero trace condition by augmenting original shape functions with functions of lower order. In other words, we change the complement but *do not alter* the ultimate polynomial space.

The choice of the complements may be made unique by imposing additional conditions. The original Nédélec's construction, for elements of uniform order p, employs symmetric [2] polynomials,

$$\boldsymbol{R}^p = \{\boldsymbol{E} \in \boldsymbol{P}^p \,:\, \boldsymbol{\epsilon}^p(\boldsymbol{E}) = \mathbf{0}\} \,,$$

where $\boldsymbol{\epsilon}^p$ is the Nédélec *symmetrization operator*,

$$\begin{aligned}&(\boldsymbol{\epsilon}^p(\boldsymbol{E}))_{i_1,\ldots,i_{p+1}} \\ &= \frac{1}{p+1}\left(\frac{\partial^p E_{i_1}}{\partial x_{i_2}\ldots\partial x_{i_p}\partial x_{i_{p+1}}} + \frac{\partial^p E_{i_2}}{\partial x_{i_3}\ldots\partial x_{i_{p+1}}\partial x_{i_1}} + \ldots + \frac{\partial^p E_{i_{p+1}}}{\partial x_{i_1}\ldots\partial x_{i_{p-1}}\partial x_{i_p}}\right) \,.\end{aligned}$$

The algebraic complement can then be selected as the subspace of *homogeneous*[3] symmetric polynomials $\boldsymbol{D}^p$,

$$\boldsymbol{R}^p = \boldsymbol{P}^p \oplus \boldsymbol{D}^p \,.$$

There are many equivalent conditions characterizing the space $\boldsymbol{D}^p$. The most popular one reads as follows.

$$\boldsymbol{E} \in \boldsymbol{D}^p \Leftrightarrow \boldsymbol{E} \text{ is homogeneoous and } \boldsymbol{x}\cdot\boldsymbol{E}(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x} \,.$$

[2] They should have been called rather *antisymmetric* polynomials

[3] A polynomial of order p is homogeneous if it can be represented as a sum of monomials of order p. Equivalently, $u(\xi x_1,\ldots,\xi x_n) = \xi^p u(x_1,\ldots,x_n)$.

The space $\boldsymbol{D}^p$ can also nicely be characterized as the image of homogeneous polynomials of order $p-1$ under the Poincare map, see [77, 78],

$$\begin{aligned} E_1(\boldsymbol{x}) &= -x_2 \int_0^1 t\psi(t\boldsymbol{x})\, dt \\ E_2(\boldsymbol{x}) &= x_1 \int_0^1 t\psi(t\boldsymbol{x})\, dt\,. \end{aligned} \tag{18.1.3}$$

The Poincare map is a right inverse of the curl map, $\boldsymbol{\nabla} \times \boldsymbol{E} = \psi$, for the $\boldsymbol{E}$ defined above. Consistently with our discussion, it can be shown that the tangential trace of a symmetric polynomial of order p is always a polynomial of order less or equal $p-1$. For other characterizations of space $\boldsymbol{D}^p$, see [73].

An important property of the Nédélec space $\boldsymbol{R}^p$ is that it is invariant under affine transformations, comp. Exercise 18.1. Consequently, the polynomial space is independent of the way in which the vertices of the triangle are denumerated.

Uniqueness of the spaces could also be naturally enforced by requesting *orthogonality* of algebraic complements [121, 39],

$$\mathcal{P}^{p+1}_{p_e} = \mathcal{P}^p_{p_e} \oplus \tilde{\mathcal{P}}^{p+1}_{-1}, \quad \mathcal{P}^{p+1}_{-1} = \mathcal{P}^p_{-1} \overset{\perp}{\oplus} \tilde{\mathcal{P}}^{p+1}_{-1}$$

$$\boldsymbol{P}^p_{p_e-1} = \boldsymbol{P}^{p-1}_{p_e-1} \oplus \boldsymbol{\nabla}(\tilde{\mathcal{P}}^{p+1}_{p_e}) \oplus \tilde{\boldsymbol{P}}^p_{-1}, \quad \boldsymbol{P}^p_{-1} = \boldsymbol{P}^{p-1}_{-1} \overset{\perp}{\oplus} \boldsymbol{\nabla}(\tilde{\mathcal{P}}^{p+1}_{-1}) \overset{\perp}{\oplus} \tilde{\boldsymbol{P}}^p_{-1}\,.$$

The orthogonality in the $\boldsymbol{E}$ space is understood in the $\boldsymbol{H}(\mathrm{curl})$ sense, with the corresponding L^2 orthogonality for the gradients.

18.1.4 Parametric elements

Given a bijective map $\boldsymbol{x} = \boldsymbol{x}_K(\boldsymbol{\xi})$ transforming master element $\hat{K}$ onto a physical element K, and master element shape functions $\hat{\phi}(\boldsymbol{\xi})$, we define the H^1-conforming shape functions on the physical element in terms of master element coordinates,

$$\phi(\boldsymbol{x}) = \hat{\phi}(\boldsymbol{\xi}) = \hat{\phi}(\boldsymbol{x}_K^{-1}(\boldsymbol{x})) = (\hat{\phi} \circ \boldsymbol{x}_K^{-1})(\boldsymbol{x})\,.$$

The definition reflects the fact that the integration of master element matrices is always done in terms of master element coordinates and, therefore, it is simply convenient to define the shape functions in terms of master coordinates $\boldsymbol{\xi}$. This implies that the parametric element shape functions are compositions of the inverse $\boldsymbol{x}_K^{-1}$ and the master element polynomial shape functions. In general, we do not deal with polynomials anymore. In order to keep the exact sequence property, we have to define the $\boldsymbol{H}(\mathrm{curl})$-, $\boldsymbol{H}(\mathrm{div})$-, and L^2-conforming elements consistently with the way the differential operators transform. For gradients we have,

$$\frac{\partial u}{\partial x_i} = \frac{\partial \hat{u}}{\partial \xi_k}\frac{\partial \xi_k}{\partial x_i}$$

and, therefore,

$$E_i = \hat{E}_k \frac{\partial \xi_k}{\partial x_i} .$$

For the curl operator we have,

$$\begin{aligned}
\mathrm{curl}\boldsymbol{E} &= \frac{\partial E_2}{\partial x_1} - \frac{\partial E_1}{\partial x_2} = \frac{\partial}{\partial x_1}\left(\hat{E}_k \frac{\partial \xi_k}{\partial x_2}\right) - \frac{\partial}{\partial x_2}\left(\hat{E}_l \frac{\partial \xi_l}{\partial x_1}\right) \\
&= \frac{\partial \hat{E}_k}{\partial x_1}\frac{\partial \xi_k}{\partial x_2} + \hat{E}_k \frac{\partial^2 \xi_k}{\partial x_1 \partial x_2} - \frac{\partial \hat{E}_l}{\partial x_2}\frac{\partial \xi_l}{\partial x_1} - \hat{E}_l \frac{\partial^2 \xi_l}{\partial x_1 \partial x_2} \\
&= \frac{\partial \hat{E}_k}{\partial x i_l}\frac{\partial \xi_l}{\partial x_1}\frac{\partial \xi_k}{\partial x_2} - \frac{\partial \hat{E}_l}{\partial \xi_k}\frac{\partial \xi_k}{\partial x_2}\frac{\partial \xi_l}{\partial x_1} \\
&= \left(\frac{\partial \hat{E}_k}{\partial \xi_l} - \frac{\partial \hat{E}_l}{\partial \xi_k}\right)\frac{\partial \xi_l}{\partial x_1}\frac{\partial \xi_k}{\partial x_2} \\
&= (\mathrm{curl}\hat{\boldsymbol{E}})J^{-1} .
\end{aligned}$$

Here $\mathrm{curl}\hat{\boldsymbol{E}}$ denotes the curl in master coordinates, and J^{-1} is the inverse jacobian of the map from the master to the physical coordinates.

This establishes the transformation rule for the L^2-conforming elements,

$$f = J^{-1}\hat{f} .$$

Defining the parametric element spaces $W_p, \boldsymbol{Q}_p, Y_p$ using the transformation rules listed above, we preserve for the parametric element the exact sequence (18.1.2).

REMARK 1 In the case of the *isoparametric element*, the components of the transformation map $\boldsymbol{x}_K$ come from the space of the H^1-conforming master element,

$$x_j = \sum_k x_{j,k}\hat{\phi}_k(\boldsymbol{\xi}) = \sum_k \boldsymbol{x}_k \phi_k(\boldsymbol{x}) .$$

Here $x_{j,k}$ denote the (vector-valued) geometry degrees-of-freedom corresponding to element shape functions $\phi_k(\boldsymbol{x})$. By construction, therefore, the parametric element shape functions can reproduce any linear function $a_j x_j$. As they also can reproduce constants, the isoparametric element space of shape functions contains the space of all linear polynomials in $\boldsymbol{x}$ - $a_j x_j + b$, in mechanical terms - the space of linearized rigid body motions. The exact sequence property implies that the $\boldsymbol{H}(\mathrm{curl})$-conforming element can reproduce only constant fields, but the L^2-conforming element, in general, cannot reproduce even constants. This indicates in particular that, in context of general parametric (non affine) elements [4] unstructured mesh generators should be used with caution, comp. [12]. The critique does not apply to (algebraic) mesh generators based on a consistent representation of the domain as a manifold, with underlying global maps parametrizing portions of the domain. Upon a change of variables, the original problem can then be redefined in the reference domain discretized with affine elements. ∎

[4] Note that general quadrilaterals with straight edges are not affi ne elements

18.2 Projection-based interpolation

The goal of this section is to extend the notion of the projection-based interpolation to the $H(\mathrm{curl})$-conforming elements. We recall first the main ideas of the interpolation for H^1-conforming elements discussed in the previous chapters. The construction was based on three main assumtions.

Locality. The interpolant of a function over an element is defined entirely in terms of the restriction of the function to the element only.

Global continuity. The union of element interpolants is globally conforming (continuous for the H^1 case).

Optimality. The interpolation error behaves asymptotically, *both in h and p,* in the same way as the actual approximation error.

Locality and global continuity imply that the interpolant $u^p = \Pi u$ must match interpolated function u at vertices:

$$u^p(v) = u(v) \quad \text{for each vertex } v\,.$$

With the vertex values fixed, locality and global continuity imply that the restriction of the interpolant to an edge, should be calculated using the restriction of function u to that edge only. Optimality, in turn, implies that we should use a projection in the appropriate "edge norm".

$$\|u - u^p\|_e \to \min, \quad \text{for each edge } e\,.$$

Finally, we project over the element in the H^1-seminorm to complete the definition. The choice of element norm is dictated by the problem being solved: the H^1-seminorm for elliptic problems. It follows from the optimality condition then that the edge norm is implied by the Trace Theorem. The H^1-conforming interpolant $\Pi u \in X(K)$ of function $u \in H^r(T), r > 0$, is formally defined as follows.

$$\begin{cases} u^p(v) = u(v) & \text{for each vertex } v \\ |u^p - u|_{\frac{1}{2},e} \to \min & \text{for each edge } e \\ |u^p - u|_{1,T} \to \min & \text{for element } K\,. \end{cases} \tag{18.2.4}$$

Finally recall that, in practical implementation, the $H^{\frac{1}{2}}(e)$ norm has been replaced with a weighted H^1- seminorm.

We follow now the same guidelines when defining the projection-based interpolation for the $H(\mathrm{curl})$-conforming elements. $\boldsymbol{H}(\mathrm{curl})$-conformity involves only the continuity of the tangential component. Consequently, there is no interpolation at vertices, and the interpolation process starts from edges. Given a function $\boldsymbol{E}(\boldsymbol{x})$, we define the projection based interpolant $\Pi^{curl}\boldsymbol{E} := \boldsymbol{E}^p$ by requesting the following conditions.

- Locality and global continuity imply that values of the interpolant along an edge e must be determined using values of the tangential component E_t only. Optimality condition implies that we should use a projection,

$$\|E_t^p - E_t\|_e \rightarrow \min \ . \tag{18.2.5}$$

- Once we have determined the interpolant over the element boundary, locality and optimality imply again that we should use a projection,

$$\|\boldsymbol{E}^p - \boldsymbol{E}\|_K \rightarrow \min \ .$$

The choice of the element norm follows from the energy norm for the Maxwell equations - we should use the $H(\mathrm{curl})$-norm. This is equivalent to solving the following local problem,

$$\begin{cases} E_t^p \text{ fixed (determined in the first step)} \\ \displaystyle\int_K \mathrm{curl}(\boldsymbol{E}^{\mathrm{p}} - \boldsymbol{E})\mathrm{curl}\boldsymbol{\phi} + (\boldsymbol{E}^p - \boldsymbol{E})\boldsymbol{\phi}\, d\boldsymbol{x} = 0, \quad \forall \text{ element vector bubble function } \boldsymbol{\phi} \ . \end{cases}$$

Notice that the vector-valued element bubble functions are understood here differently from those for elliptic problems. The tangential component of the vector-valued bubble must vanish along the element boundary but the normal one need not.

The exact sequence property implies that the vector bubbles contain gradients of scalar bubbles. This implies that the following compatibility equation is automatically satisfied.

$$\int_K (\boldsymbol{E}^p - \boldsymbol{E})\boldsymbol{\nabla}\phi\, d\boldsymbol{x} = 0, \quad \forall \text{ element scalar bubble function } \phi \ .$$

If we reimpose the compatibility condition in the original system, and introduce the corresponding Lagrange multiplier (equal zero), we get,

$$\begin{cases} \displaystyle\int_K \mathrm{curl}(\boldsymbol{E}^{\mathrm{p}} - \boldsymbol{E})\mathrm{curl}\boldsymbol{\phi} + (\boldsymbol{E}^p - \boldsymbol{E})\boldsymbol{\phi}\, d\boldsymbol{x} \ + \int_K \boldsymbol{\nabla} p\, \boldsymbol{\phi}\, d\boldsymbol{x} \ = 0, \quad \forall \text{ vector bubbles } \boldsymbol{\phi} \\ \displaystyle\int_K (\boldsymbol{E}^p - \boldsymbol{E})\boldsymbol{\nabla}\phi\, d\boldsymbol{x} \qquad\qquad\qquad\qquad\qquad\qquad = 0, \quad \forall \text{ scalar bubbles } \phi \ . \end{cases}$$

Switching to master coordinates, we get,

$$\begin{cases} \displaystyle\int_{\hat{K}} \mathrm{curl}(\hat{\boldsymbol{E}}^p - \hat{\boldsymbol{E}})\ \mathrm{curl}\hat{\boldsymbol{\phi}}\ J^{-1} + \bar{a}_{ij}(\hat{E}_i^p - \hat{E}_i)\hat{\phi}_j\, d\boldsymbol{\xi} + \int_{\hat{K}} \bar{a}_{ij}\frac{\partial \hat{p}}{\partial \xi_i}\hat{\phi}_j\, d\boldsymbol{\xi} \ = 0 \qquad \forall \hat{\boldsymbol{\phi}} \\ \displaystyle\int_{\hat{K}} \bar{a}_{ij}(\hat{E}_i^p - \hat{E}_i)\frac{\partial \hat{\phi}}{\partial \xi_j}\, d\boldsymbol{\xi} \qquad\qquad\qquad\qquad\qquad\qquad = 0 \qquad \forall \hat{\phi} \ , \end{cases}$$

where metric $\bar{a}_{ij}$ results from the change of variables, and the relation between the master and physical element shape functions,

$$\bar{a}_{ij} = \frac{\partial \xi_i}{\partial x_k}\frac{\partial \xi_j}{\partial x_k}\, J \ ,$$

and J stands for the jacobian of the transformation from master to physical coordinates.

As jacobian (element size) $J \to 0$, the relative L^2-contribution to the $H(\mathrm{curl})$-norm vanishes. This suggests eliminating this term from the projection problem alltogether. The modified projection problem looks then as follows.

$$\left\{ \begin{array}{ll} \displaystyle\int_K \mathrm{curl}(\boldsymbol{E}^{\mathrm{p}} - \boldsymbol{E})\mathrm{curl}\boldsymbol{\phi}\, d\boldsymbol{x} \;+ \int_K \boldsymbol{\nabla} p\, \boldsymbol{\phi}\, d\boldsymbol{x} & = 0, \quad \forall \text{ element vector bubbles } \boldsymbol{\phi} \\ \displaystyle\int_K (\boldsymbol{E}^p - \boldsymbol{E})\boldsymbol{\nabla}\phi\, d\boldsymbol{x} & = 0, \quad \forall \text{ element scalar bubbles } \phi\,, \end{array} \right.$$

or in master coordinates,

$$\left\{ \begin{array}{ll} \displaystyle\int_{\hat{K}} \mathrm{curl}(\hat{\boldsymbol{E}}^p - \hat{\boldsymbol{E}})\, \mathrm{curl}\hat{\boldsymbol{\phi}}\, J^{-1}\, d\boldsymbol{\xi} + \int_{\hat{K}} \bar{a}_{ij} \frac{\partial \hat{p}}{\partial \xi_i} \hat{\phi}_j\, d\boldsymbol{\xi} & = 0 \qquad \forall \hat{\boldsymbol{\phi}} \\ \displaystyle\int_{\hat{K}} \bar{a}_{ij}(\hat{E}_i^p - \hat{E}_i) \frac{\partial \hat{\phi}}{\partial \xi_j}\, d\boldsymbol{\xi} & = 0 \qquad \forall \hat{\phi}\,, \end{array} \right.$$

Most importantly, by dropping the term, we enable commutativity of the interpolation operators. Indeed, if $\boldsymbol{E}$ is a gradient, $\boldsymbol{E} = \boldsymbol{\nabla} u$, then the first equation will imply that curl $\boldsymbol{E}^p = 0$, i.e. $\boldsymbol{E}^p = \boldsymbol{\nabla} u^{p+1}$, for some potential u^{p+1}. The second (constraint) equation is reduced then to the projection problem used to define the H^1-conforming projection,

$$\int_K \boldsymbol{\nabla}(u^{p+1} - u)\boldsymbol{\nabla}\phi\, d\boldsymbol{x} = 0, \quad \forall \phi\,.$$

Edge projections. The choice of the norm for the edge projections is more elaborate. There are two guiding principles: the trace theorem and the commutativity of interpolation operators. The commutativity of the interpolation operators (see the next section) implies that, for gradients $\boldsymbol{E} = \boldsymbol{\nabla} u$, the edge projections (18.2.5) must reduce to the $H^{\frac{1}{2}}$-projection of potential u. In other words, we must have,

$$\|\frac{\partial u}{\partial s}\|_e = |u|_{\frac{1}{2},e}\,.$$

One option, see [45], is to start with the $H^{-\frac{1}{2}}$ norm for the $H(\mathrm{curl})$ edge projections (Here space $H^{-\frac{1}{2}}$ is understood as the dual of space $H^{\frac{1}{2}}_{00}$). Utilizing the isomorphism,

$$\frac{\partial}{\partial s}\; :\; H^{\frac{1}{2}}/\mathbb{R} \to H^{-\frac{1}{2}}\,,$$

we can define then the $H^{\frac{1}{2}}$ seminorm for the H^1 edge projections,

$$|u|_{\frac{1}{2},e} := \|\frac{\partial u}{\partial s}\|_{-\frac{1}{2},e}\,.$$

In other words, the $H^{-\frac{1}{2}}$ norm implies a specific realization of the $H^{\frac{1}{2}}$ seminorm.

We can also proceed in the other direction. Projection (18.2.5) is equivalent to the variational condition,

$$(E_t^p - E_t, \phi)_{-\frac{1}{2},e} = 0, \quad \forall \phi \, .$$

For constant test functions $\phi = 1$, the $H^{-\frac{1}{2}}$ inner product reduces to the L^2-product, and the condition above implies that $E_t^p - E^p$ must be of zero average. Consequently, representing both E_t^p and E_t as sums of a common constant c and contributions of zero average,

$$E_t = c + E_{t,0}, \quad E_t^p = c + E_{t,0}^p,$$

we can integrate $E_{t,0}$ to obtain the corresponding potential,

$$u(x) = \int_0^x E_{t,0}(s) \, ds \, ,$$

project u onto $\mathcal{P}_{-1}^{p+1}$ in the $H^{\frac{1}{2}}$-seminorm,

$$u^{p+1} \in \mathcal{P}_{-1}^{p+1}, \quad |u - u^p|_{\frac{1}{2},e} \rightarrow \min \, ,$$

and differentiate the result of the projection to obtain $E_{t,0}^p$,

$$E_{t,0} \in \mathcal{P}_0^p, \quad E_{t,0} = \frac{\partial u^p}{\partial s} \, .$$

In this construction, the $H^{\frac{1}{2}}$-seminorm implies a specific realization of the $H^{-\frac{1}{2}}$ norm.

In practical computation presented in this book, we have used a simple weighted L^2-norm,

$$\|E_t\|_e^2 \approx \int_e |E_t|^2 \frac{ds}{d\xi} \, ds \, .$$

Recalling the transformation from the master to physical coordinates for the $H(\mathrm{curl})$-conforming elements, we have,

$$E_t = E_i \frac{\partial x_i}{\partial \xi} \left(\frac{ds}{d\xi} \right)^{-1} = \hat{E}_j \frac{\partial \xi_j}{\partial x_i} \frac{dx_i}{d\xi} \left(\frac{ds}{d\xi} \right)^{-1} = \hat{E}_j \frac{d\xi_j}{d\xi} \left(\frac{ds}{d\xi} \right)^{-1} = \hat{E}_t l_e \left(\frac{ds}{d\xi} \right)^{-1} \, ,$$

where $\hat{E}_t$ is the tangential component of $\hat{\boldsymbol{E}}$, and l_e is the length of the master element edge (equal one, except for the "sloped" edge of the master triangle, where $l_e = \sqrt{2}$. The transformation rule for the tangent component leads to the simple formula for the "edge norm" on the master edge,

$$\|E_t\|_e^2 = \int_{\hat{e}} |\hat{E}_t|^2 l_e \, d\hat{s} = \int_0^1 |\hat{E}_t|^2 l_e^2 \, d\xi \, .$$

Notice that for $E_t = \partial u / \partial s$, the norm reduces to the norm used in the H^1-interpolation.

The corresponding edge projection problem looks then as follows,

$$\int_0^1 (\hat{E}_t^p - \hat{E}_t) \phi \, d\xi = 0, \quad \text{for every } \phi \in \mathcal{P}^{p_e}(e) \, .$$

If we isolate explicitly the constant test functions, we obtain,

$$\left\{ \begin{array}{l} \displaystyle\int_0^1 (\hat{E}_t^p - \hat{E}_t)\, d\xi = 0, \\ \displaystyle\int_0^1 (\hat{E}_t^p - \hat{E}_t)\phi\, d\xi = 0, \quad \text{for every } \phi \in \mathcal{P}_0^{p_e}(e)\ , \end{array} \right. \tag{18.2.6}$$

where $\mathcal{P}_0^{p_e}(e)$ denotes polynomials of order less or equal p_e, with zero average.

We conclude this section with a formal restatement of the $H(\mathrm{curl})$-conforming projection-based interpolation on the physical element.

$$\left\{ \begin{array}{ll} \displaystyle\int_e (E_t^p - E_t)\, \phi \frac{ds}{d\xi}\, ds = 0, & \forall \phi \in \mathcal{P}^{p_e}(e), \quad \forall e \\ \displaystyle\int_K \mathrm{curl}\boldsymbol{E}\ \mathrm{curl}\boldsymbol{\phi}\, d\boldsymbol{x} = 0 & \forall \text{ vector bubble } \boldsymbol{\phi}\ , \\ \displaystyle\int_K (\boldsymbol{E}^p - \boldsymbol{E})\boldsymbol{\nabla}\phi = 0 & \forall \text{ scalar bubble } \phi\ . \end{array} \right. \tag{18.2.7}$$

The corresponding procedure on the master element is fully analogous. Simirarly as for the H^1-case, the two interpolation operators do not commute unless the transformation from the master to the physical element reduces to a simple similarity, $\boldsymbol{x} = h\boldsymbol{\xi}$. Even in the case of a rectangular element with lenghths $h_1 \neq h_2$, the two interpolation procedures yield different results. We shall return to this issue in the chapter on automatic hp-adaptivity.

18.3 De Rham diagram

With the projection-based interpolation operators in place, we have the following fundamental result.

Let Π, Π^{curl} denote the H^1- and $H(\mathrm{curl})$- conforming projection-based interpolation operators, and let P denote the L^2-projection. Let $\mathcal{W}$ and $\mathcal{Q}$ denote subspaces of $H^1(K)$ and $H(\mathrm{curl})$ incorporating appropriate regularity assumptions. The following de Rham diagram commutes.

$$\begin{array}{ccccc} \mathcal{W} & \xrightarrow{\boldsymbol{\nabla}} & \mathcal{Q} & \xrightarrow{\boldsymbol{\nabla}\times} & L^2 \\ \downarrow \Pi & & \downarrow \Pi^{\mathrm{curl}} & & \downarrow P \\ W_p & \xrightarrow{\boldsymbol{\nabla}} & \boldsymbol{Q}_p & \xrightarrow{\boldsymbol{\nabla}\times} & Y_p \quad . \end{array} \tag{18.3.8}$$

Here $W_p, \boldsymbol{Q}_p, Y_p$ denote the spaces of element shape functions corresponding to *any* of the discussed master elements: the quad element of variable order, and the two types of triangular elements. The interpolation does not depend upon the element being considered.

We show first that

$$\Pi^{curl}(\boldsymbol{\nabla} u) = \boldsymbol{\nabla}(\Pi u)\ .$$

We will demonstrate that $\boldsymbol{\nabla}(\Pi u)$ satisfies the conditions defining the $H(\mathrm{curl})$-interpolant. Indeed, if

$$E_t = \frac{\partial u}{\partial s}, \quad E_t^p = \frac{\partial u^p}{\partial s},$$

then,

$$\int_e (\frac{\partial u^p}{\partial s} - \frac{\partial u}{\partial s})\, ds = \int_e \frac{\partial (u^p - u)}{\partial s}\, ds = (u^p - u)(\boldsymbol{a}_{i+1}) - (u^p - u)(\boldsymbol{a}_i) = 0\,.$$

Here $\boldsymbol{a}_i, \boldsymbol{a}_{i+1}$ denote the end points (vertices) of the edge. We have shown that the gradient of the H^1-interpolant satisfies the first edge projection condition (18.2.6). The second condition follows from the fact that each polynomial with zero average, $\phi \in \mathcal{P}_0^{p_e}(e)$, can be integrated to obtain a polynomial ψ of one order higher, vanishing at the endpoints of the edge, $\psi \in \mathcal{P}_{-1}^{p_e+1}(e), \phi = \partial\psi/\partial s$. Consequently, the second edge projection condition in (18.2.6), is equivalent to,

$$\int_e (\frac{\partial u^p}{\partial s} - \frac{\partial u}{\partial s})\frac{\partial \psi}{\partial s}\, ds = 0, \quad \forall \psi \in \mathcal{P}_{-1}^{p_e+1}(e)\,.$$

We have shown that the tangential components of the gradient of the H^1-interpolant and the $H(\mathrm{curl})$-interpolant are identical along the edges of the element. To conclude the proof, it is sufficient to recall now the construction of the second step of the $H(\mathrm{curl})$- interpolation. For $\boldsymbol{E}^p = \boldsymbol{\nabla} u^{p+1}$, the curl projection condition is trivially satisfied (both curls are zero), and the second, constraint condition reduces to the projection condition defining the H^1-interpolant.

In order to demonstrate the commutativity of the second part of the diagram, we need to show that,

$$P\mathrm{curl}\boldsymbol{E} = \mathrm{curl}(\Pi^{curl}\boldsymbol{E})\,.$$

It is sufficient to show that

$$\int_K \mathrm{curl}(\boldsymbol{E}^p - \boldsymbol{E})\psi\, d\boldsymbol{x} = 0, \quad \forall \psi \in Y_p\,.$$

But any function $\psi \in Y_p$ can be decomposed into a constant, and a function with zero average,

$$\psi = c + \psi_0, \quad \int_K \phi_0\, d\boldsymbol{x} = 0\,.$$

In turn, it follows from the exact sequence property that there exists a function $\boldsymbol{\phi}_0 \in \boldsymbol{Q}_p$, with zero tangential trace (an $H(\mathrm{curl})$-bubble), such that $\mathrm{curl}\,\boldsymbol{\phi}_0 = \psi_0$. Orthogonality with ψ_0 follows thus from condition (18.2.7)$_2$.

Orthogonality with constants follows from condition (18.2.6)$_1$ and the Stokes Theorem. Indeed,

$$0 = \sum_{\hat{e}} \int_{\hat{e}} (\hat{E}_t^p - \hat{E}_t)\, d\hat{s} = \int_{\hat{K}} \mathrm{curl}_{\boldsymbol{\xi}}(\hat{\boldsymbol{E}}^p - \hat{\boldsymbol{E}})\, d\boldsymbol{\xi} = \int_K \mathrm{curl}(\boldsymbol{E}^p - \boldsymbol{E})\, d\boldsymbol{x}\,.$$

18.4 Shape functions

We conclude our discussion on master elements with the definition of sample shape functions for the three master elements forming the exact sequences: the Nédélec's triangles of second and first type, and the Nédélec's square element of the first type. We have postponed the definition of shape functions to after the introduction of the exact sequences. The exact sequence property is indeed the key to understand the construction of the shape functions for the edge elements.

Quadrilateral. The element has nine nodes: four vertex nodes, for mid-edge nodes, and the middle node. We associate the anisotropic order (p_h, p_v) with the middle node, and orders $p_i, i = 1, \ldots, 4$, with the mid-edge nodes. Recall that we assume the minimum rule,

$$p_1, p_3 \leq p_h, \quad p_2, p_4 \leq p_v \, .$$

The order of the nodes refers to the H^1-conforming element. The actual order of the $H(\mathrm{curl})$-conforming element is smaller.

There are no shape functions associated with element vertices. The mid-edge and middle nodes shape functions are defined as tensor products of 1D H^1-shape functions $\hat{\chi}_i(\xi)$ and 1D $H(\mathrm{curl})$ shape functions $\hat{\zeta}_j(\xi)$, equal to derivatives of the H^1-shape functions. More precisely,

$$\begin{aligned} \hat{\zeta}_1 &= \tfrac{d}{d\xi}\hat{\chi}_2 = 1 \\ \hat{\zeta}_{1+j} &= \tfrac{d}{d\xi}\hat{\chi}_{2+j}, j = 1, \ldots, p-1 \, . \end{aligned}$$

The square element *vector-valued* shape functions are defined as follows,

- p_i shape functions for each of the mid-edge nodes,

$$\begin{aligned} \hat{\boldsymbol{\phi}}_{1,j}(\xi_1, \xi_2) &= \Big(\hat{\zeta}_j(\xi_1)\hat{\chi}_1(\xi_2), 0\Big)^T, \qquad j = 1, \ldots, p_1 \\ \hat{\boldsymbol{\phi}}_{2,j}(\xi_1, \xi_2) &= \Big(0, \hat{\chi}_2(\xi_1)\hat{\zeta}_j(\xi_2)\Big)^T, \qquad j = 1, \ldots, p_2 \\ \hat{\boldsymbol{\phi}}_{3,j}(\xi_1, \xi_2) &= \Big(\hat{\zeta}_j(1-\xi_1)\hat{\chi}_2(\xi_2), 0\Big)^T, \quad j = 1, \ldots, p_3 \\ \hat{\boldsymbol{\phi}}_{4,j}(\xi_1, \xi_2) &= \Big(0, \hat{\chi}_1(\xi_1)\hat{\zeta}_j(1-\xi_2)\Big)^T, \quad j = 1, \ldots, p_2 \, , \end{aligned}$$

- $p_h(p_v - 1)$ "horizontal" $H(\mathrm{curl})$ *bubble shape functions* for the middle node,

$$\hat{\boldsymbol{\phi}}^h_{5,ij}(\xi_1, \xi_2) = \Big(\hat{\zeta}_i(\xi_1)\hat{\chi}_{2+j}(\xi_2), \, 0\Big)^T \quad i = 1, \ldots, p_h, \; j = 1, \ldots, p_v - 1 \, ,$$

- $(p_h - 1)p_v$ "vertical" $H(\mathrm{curl})$ *bubble shape functions* for the middle node,

$$\hat{\boldsymbol{\phi}}^v_{5,ij}(\xi_1, \xi_2) = \Big(0, \, \hat{\chi}_{2+i}(\xi_1)\hat{\zeta}_j(\xi_2)\Big)^T \quad i = 1, \ldots, p_h - 1, \; j = 1, \ldots, p_v \, .$$

Note the difference between first and third, and second and fourth edge shape functions, accounting for the counterclockwise orientation of the edges.

Triangle of the second type. We associate order p with the middle node and orders $p_i, i = 1, 2, 3$ with the element mid-edge nodes. The minimum rule implies that

$$p_i \leq p, i = 1, 2, 3 \, .$$

Recall that the element space of shape functions consists of all vector-valued polynomials of order $p-1$ such that their tangential traces on edge e_i reduce to polynomials of order $p_i - 1$. As for the H^1 element, we will define the shape functions in terms of affine coordinates:

$$\lambda_1 = 1 - \xi_1 - \xi_2, \quad \lambda_2 = \xi_1, \quad \lambda_3 = \xi_2 \, .$$

With i-th mid-edge node we associate p_i shape functions. The first one is known as the Whitney function,

$$\hat{\boldsymbol{\phi}}_{i,1} = \lambda_i \boldsymbol{\nabla}^T \lambda_{i+1} - \lambda_{i+1} \boldsymbol{\nabla}^T \lambda_i \, .$$

As usual, the edges are denumerated modulo 3. Function λ_i and its tangential derivative vanish along the edge opposite to the i-th vertex. Consequently, tangential component of the i-th Whitney shape function, vanishes along the two remaining edges. Along the i-th edge,

$$\frac{\partial \lambda_{i+1}}{\partial s} = -\frac{\partial \lambda_i}{\partial s} = \frac{1}{l_i}$$

and, therefore, the tangential component along the i-th edge is constant and equal to $1/l_i$, where l_i denotes the length of the edge,

$$\lambda_i \frac{\partial \lambda_{i+1}}{\partial s} - \lambda_{i+1} \frac{\partial \lambda_i}{\partial s} = (\lambda_i + \lambda_{i+1}) \frac{1}{l_i} = \frac{1}{l_i} \, .$$

The remaining $p_i - 1$ shape functions for the i-th mid-edge node, are defined as gradients of the corresponding H^1 shape functions,

$$\hat{\boldsymbol{\phi}}_{i,1+j} = \boldsymbol{\nabla}^T \hat{\phi}_{3+i,j} = \boldsymbol{\nabla}^T \left(\lambda_i \lambda_{i+1} \hat{\chi}_j^{ker} \left(\frac{\lambda_{i+1} - \lambda_i + 1}{2} \right) \right) \quad i = 1, 2, 3, \quad j = 1, \ldots, p_i - 1 \, .$$

Recall that kernel functions $\hat{\chi}_j^{ker}$ were obtained by dividing the 1D H^1 shape functions by the product of the first two linear shape functions,

$$\hat{\chi}_j^{ker}(x) = \frac{\hat{\chi}_{2+j}(x)}{x(1-x)} \, .$$

It is easy to see that the tangential component of $\hat{\boldsymbol{\phi}}_{i,1+j}$ along the i-th edge coincides with $\hat{\zeta}_{1+j} = d\hat{\chi}_{2+j}/d\xi$, and it vanishes along the two remaining edges.

Recall now that the middle node, H^1 bubble functions were defined as,

$$\lambda_1 \lambda_2^j \lambda_3^{q-j-1}, \quad j = 1, \ldots, q-2, \quad q = 3, \ldots, p \, .$$

The corresponding gradients are given by,

$$\lambda_2^j\lambda_3^{q-j-1}\boldsymbol{\nabla}\lambda_1 + j\lambda_1\lambda_2^{j-1}\lambda_3^{q-j-1}\boldsymbol{\nabla}\lambda_2 + (q-j-1)\lambda_1\lambda_2^j\lambda_3^{q-j-2}\boldsymbol{\nabla}\lambda_3,\ . \tag{18.4.9}$$

The simplest way to define the middle node, $H(\mathrm{curl})$-bubble functions, is to employ the three terms in the formula for the gradient above. Not all of them are linearly independent. For $q > 3$, it is sufficient to retain, e.g. only the first term corresponding to $\boldsymbol{\nabla}\lambda_1$ component. Indeed, we have for $i = j-1$,

$$\begin{aligned}
&\lambda_1\lambda_2^{j-1}\lambda_3^{q-j-1}\boldsymbol{\nabla}\lambda_2 + \lambda_1\lambda_2^i\lambda_3^{q-i-2}\boldsymbol{\nabla}\lambda_3\\
&= (1-\xi_1-\xi_2)\xi_1^{j-1}\xi_2^{q-j-1}(1,1)\\
&= \xi_1^{j-1}\xi_2^{q-j-1}(1,1) - \xi_1^j\xi_2^{q-j-1}(1,1) - \xi_1^{j-1}\xi_2^{q-j}(1,1)\,.
\end{aligned}$$

Consequently, $\lambda_2^j\lambda_3^{q-j-1}(1,1)$ can be represented in terms of $\lambda_2^{j-1}\lambda_3^{q-j}(1,1)$, lower order term $\lambda_2^{j-1}\lambda_3^{q-j-1}(1,1)$, and the corresponding terms with $\boldsymbol{\nabla}\lambda_2, \boldsymbol{\nabla}\lambda_3$.

Concluding, the middle node shape functions start with $p = 3$, and for each $q = 3,\dots,p$ are organized into three groups corresponding to $\boldsymbol{\nabla}\lambda_1, \boldsymbol{\nabla}\lambda_2, \boldsymbol{\nabla}\lambda_3$.

$$\begin{aligned}
\hat{\boldsymbol{\phi}}^1_{4,q} &= \lambda_2\lambda_3^{q-2}\boldsymbol{\nabla}^T\lambda_1\\
\hat{\boldsymbol{\phi}}^2_{4,q,j} &= \lambda_1\lambda_2^{j-1}\lambda_3^{q-j-1}\boldsymbol{\nabla}^T\lambda_2 \quad j = 1,\dots,q-2,\\
\hat{\boldsymbol{\phi}}^3_{4,q,j} &= \lambda_1\lambda_2^j\lambda_3^{q-j-2}\boldsymbol{\nabla}^T\lambda_3 \qquad j = 1,\dots,q-2\,.
\end{aligned} \tag{18.4.10}$$

Triangle of the first type. Recall that the main idea of the Nédélec triangle of the first type is to increase order of approximation for the element, without adding gradients.

The first natural step towards constructing such an element, therefore, is a separation of higher order shape functions into gradients and those with non-zero curl. No action is necessary for the mid-edge nodes, as only the Whitney functions produce non-zero curl. Switching from the triangle of the second type to the first type, threfore, involves no change in the definition of the edge shape functions.

Denoting by $\boldsymbol{P}^p_{-1}$ the space of all vector-valued polynomials of order p, with vanishing tangential trace on the boundary of the triangle, and by $\mathcal{P}^p_{-1}$ the space of all scalar-valued polynomials of order p with vanishing trace on the boundary, we record the dimensions of the involved polynomial spaces,

$$\begin{aligned}
\dim\boldsymbol{P}^p_{-1} &= (p+1)(p+2) - 3(p+1) = p^2-1\\
\dim\mathcal{P}^{p+1}_{-1} &= \frac{(p-1)p}{2}\\
\dim\boldsymbol{P}^p_{-1} - \dim\boldsymbol{P}^{p-1}_{-1} &= (p^2-1) - ((p-1)^2-1) = 2p-1\\
\dim\mathcal{P}^{p+1}_{-1} - \dim\mathcal{P}^p_{-1} &= \frac{(p-1)p}{2} - \frac{(p-2)(p-1)}{2} = p-1\,.
\end{aligned}$$

Consequently, when increasing the order of polynomials from order $p-1$ to order p, we have to add $p-1$ new "gradient" and p new "rotational" shape functions with non-zero gradient. Employing the scalar middle node shape functions discussed earlier, for each $q = 3, \ldots, p+1$, we define the gradient shape functions as follows,

$$\begin{aligned}\hat{\boldsymbol{\psi}}^1_{4,q,j} &= \boldsymbol{\nabla}(\lambda_1\lambda_2^j\lambda_3^{q-j-1}) \\ &= \lambda_2^j\lambda_3^{q-j-1}\boldsymbol{\nabla}\lambda_1 + j\lambda_1\lambda_2^{j-1}\lambda_3^{q-j-1}\boldsymbol{\nabla}\lambda_2 + (q-j-1)\lambda_1\lambda_2^j\lambda_3^{q-j-2}\boldsymbol{\nabla}\lambda_3, \\ & j = 1, \ldots, q-2\,.\end{aligned} \tag{18.4.11}$$

The rotational shape functions can be selected as follows (comp. [73]),

$$\begin{aligned}\hat{\boldsymbol{\psi}}^2_{4,q,j} &= \lambda_1\lambda_2^{j-1}\lambda_3^{q-j-1}\boldsymbol{\nabla}\lambda_2 - \lambda_1\lambda_2^j\lambda_3^{q-j-2}\boldsymbol{\nabla}\lambda_3 \\ &= \lambda_1\lambda_2^{j-1}\lambda_3^{q-j-2}(\lambda_3\boldsymbol{\nabla}\lambda_2 - \lambda_2\boldsymbol{\nabla}\lambda_3) \\ & j = 1, \ldots, q-2\,, \\ \hat{\boldsymbol{\psi}}^3_{4,q} &= \lambda_1^{q-3}\lambda_2\lambda_3\boldsymbol{\nabla}\lambda_1 - \lambda_1^{q-2}\lambda_3\boldsymbol{\nabla}\lambda_2 + \lambda_1^{q-3}\lambda_2\lambda_3\boldsymbol{\nabla}\lambda_1 - \lambda_1^{q-2}\lambda_2\boldsymbol{\nabla}\lambda_3 \\ &= 2\lambda_1^{q-3}\lambda_2\lambda_3\boldsymbol{\nabla}\lambda_1 - \lambda_1^{q-2}\lambda_3\boldsymbol{\nabla}\lambda_2 - \lambda_1^{q-2}\lambda_2\boldsymbol{\nabla}\lambda_3 \\ &= \lambda_1^{q-3}(2\lambda_2\lambda_3\boldsymbol{\nabla}\lambda_1 - -\lambda_1\lambda_3\boldsymbol{\nabla}\lambda_2 - \lambda_1\lambda_2\boldsymbol{\nabla}\lambda_3)\,.\end{aligned} \tag{18.4.12}$$

Notice that the functions above belong to the Nédélec's space of symmetric polynomials, comp. [73]. Defining the interior shape functions for the triangle of the first kind is now easy, one has simply to drop the gradient shape functions, retaining the rotational shape functions only.

From the point of view of p-refinements, the triangle of the first type provides thus an intermediate step. Switching from the triangle of the second kind of order p to the triangle of the first kind, we increase (symbolically) the order of the element from p to $p+\frac{1}{2}$. Adding gradients, we restore the complete polynomials of order $p+1$. We have not implemented the triangle of the first kind in the code, except for the Whitney element corresponding to $p = \frac{1}{2}$, restricting ourselves to complete polynomials only.

An alternative could be to use *only* triangles of the first kind. In such a case, it is not necessary to split the shape functions into the gradients and "non-gradients", see Exercise 18.2.

The 2D triangular shape functions discussed in this volume cannot be employed for face shape functions of a 3D tetrahedral element. Construction of face shape functions for tetrahedrals involves non-trivial connectivity issues motivating choosing families that display rotational symmetry. Similarly to H^1-conforming shape functions, one cannot select a set of hierarchical shape functions that would be "rotationally invariant". Notice that the rotational invariance for the $H(\mathrm{curl})$-conforming shape functions differs from that for the H^1-conforming functions. If $\boldsymbol{\phi}_i, i = 1, 2, 3$ denote shape functions from $\boldsymbol{P}^p_{-1}$ associated logically with vertices $\boldsymbol{a}_i,\ i = 1, 2, 3$, and $\boldsymbol{y} = \boldsymbol{B}\boldsymbol{x} + \boldsymbol{b}$ is the affine map mapping triangle with vertices $\boldsymbol{a}_i,\ i = 1, 2, 3$ into triangle with vertices $\boldsymbol{a}_{i+1},\ i = 1, 2, 3$, the

$H(\mathrm{curl})$ rotational invariance means that,

$$\boldsymbol{\phi}_{i+1}(\boldsymbol{y}) = \boldsymbol{B}^{-T}(\boldsymbol{y})\boldsymbol{\phi}_i(\boldsymbol{x}), \text{ where } \boldsymbol{y} = \boldsymbol{B}\boldsymbol{x} + \boldsymbol{b}\,.$$

Finally, we emphasize that none of the sets of shape functions discussed so far is optimal from the point of view of conditioning of element matrices. Construction of such shape functions requires an additional research.

Exercises

Exercise 18.1 Prove that Nédélec's space is affine invariant.

More precisely, let $\boldsymbol{x} \to \boldsymbol{y} = \boldsymbol{B}\boldsymbol{x} + \boldsymbol{b}$ denote a non-singular affine map from $\mathbb{R}^n$ into itself. Let $\hat{\boldsymbol{E}} = \hat{\boldsymbol{E}}(\boldsymbol{x})$ be a symmetric polynomial of order p, i.e.,

$$\boldsymbol{x} \cdot \hat{\boldsymbol{E}}(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x}\,.$$

Define,

$$E_i(\boldsymbol{y}) = \sum_j \hat{E}_j(\boldsymbol{x}) \frac{\partial x_j}{\partial y_i}, \quad \boldsymbol{y} = \boldsymbol{B}\boldsymbol{x} + \boldsymbol{b}\,.$$

Show that,

$$\boldsymbol{y} \cdot \boldsymbol{E}(\boldsymbol{y}) = 0, \quad \forall \boldsymbol{y}\,.$$

The affine invariance of Nédélec's space implies that, for an affine element, the element space of shape functions is independent of the order in which the vertices of the element has been denumerated. Notice that, consistenly with the transformation rule for $H(\mathrm{curl})$-conforming elements, this invariance is *different* from the invariance for H^1-conforming elements, expressed frequently in terms of "rotation operator" corresponding to the affine map mapping triangle vertices $\boldsymbol{a}_i$ into $\boldsymbol{a}_{i+1}$ (mod 3).

Exercise 18.2 According to the discussion in [73], a set of non-hierarchical middle node shape functions for the Nédélec triangle of the first kind, can be constructed by considering the family of integer indices $(i,j,k), i,j,k \geq 1, i+j+k = p$. For each triple (i,j,k) we have then two corresponding shape functions,

$$\lambda_1^{i-1}\lambda_2^j\lambda_3^k\boldsymbol{\nabla}\lambda_1 - \lambda_1^i\lambda_2^{j-1}\lambda_3^k\boldsymbol{\nabla}\lambda_2 \text{ and } \lambda_1^i\lambda_2^{j-1}\lambda_3^k\boldsymbol{\nabla}\lambda_2 - \lambda_1^i\lambda_2^j\lambda_3^{k-1}\boldsymbol{\nabla}\lambda_3\,.$$

Try to visualize the set of indices as points in a three-dimensional system of coordinates.

Prove that a hierarchical basis can now be obtained by projecting the points on the $\lambda_1 = 1$ plane,

$$\lambda_2^{q-1-j}\lambda_3^j\boldsymbol{\nabla}\lambda_1 - \lambda_1\lambda_2^{q-2-j}\lambda_3^j\boldsymbol{\nabla}\lambda_2 \text{ and } \lambda_1\lambda_2^{q-2-j}\lambda_3^j\boldsymbol{\nabla}\lambda_2 - \lambda_1\lambda_2^{q-1-j}\lambda_3^{j-1}\boldsymbol{\nabla}\lambda_3,$$
$$j = 1,\ldots,q-2,\ q = 3,\ldots,p-1\,.$$

Notice that the basis functions of order q span rotational shape functions of order q and gradient shape functions of order $q-1$. Inncreasing the order of elements from, e.g. $1\frac{1}{2}$ to $2\frac{1}{2}$, involves adding four new shape functions of order 3 which span the gradient $\boldsymbol{\nabla}(\lambda_1\lambda_2\lambda_3)$. Find explicitly the linear combination of the four shape functions representing the gradient.

Exercise 18.3 Splitting of the homogenenous polynomials of order p with vanishing tangential trace, into gradients and "non-gradients" may be done in many different ways. Prove that the following shape functions are linearly independent.

$$\begin{aligned}
\hat{\boldsymbol{\phi}}^1_{4,q} &= \lambda_2\lambda_3^{q-2}\boldsymbol{\nabla}^T\lambda_1 \\
\hat{\boldsymbol{\phi}}^2_{4,q,j} &= \lambda_2^j\lambda_3^{q-j-1}\boldsymbol{\nabla}^T\lambda_1 + j\lambda_1\lambda_2^{j-1}\lambda_3^{q-j-1}\boldsymbol{\nabla}^T\lambda_2 + (q-j-1)\lambda_1\lambda_2^j\lambda_3^{q-j-2}\boldsymbol{\nabla}^T\lambda_3, \\
&\quad j=1,\ldots,q-2, \\
\hat{\boldsymbol{\phi}}^3_{4,q,j} &= \lambda_2^j\lambda_3^{q-j-1}\boldsymbol{\nabla}^T\lambda_1 + j\lambda_1\lambda_2^{j-1}\lambda_3^{q-j-1}\boldsymbol{\nabla}^T\lambda_2 - (q-j-1)\lambda_1\lambda_2^j\lambda_3^{q-j-2}\boldsymbol{\nabla}^T\lambda_3, \\
&\quad j=1,\ldots,q-2\,,
\end{aligned}$$

for $q = 3,\ldots,p$. Show that the rotational shape functions are not, however, symmetric polynomials.

Selecting the non-gradients of order $q = p+1$,

$$\begin{aligned}
\hat{\boldsymbol{\phi}}^1_{4,q} &= \lambda_2\lambda_3^{q-j-1}\boldsymbol{\nabla}^T\lambda_1 \\
\hat{\boldsymbol{\phi}}^3_{4,q,j} &= \lambda_2^j\lambda_3^{q-j-1}\boldsymbol{\nabla}^T\lambda_1 + j\lambda_1\lambda_2^{j-1}\lambda_3^{q-j-1}\boldsymbol{\nabla}^T\lambda_2 - (q-j-1)\lambda_1\lambda_2^j\lambda_3^{q-j-2}\boldsymbol{\nabla}^T\lambda_3, \\
&\quad j=1,\ldots,q-2\,.
\end{aligned}$$

we define a Nédélec-type triangle of the first kind. Many shape functions introduced in the engineering literature, see e.g. [121], fall into this category.

Chapter 19

2D Maxwell Code

In this chapter we discuss modifications of the elliptic code turning it into a Maxwell code. In fact, the code is set up to solve any system of equations involving unknowns in H^1, approximated with H^1-conforming (continuous) elements, and unknowns in $H(\mathrm{curl})$, approximated with $H(\mathrm{curl})$-conforming (edge) elements. It is assumed that the order of approximation for the edge elements is determined by the exact sequence property. The code supports Nédeléc quads of the first type, and Nédeléc triangles of the second type. The triangles of the second type have not been coded. Since the order of edge elements is implied by the order of the continuous elements, there is no need to store a separate information on the order of edge elements. Whenever, we speak of elements of order p, we mean the order of H^1-conforming element, the order of the corresponding $H(\mathrm{curl})$-conforming element is $p-1$ for triangles and $(p_h-1,p_v)\times(p_h,p_v-1)$ for quads.

The logic of the code has been set up in such a way that operations on nodes have been separated from operations on the corresponding d.o.f. Both types of the discretization, inluding the whole adaptivity package, share the same operations at the nodal level. For instance, refining an element consists of breaking its nodes, which involves generating new nodes. These topological operations are identical for both types of the discretization. Only a manipulation with d.o.f. reflects the type of discretization being considered.

We begin by listing additional directories added to the elliptic code and discuss changes made in the data structure module, and data structure supporting routines. Next we present the element routine corresponding to the time-harmonic Maxwell equations. Discussion on constrained approximation and the concept of the modified element follows. We conclude the chapter with a ”do list” to set up a data for the Maxwell problem, and discuss an important example: scattering of a plane wave on a screen.

19.1 Directories. Data structure

Compared with the elliptic code, two additional directories have been added:

- *adaptE* - routines executing automatic hp-refinements for the $H(\mathrm{curl})$-conforming discretization,
- *maxwell* - element routines for the Maxwell problem, including a collection of routines evaluating the exact solution to a model diffraction problem.

The Maxwell code can be obtained by preprocessing the .F source code routines with flag *EM_MODE=1*. Most of significant Maxwell problems involve complex-valued solutions, so flag *C_MODE* should be set to one. The real version of the code has been used only for debugging. As we proceed with the discussion of the code, we advocate studying the .F routines rather then the resulting Maxwell code .f routines. That way, it is easy to follow the changes and realize the differences between the two versions.

We begin by visiting data structure module - *module/data_ structure2D*. Compared with the elliptic code, a number of new attributes have been added.

The new attributes of the mesh include:

- *NRDOFSE* - total number of d.o.f. corresponding to edge elements and a single (vector-valued) unknown,
- *NREQNE* - number of unknowns discretized with edge elements.
- *MAXEQNE* - maximum number of solution components discretized with edge elements.

For the Maxwell problem, *NREQNE* = 1, and parameter *MAXEQNE*=2 is set through the parameter statement. This is because, similarly to *MAXEQNS*, parameter *MAXEQNE* is used to dimension arrays in the code. Value of *MAXEQNE* must be twice of *NREQNE*, if the automatic adaptivity is to be used. *NRDOFSE* is used only for statistics, in particular for computing and drawing the convergence rates.

The definition of the in initial mesh object and vertex node remain unchanged, The definition of the (higher order) node object includes just one more attribute - $H(\mathrm{curl})$ d.o.f. *zdofE*.

Changes made in the data structure supporting routines from directory *datstrs: celndof,dumpin ,dumpout,fdoflq,fdoflt* and two debug printing routines *datstrs/print_quad_dof, print_trian_dof* are quite self-explanatory, they are all related to the new attribute of nodes - $H(\mathrm{curl})$-conforming d.o.f. *zdofE*. Note that in some cases, changes involve the heads of routines, see e.g. routines *datstrs/celndof,fdoflq,fdoflt*.

Finally, we introduce two additional global flags in *module/control* read from file *files/control*: Flag *PEN_FLAG* = 1, if the stabilized variational formulation is used, for the standard formulation *PEN_FLAG* = 0. The additional flag *PEN_FLAG_INTERP* controls the choice of the element projection problem in the projection-based interpolation procedure. We shall discuss it in the chapter on automatic hp-adaptivity for Maxwell equations.

19.2 The element routine

Please open routine *maxwell/elem*. Ihe routine evaluates element stiffness matrix and load vector, both for the standard and stabilized variational formulations. We begin by recalling the transformation formulas for gradients of H^1 shape functions, $H(\mathrm{curl})$ shape functions, and curl of $H(\mathrm{curl})$ shape functions,

$$\begin{aligned}\frac{\partial \phi}{\partial x_i} &= \frac{\partial \hat{\phi}}{\partial \xi_k}\frac{\partial \xi_k}{\partial x_i} \\ \psi_i &= \hat{\psi}_k \frac{\partial \xi_k}{\partial x_i} \\ \mathrm{curl}_{\boldsymbol{x}} \boldsymbol{\psi} &= \left(\mathrm{curl}_{\boldsymbol{\xi}} \hat{\boldsymbol{\psi}}\right) J^{-1} .\end{aligned}$$

Here $\boldsymbol{x} = \boldsymbol{x}(\boldsymbol{\xi})$ is the map transforming master element into physical element, $\partial \xi_k / \partial x_i$ denotes the corresponding inverse jacobian matrix, and J stands for the jacobian.

For the standard variational formulation, the element contributions to the stiffness matrix and load vector are evaluated as follows.

$$\begin{aligned}B_{K,ij} &= \int_K \left\{ \frac{1}{\mu} \mathrm{curl}\boldsymbol{\psi}_i \mathrm{curl}\boldsymbol{\psi}_j - (\omega^2 \epsilon - i\omega\sigma)\psi_{i,k}\psi_{j,k} \right\} d\boldsymbol{x} \\ &= \int_{\hat{K}} \left\{ \frac{1}{\mu} J^{-1} \mathrm{curl}\hat{\boldsymbol{\psi}}_i \mathrm{curl}\hat{\boldsymbol{\psi}}_j - (\omega^2 \epsilon - i\omega\sigma)\bar{a}_{mn}\hat{\psi}_{i,m}\hat{\psi}_{j,n} \right\} d\boldsymbol{\xi} \\ &\approx \sum_l \left\{ \frac{1}{\mu} J^{-1}(\boldsymbol{\xi}_l) \mathrm{curl}\hat{\boldsymbol{\psi}}_i(\boldsymbol{\xi}_l) \mathrm{curl}\hat{\boldsymbol{\psi}}_j(\boldsymbol{\xi}_l) - (\omega^2 \epsilon - i\omega\sigma)\bar{a}_{mn}(\boldsymbol{\xi}_l)\hat{\psi}_{i,m}(\boldsymbol{\xi}_l)\hat{\psi}_{j,n}(\boldsymbol{\xi}_l) \right\} w_l \\ l_{K,j} &= -i\omega \int_K J_k^{imp} \psi_{j,k}\, d\boldsymbol{x} \\ &= -i\omega \int_{\hat{K}} \left(J_k^{imp} \frac{\partial \xi_m}{\partial x_k} J \right) \hat{\psi}_{j,m}\, d\boldsymbol{\xi} \\ &\approx -i\omega \sum_l \left(J_k^{imp} \frac{\partial \xi_m}{\partial x_k} J \right)(\boldsymbol{\xi}_l) \hat{\psi}_{j,m}(\boldsymbol{\xi}_l)\, w_l\end{aligned}$$

where

$$\bar{a}_{mn} = \frac{\partial \xi_m}{\partial x_k}\frac{\partial \xi_n}{\partial x_k} J ,$$

and $\boldsymbol{\xi}_l, w_l$ denotes the quadrature points and weights for the master element. Note that $\psi_{i,k}$ denotes the k-th component of the (vector-valued) i-th shape function.

Let e be an element edge parametrized through the master element in the usual way,

$$\boldsymbol{x}(\xi) = \boldsymbol{x}(\boldsymbol{\xi}(\xi)), \quad \xi \in (0,1)\,.$$

The tangent unit vector is then evaluated as,

$$\tau_k = \frac{\partial x_k}{\partial \xi} |\frac{d\boldsymbol{x}}{d\xi}|^{-1}$$

Consequently, the tangent component of shape function $\boldsymbol{\psi}$ transforms as,

$$\begin{aligned}
\psi_t &= \psi_k \tau_k \\
&= \hat{\psi}_m \frac{\partial \xi_m}{\partial x_k} \frac{\partial x_k}{\partial \xi} |\frac{d\boldsymbol{x}}{d\xi}|^{-1} \\
&= \hat{\psi}_m \frac{\partial \xi_m}{\partial \xi} |\frac{d\boldsymbol{x}}{d\xi}|^{-1}
\end{aligned}$$

The element boundary integral contribution corresponding to an edge $e \subset \Gamma_C$ to the element stiffness matrix is now evaluated as follows.

$$\begin{aligned}
B_{e,ij} &= i\omega \int_e \gamma \psi_{i,t} \psi_{j,t}\, ds \\
&= i\omega \int_0^1 \gamma \left(\frac{\partial \xi_m}{\partial \xi} \frac{\partial \xi_n}{\partial \xi} |\frac{d\boldsymbol{x}}{d\xi}|^{-1} \right) \hat{\psi}_{i,m} \hat{\psi}_{j,n}\, d\xi \\
&\approx i\omega \sum_l \gamma \left(\frac{\partial \xi_m}{\partial \xi} \frac{\partial \xi_n}{\partial \xi} |\frac{d\boldsymbol{x}}{d\xi}|^{-1} \right) (\xi_l) \hat{\psi}_{i,m}(\xi_l) \hat{\psi}_{j,n}(\xi_l)\, w_l
\end{aligned}$$

Similarly, the edge $e \subset \Gamma_C \cup \Gamma_N$ contribution to the element load vector is given by the formula,

$$\begin{aligned}
l_{e,ij} &= i\omega \int_e J_S^{imp} \psi_{j,t}\, ds \\
&= i\omega \int_0^1 \left(J_S^{imp} \frac{\partial \xi_n}{\partial \xi} \right) \hat{\psi}_{j,n}\, d\xi \\
&\approx i\omega \sum_l \left(J_S^{imp} \frac{\partial \xi_n}{\partial \xi} \right) (\xi_l) \hat{\psi}_{j,n}(\xi_l)\, w_l
\end{aligned}$$

In the case of the stabilized formulation (17.2.11), we need to calculate additionally stiffness matrix D_{ij} and load vector m_j. Consistently with the way we have arrived at the additional equation, formulas for both matrices can be obtained by substituting gradients of Lagrange multiplier p and

test function q for fields $\boldsymbol{E}, \boldsymbol{F}$. The element integral contributions are as follows.

$$\begin{aligned}
D_{K,ij} &= -(\omega\epsilon - i\sigma)\int_K \frac{\partial\phi_i}{\partial x_k}\psi_{j,k}\,d\boldsymbol{x} \\
&= -(\omega\epsilon - i\sigma)\int_{\hat{K}} \bar{a}_{mn}\frac{\partial\hat{\phi}_i}{\partial\xi_m}\hat{\psi}_{j,n}\,d\boldsymbol{\xi} \\
&\approx -(\omega\epsilon - i\sigma)\sum_l \left\{ \bar{a}_{mn}(\boldsymbol{\xi}_l)\frac{\partial\hat{\phi}_i}{\partial\xi_m}(\boldsymbol{\xi}_l)\hat{\psi}_{j,n}(\boldsymbol{\xi}_l) \right\} w_l \\
m_{K,j} &= -i\int_K J_k^{imp}\frac{\partial\phi_j}{\partial x_k}\,d\boldsymbol{x} \\
&= -i\int_{\hat{K}} \left(J_k^{imp}\frac{\partial\xi_m}{\partial x_k}J \right)\frac{\partial\hat{\phi}_j}{\partial\xi_m}\,d\boldsymbol{\xi} \\
&\approx -i\sum_l \left(J_k^{imp}\frac{\partial\xi_m}{\partial x_k}J \right)(\boldsymbol{\xi}_l)\frac{\partial\hat{\phi}_j}{\partial\xi_m}(\boldsymbol{\xi}_l)\,w_l
\end{aligned}$$

Similarly, the element boundary integral contribution corresponding to an edge e are,

$$\begin{aligned}
D_{e,ij} &= i\int_e \gamma\frac{\partial\phi_i}{\partial t}\psi_{j,t}\,ds \\
&= i\int_0^1 \gamma\left(\frac{\partial\xi_m}{\partial\xi}\frac{\partial\xi_n}{\partial\xi}|\frac{d\boldsymbol{x}}{d\xi}|^{-1} \right)\frac{\partial\hat{\phi}_i}{\partial\xi_m}\hat{\psi}_{j,n}\,d\xi \\
&\approx i\sum_l \gamma\left(\frac{\partial\xi_m}{\partial\xi}\frac{\partial\xi_n}{\partial\xi}|\frac{d\boldsymbol{x}}{d\xi}|^{-1} \right)(\xi_l)\frac{\partial\hat{\phi}_i}{\partial\xi_m}(\xi_l)\hat{\psi}_{j,n}(\xi_l)\,w_l \qquad e\subset\Gamma_C \\
m_{e,ij} &= i\int_e J_S^{imp}\frac{\partial\phi_j}{\partial t}\,ds \\
&= i\int_0^1 \left(J_S^{imp}\frac{\partial\xi_n}{\partial\xi} \right)\frac{\partial\hat{\phi}_j}{\partial\xi_n}\,d\xi \\
&\approx i\sum_l \left(J_S^{imp}\frac{\partial\xi_n}{\partial\xi} \right)(\xi_l)\frac{\partial\hat{\phi}_j}{\partial\xi_n}(\xi_l)\,w_l \qquad e\subset\Gamma_C\cup\Gamma_N\,.
\end{aligned}$$

Notice that, in order to improve the stability, we have "incorporated" one ω into functions p and q.

The matrices corresponding to forms $b(\boldsymbol{E},\boldsymbol{F}), d(p,\boldsymbol{F}), d^*(\boldsymbol{E},q)$, and $l(\boldsymbol{F}), m(q)$ are "packed" into a single "big" stiffness matrix, and a single load vector,

$$\begin{pmatrix} \boldsymbol{B} & \boldsymbol{D} \\ \boldsymbol{D}^T & \boldsymbol{0} \end{pmatrix} \qquad \begin{pmatrix} \boldsymbol{l} \\ \boldsymbol{m} \end{pmatrix}.$$

The algorithm for computing the element matrix and load vector is very much the same as for the elliptic case, and we will not repeat all the details here. Upon accessing element order of approximation and geometry d.o.f., the routine proceeds in two major steps: computation of element contributions, and evaluation of contributions from edges. Each of the two parts begins with the loop through the element integration points. Once at an integration point, we compute all geometry data, i.e. the corresponding physical coordinates, Jacobian matrix, jacobian and the inverse Jacobian

matrix. Next we access the material and load data and compute the "modified material and load data" including the geometric quantities. Finally, we loop through the element shape functions and accumulate for the element matrices. For the stabilized formulation, the formal algorithm looks as follows.

```
for each H(curl) test shape function j = 1,...,N
  accumulate for the l part of the load vector:
    l_j^big = l_j^big + ...
  for each H(curl) trial shape function i = 1,...,N
    accumulate for the B part of the stiffness matrix:
      B_{i,j}^big = B_{i,j}^big + ...
  end of loop through H(curl) trial shape functions
  for each H^1 trial shape function i = 1,...,M
    accumulate for the D part of the stiffness matrix:
      B_{N+i,j}^big = B_{N+i,j}^big + ...
  end of loop through H^1 trial shape functions
end of loop through H(curl) test shape functions
for each H^1 test shape function j = 1,...,M
  accumulate for the m part of the load vector:
    l_{N+j}^big = l_{N+j}^big + ...
  for each H(curl) trial shape function i = 1,...,N
    accumulate for the D^T part of the stiffness matrix:
      B_{i,N+j}^big = B_{i,N+j}^big + ...
  end of loop through H(curl) trial shape functions
end of loop through H^1 test shape functions
```

Notice that the part corresponding to the H^1 trial *and* H^1 test functions is missing.

19.3 Constrained approximation. Modified element

1D h-extension operator. Recall the discussion from Section 5.1, and consider a function defined on 1D element $(0,1)$, represented in terms of 1D $H(\mathrm{curl})$ shape functions ζ_j evaluated in routine *elem_util/shape1E*,

$$E(x) = \sum_{j=1}^{p} E_j \zeta_j(x), \quad x \in [0,1]\,.$$

Upon refining the element into two element sons, the function can be represented in terms of the small element shape functions,

$$E(x) = \sum_{j=1}^{p} e_j^1 \zeta_j(\xi_1), \quad \xi_1 = 2x, x \in [0, \frac{1}{2}]$$
$$E(x) = \sum_{j=1}^{p} e_j^2 \zeta_j(\xi_2), \quad \xi_2 = 2(x - \frac{1}{2}), x \in [\frac{1}{2}, 1]$$

By comparing the two representations at p uniformly distributed collocation points, for each small element, we can formulate a system of p linear equations to be solved for coefficients e_j^1 or e_j^2. The operation is performed in routine *constr_util/setcnstrE* with the resulting coefficients stored in array *RRRE* in common block *commons/crrrE*. The relation between the small and big element d.o.f. is then as follows:

$$e_k^I = \sum_{j=k}^{p} RRRE(1, j, I, k) E_j \ .$$

As in the elliptic case, the first and third index in array $RRRE$ correspond to the mid-edge nodes of the big (just one) and (two) small elements, the second index corresponds to the big element shape functions and the fourth one to the small mid-edge node shape functions. Notice that the hierarchical construction of shape functions implies that $RRRE(1, j, I, k) = 0$ for $j < k$.

2D h-extension operators. With the quad element $H(\mathrm{curl})$ shape functions defined as tensor products of 1D $H(\mathrm{curl})$- and 1D H^1-shape functions, the matrix representations of extension operators for $h4$ and $h2$ refinements are constructed in terms of products of 1D arrays RRR and $RRRE$, comp. Exercise 19.2. The situation is more complicated with the triangular element, for which the shape functions are not expressed in terms of tensor products of 1D shape functions. The matrix representation of the $H(\mathrm{curl})$ h-extension operator is determined in routine *constr_util/setcnstrtE* using a similar collocation method as in 1D, and stored in array $RRTE$, in common block *commons/crrrE*. The denumeration of the constrained nodes, depicted in Fig. 19.1, is unfortunately inconsistent with the denumeration of son nodes in the data structure (another legacy issue...).

Adaptivity. The extension operators discussed above are used in routines *meshmods/zini-dof_medg, zinidof4_mdlq, zinidof2_mdlq, zinidof4_mdlt* initiating d.o.f. for new nodes generated in process of breaking father element nodes. New d.o.f. corresponding to p-refinements, are initiated simply with zeros, see *meshmods/nodmod*. Except for the additional operations with the $H(\mathrm{curl})$ d.o.f., the mesh modification routines are identical with those for the elliptic code.

Modified element. Constrained approximation. The entire logic leading to the concept of the modified element is identical with the elliptic case. The crucial routine *constrs/history* remains

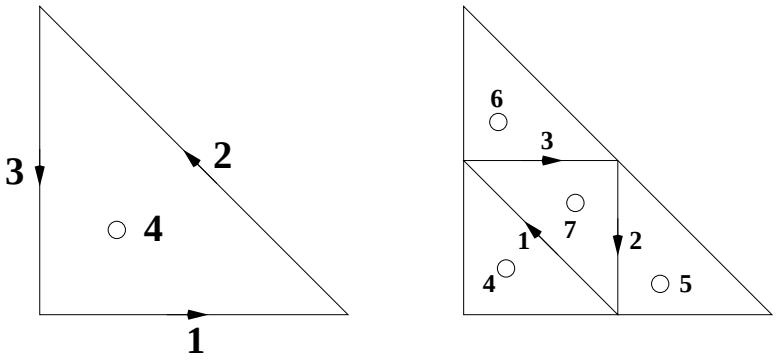


Figure 19.1: Denumeration of parent and constrained nodes corresponding to the $h4$ refinement of a triangular element

unchanged as it deals only with elements and nodes, and *not* nodal d.o.f. The content of module *module/tree* is unaffected, too. Routines *constrs/logicq*, *logict* are expanded to include additional arrays handling the constrained approximation for $H(\mathrm{curl})$ d.o.f.

- number of $H(\mathrm{curl})$ d.o.f. for the modified element non-vertex nodes,
  Ndof2E(i), i=1,...,Nrnod

- number of "connected" parent modified element d.o.f. for each local d.o.f.,
  NrconE(k),k=1,...,N

- d.o.f. connectivities for the local d.o.f.,
  NacE(j,k),j=1,...,NrconE(k),k=1,...,N, and

- the corresponding constrained approximation coefficients,
  ConstrE(j,k),j=1,...,NrconE(k),k=1,...,N

Similarly to the elliptic case, the actual constrained approximation coefficients are determined in routine *constrs/logic2* "translating" the constraints info from the nodal to the d.o.f. level.

The arrays listed above are used to compute the modified element matrices in routine *constrs/celem* and to evaluate local element d.o.f. from the global ones, in routine *constrs/ solelm*. The algorithms are identical with those for the elliptic case. Similarly to the elliptic case, the transformation from the local to the modified element matrices, and from the global to local element d.o.f. includes taking care of the orientation of the mid-edge nodes and the sign factors. If the local and global orientations for an (unconstrained) mid-edge nodes are opposite, the transformation from the global to the local d.o.f. is as follows,

$$E_j^{local} = (-1)^j E_j^{global} .$$

Notice that for the H^1-conforming d.o.f. of a mid-edge node, the corresponding factor was $(-1)^{j-1}$. The difference is easily understood when one remembers that the $H(\mathrm{curl})$ d.o.f. of a mid-edge node correspond to gradients of H^1 shape functions (the tangent coordinate is reversed).

Interface with the frontal solver. The logic of the frontal solver requires that the denumeration of an element d.o.f. is implied by the denumeration of element nodes, i.e. we list the nodes first and then, for each node, we list the corresponding nodal d.o.f. As we have decided to keep the element $H(\mathrm{curl})$- and H^1- d.o.f. separated (this corresponds naturally to the logic of the mixed variational formulation), the higher order nodes have to be listed twice, first for the edge element and then for the elliptic element d.o.f. Consequently, when we define the nodal nicknames for the prefront in *solver1/solve1*, we first consider the $H(\mathrm{curl})$ (only higher order) nodes, and then the H^1 vertex and higher order nodes. The same order of nodes must be maintained in routine *solver1/solout* where the (modified) element d.o.f. are received from the frontal solver and stored in the data structure arrays.

In our discussion, we have focused on the stabilized formulation. In the case of the standard variational formulation, all operations on H^1 d.o.f. are simply skipped.

19.4 Setting up a Maxwell problem

In this section, we will use the problem of scattering a plane wave on a screen, to guide the reader through the process of preparing the input file and routines specifying the load data.

19.4.1 Example: scattering of a plane wave on a screen

The problem is illustrated in Fig. 19.2. A plane wave shines an an infinite screen at a 45 degress angle (from south west). The exact solution is represented in terms of Fresnel integrals.

$$H(r,\theta) = \frac{1}{\sqrt{\pi}} e^{\frac{\pi j}{4} - jkr} \left\{ F\left[(2kr)^{\frac{1}{2}} \sin \frac{1}{2}(\theta - \frac{\pi}{4}) \right] + F\left[(2kr)^{\frac{1}{2}} \sin \frac{1}{2}(\theta + \frac{\pi}{4}) \right] \right\}$$

$$F(u) = \frac{1}{2}\sqrt{\pi} \left\{ e^{\frac{\pi j}{4}} - \sqrt{2} \left[C\left(\sqrt{\frac{2}{\pi}} u \right) - jS\left(\sqrt{\frac{2}{\pi}} u \right) \right] \right\}$$

$$C(z) - jS(z) := \int_0^s e^{-\frac{1}{2}\pi j t^2}\, dt \quad \text{(Fresnel integrals)}$$

The solution represents the most singular scenario for 2D homogeneous problems. In order to avoid trouble with modeling an unbounded domain (see the next chapter), we cut off a unit square computational domain around the diffracting edge, and use the exact solution to impose Dirichlet boundary conditions. The material data are $\epsilon = \mu = \omega = 1, \sigma = 0$.

Two input files have been prepared for the problem: *files/inputs/input_diffrq, input_diffrt*. The first one contains a description of the geometry in terms of GMP rectangles, the second in terms

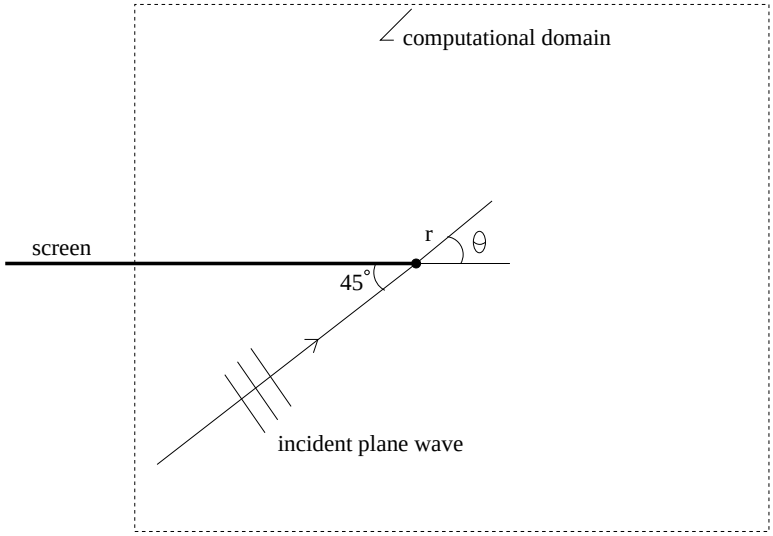


Figure 19.2: The diffraction problem: scattering of a plane wave on a screen

of GMP triangles. Notice that modeling a screen (a "crack" in the domain), involves duplicating a point and a line, e.g. points 4 and 5 in *input_diffrq* have identical coordinates. The data for the mesh generator has the same format as for the elliptic problems, except for the line specifying parameters *MAXELIS,MAXVERS,MAXNODS,NREQNS,NREQNE* where the new parameter: *NREQNE* - number of $H(curl)$-conforming unknowns, has to be added. For Maxwell problems, *NREQNE* = 1. Remember that $H(curl)$-conforming unknowns are vector-valued with the number of components equal to the space dimension.

Routines evaluating the Fresnel integrals (downloaded from ???) have been deposited in directory *fresnel*. The exact solution is evaluated in routine *maxwell/fresnel* which is called from routine *maxwell/exact*. The latter contains a number of additional exact solutions including a number of simple "manufactured solutions" used for debugging the code and one more non-trivial solution for the problem of scattering a plane wave on a cylinder discussed later.

Consistently with the mode of operation for the case when the exact solution is known, routine *maxwell/dirichletE* specifies the Dirichlet data by calling routine *exact*. Same comment applies, in general, to routine *maxwell/getJ_S* returning the prescribed surface current (Neumann data). In the discussed problem, only Dirichlet boundary conditions are used, so the routine *getJ_S* is irrelevant. In the case of the routine *maxwell/getJ* which returns the impressed volume current, we take a "short cut". As the equation is homogeneous, it makes little sense to code the second order derivatives of a rather complicated solution to sum them up to obtain the zero value. Instead, we have introduced an extra common block (/cexact/) with variable *NR_EXACT*. Value *NR_EXACT* = 1 indicates a homogenous problem, and causes routine *maxwell/getJ* to return zero impressed volume current. Consequently, the second order derivatives of the exact solution returned by routine *maxwell/exact*

are "faked" with zero values. Notice that all manufactured solutions used for debugging the code, do have the second derivatives evaluated, and the impressed volume current in routine *getJ* is evaluated by calling the exact solution routine.

Finally, routine *maxwell/getmatE* returns the simple, constant material data.

Obviously, for real life problems, the exact solution flag in *files/control* is set to zero, and the routines *maxwell/dirichletE, getJ, getJ_S* must be provided by the user. For simple problems a number of "if then else" statements will work but, for more complicated ones, one has to build a data base containing the material and load data. It is natural to build such a data base around the GMP. With piece-wise constant data and the requirement that FE meshes should conform to material interfaces, it is natural e.g. to assume that, for each GMP entity (triangle or rectangle in 2D), the corresponding material data and impressed volume currents are constant. In the same way, the value of the impedance constant and the impressed surface currents may be assumed to be constant for each GMP curve and then stored accordingly.

We have not coded the impedance boundary condition, leaving it as an exercise for the reader.

At this point, we are ready to run the code in the non-adaptive mode. The main menu allows for solving the problem and evaluating the $H(\mathrm{curl})$ error. Real and imaginary parts of both components of the approximate as well as the exact solution can be visualized. Other options including displaying the error and the Lagrange multiplier are possible. Routine *graph_util/soldis* can be further developed for allowing to display additional quantities of interest.

Fig. 19.3 displays for example the real part of the first component of the exact solution.

Exercises

Exercise 19.1 If the 1D mid-edge node $H(\mathrm{curl})$ shape functions $\zeta_j, j = 2, \ldots, p$ are defined as the derivatives of the 1D mid-edge node H^1 shape functions, what is the relation between coefficients RRR and $RRRE$?

Exercise 19.2 Construct matrix representations for the h extension operator of quads and $H(\mathrm{curl})$ shape functions. Consider the three refinement types shown in Fig. 19.4, and depicted there denumeration for the parent and constrained nodes. Determine matrices,

$$RRQ11E(nodp, jp, nod, j), \quad RRQ10E(nodp, jp, nod, j), \quad RRQ01E(nodp, jp, nod, j),$$

in terms of products of 1D extension operators matrices RRR and $RRRE$. Here $nodp = 1, \ldots, 5$ denumerates parent nodes, jp indicates the parent node d.o.f., nod corresponds to constrained nodes, and j denumerates the son node d.o.f. For instance, the j-th d.o.f. of nod-th son node corresponding to the horizontal $h2$-refinement, is to be computed using the

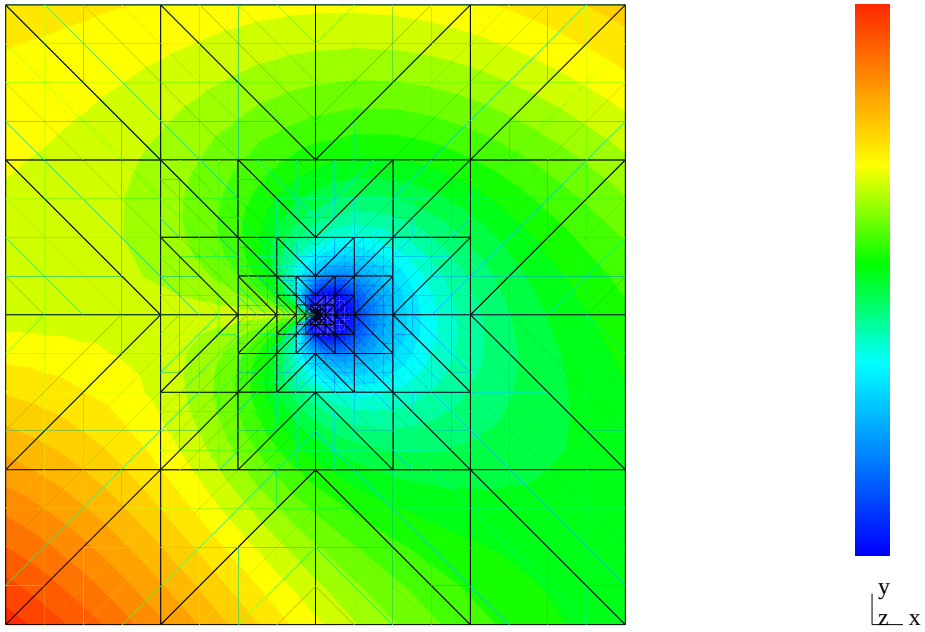


Figure 19.3: The diffraction problem: real part of component E_1 of the exact solution

following formula,

$$E_{nod,j} = \sum_{nodp} \sum_{jp} RRQ01E(nodp, jp, nod, j) E_{nodp,jp} \, ,$$

where $E_{nodp,jp}$ denotes the jp-th d.o.f. of $nodp$-th parent node.

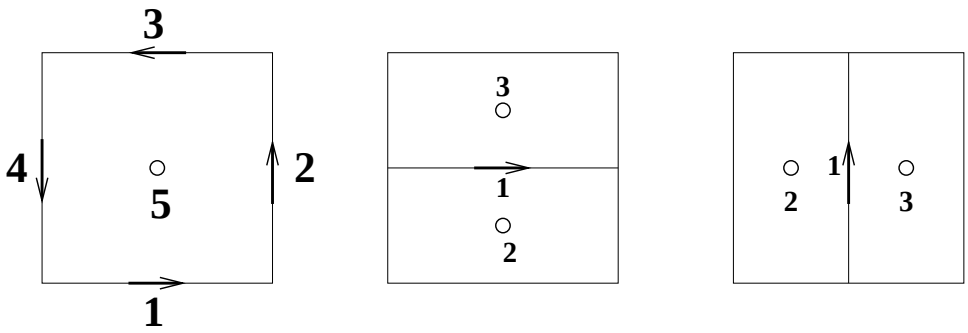


Figure 19.4: Denumeration of parent and constrained nodes corresponding to various h refinements of a quad element

Exercise 19.3 Code the impedance (Cauchy) boundary condition. Introduce an additional routine

maxwell/get_gamma and add the evaluation of the boundary integral contributions to the element routine. Test the code using a manufactured solution.

Exercise 19.4 Develop a data base for "real-life" material and load data, following the discussion in the text. Introduce an extra module with necessary objects and dimension them with GMP parameters. Discuss possible tests for debugging the data base.

Chapter 20

hp-Adaptivity for Maxwell Equations

The mesh optimization algorithm for Maxwell problems builds on the idea of the projection-based interpolation introduced in Chapter 18.2. The interpolation error exhibits the same convergence rates in both element size h and order of approximation p as the actual approximation error but, contrary to the approximation error, it depends upon the mesh only *locally*. Changes in the discretization over an element affect the value of the interpolation error for the element only. This local dependence enables the construction of the hp optimization algorithm using the same logic as for the elliptic problems.

Obviously, we cannot interpolate the unknown exact solution, and we have to replace it with some approximation that is significantly more accurate that the Finite Element (FE) solution itself. The additional "scales" must allow for making the choice between h and p-refinements, so the idea of using the global hp-refinement, and the corresponding fine grid solution suggests itself.

Introducing the fine grid solution enables a perfect a-posteriori error estimate for the coarse grid solution - the difference between the fine and coarse grid solution provides an excellent approximation to the error function.

There is no proof that the algorithm delivers exponential convergence, in fact we cannot show even that it must converge at all. All claims made here are based on numerical experiments only. It is in this context that we strive at least to achieve a certain generality of the idea - the same logic should apply to the 1D version and to both 2D (elliptic or Maxwell) versions of the algorithm.

We shall begin with an additional discussion on the projection-based interpolation for the $H(\mathrm{curl})$-conforming problems addressing the possibility of interpolation on master vs. physical element, followed by a detailed description of the algorithm and conclude with numerical experiments for the diffraction problem discussed in the previous chapter.

20.1 Projection-based interpolation revisited

Let us begin by reiterating the main "operational" points about the projection based interpolation procedure defined by (18.2.7).

- The projections are done hierarchically, first on edges and then over the elements (in 3D: over edges, faces and elements). The result of each step constitutes a Dirichlet-like boundary condition for the next step, in this sense we are talking about *constrained projections*. Use of projections as opposed to classical interpolation techniques (evaluation at nodes etc.) enables interpolating functions with low regularity including the fine grid solution.

- The linear system corresponding to the projection over the element $(18.2.7)_{2,3}$ is overdetermined, since, in general, we do not attempt seperation of element shape functions into gradients and non-gradients [1] and its solution requires introducing the Lagrange multiplier.

- The linear projection $(18.2.7)_{2,3}$ is obtained from the standard $H(\mathrm{curl})$-projection with element size $h \to 0$. Only in the limit, however, we obtain a linear projection that makes the de Rham diagram commute.

- The projection-based interpolation can be done both over the master and the physical elements. Contrary to classical interpolation procedures [32], the two operators, in general *do not* commute.

The last point leaves us with a dilemna: should we interpolate over the physical or over the master element. Intuitively, the interpolant obtained by projecting over the actual elements should approximate the actual approximate solution better. Its main drawback is the cost of the local projections, the stiffness matrix corresponding to the projection depends upon the geometry of an individual element, and it has to be computed "on a fly". The critique does not apply to affine elements. In this case the jacobian of the element map is constant which implies that the projections over the master and physical elements yield identical results.

The biggest advantage of interpolating over the master element is the possibility of precomputing the stiffness matrices for the projection problems, and the resulting significantly lower cost of the whole procedure.

In the end, the cost issue has prevailed in the implementation presented in this section, and we have decided to interpolate over the master element. In the case of rectangular elements though, we have included the possibility of projecting over the actual rectangular rather than square master

[1] Simplifi cation of the interpolation procedure provides an additional argument for such a construction

element. In the case of the anisotropically scaled quad element, the $H(\mathrm{curl})$-projection reduces to,

$$\int_{\hat{K}} \left\{ \mathrm{curl}(\hat{\boldsymbol{E}}_{hp} - \hat{\boldsymbol{E}})\, \mathrm{curl}\boldsymbol{F} + h_2^2(\hat{E}_{1,hp} - \hat{E}_1)\hat{F}_1 + h_1^2(\hat{E}_{2,hp} - \hat{E}_2)\hat{F}_2 \right\} d\xi_1 d\xi_2 = 0,$$
$$\text{for every vector bubble } \hat{\boldsymbol{F}} . \tag{20.1.1}$$

Here h_1, h_2 denote the horizontal and vertical element size. More precisely, the element map is of the form,

$$x_1 = a_1 + h_1\xi_1, \quad x_2 = a_2 + h_2\xi_2 .$$

The stabilized formulation looks as follows,

$$\begin{cases} \displaystyle\int_{\hat{K}} \left\{ \mathrm{curl}(\hat{\boldsymbol{E}}_{hp} - \hat{\boldsymbol{E}})\, \mathrm{curl}\boldsymbol{F} + h_2^2(\hat{E}_{1,hp} - \hat{E}_1)\hat{F}_1 + h_1^2(\hat{E}_{2,hp} - \hat{E}_2)\hat{F}_2 \right\} d\xi_1 d\xi_2 \\ \quad + \displaystyle\int_{\hat{K}} \left\{ \left(\frac{h_2}{h_1}\right) \frac{\partial \hat{p}}{\partial \xi_1} F_1 + \left(\frac{h_1}{h_2}\right) \frac{\partial \hat{p}}{\partial \xi_2} F_2 \right\} d\xi_1 d\xi_2 = 0, \\ \qquad\qquad \text{for every element vector bubble } \hat{\boldsymbol{F}} , \\ \displaystyle\int_{\hat{K}} \left\{ \left(\frac{h_2}{h_1}\right) (\hat{E}_{1,hp} - \hat{E}_1) \frac{\partial \hat{v}}{\partial \xi_1} + \left(\frac{h_1}{h_2}\right) (\hat{E}_{2,hp} - \hat{E}_2) \frac{\partial \hat{v}}{\partial \xi_2} \right\} d\xi_1 d\xi_2 = 0, \\ \qquad\qquad \text{for every element scalar bubble } \hat{v} . \end{cases} \tag{20.1.2}$$

Finally, dropping the L^2-term in the element projection altogether, we recover our definition of the projection-based interpolation,

$$\begin{cases} \displaystyle\int_{\hat{K}} \left\{ \mathrm{curl}(\hat{\boldsymbol{E}}_{hp} - \hat{\boldsymbol{E}})\, \mathrm{curl}\boldsymbol{F} \; + \; \left(\frac{h_2}{h_1}\right) \frac{\partial \hat{p}}{\partial \xi_1} F_1 + \left(\frac{h_1}{h_2}\right) \frac{\partial \hat{p}}{\partial \xi_2} F_2 \right\} d\xi_1 d\xi_2 = 0, \\ \qquad\qquad \text{for every element vector bubble } \hat{\boldsymbol{F}} , \\ \displaystyle\int_{\hat{K}} \left\{ \left(\frac{h_2}{h_1}\right) (\hat{E}_{1,hp} - \hat{E}_1) \frac{\partial \hat{v}}{\partial \xi_1} + \left(\frac{h_1}{h_2}\right) (\hat{E}_{2,hp} - \hat{E}_2) \frac{\partial \hat{v}}{\partial \xi_2} \right\} d\xi_1 d\xi_2 = 0, \\ \qquad\qquad \text{for every element scalar bubble } \hat{v} . \end{cases} \tag{20.1.3}$$

The three discussed procedures differ only in the second step, the interpolation along the edges in all three cases is identical. Notice that the anisotropy of the element does not affect the curl term.

Generalized projection-based inetrpolation. In context of the mesh optimization algorithm, the projection-based interpolation procedure described above has to be understood in a generalized sense. The projections are done over the coarse element edges and interiors on the polynomial space of the original element, as well as a p- or h-refined elements. In the latter case, we project on *piece-wise* polynomials resulting from the h-refinements. The "element bubbles" are understood as *piece-wise* polynomials with vanishing traces on the coarse element edges.

Trouble with the constrained nodes. In the case of meshes with hanging nodes, the projection-based interpolation is not longer a hundred percent local procedure. If the interpolant is to be

globally conforming, we must project only over "big"edges and use the constrained approximation technique to compute the corresponding d.o.f. for the constrained vertices and "small" edges. We use the procedure when we generate geometry d.o.f., compare routine *hp_interp/update_gdof*. We used the same algorithm in our original implementation of the automatic hp-adaptivity for the elliptic problems [58]. An attempt to use the same philosophy for the $H(\mathrm{curl})$-conforming elements produces a global interpolation procedure that *does not* satisfy the commuting diagram property. A typical situation is illustrated in Fig. 20.1. A partially constrained coarse grid element, upon the global hp-refinement, gives rise to a patch of four element-sons with constrained edges. The edge d.o.f. for the element sons are not longer determined by performing the projections on the *small edges*. Instead, the projections are done on the *big edges*, and the d.o.f. corresponding to small edges are determined by using the extension operators (compare the discussion on the constrained approximation). Consequently, the difference between the fine grid solution and its interpolant, $E_t - E_t^p$, is no longer of zero average over the constrained small element edges. This property is crucial in enforcing the commuttativity of interpolation operators. So what? Why should this matter? A

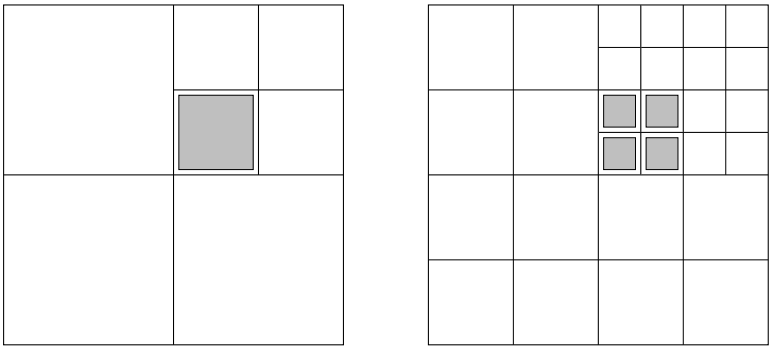


Figure 20.1: A partially refined coarse grid element results in a patch of small elements with constrained edges

simple answer is that the numerical experiments have shown that the mesh optimization procedure has failed.... The commuttativity property implies that the mesh optimization procedure for the $H(\mathrm{curl})$-conforming elements, when applied to a function with zero curl (gradient of a potential for simply connected domains), will reduce to the algorithm for the H^1-conforming elements. Thus, by enforcing the commuttativity property, we generalize the H^1 algorithm without changing it. This is the best explanation we can give at this point...

Restoring the commuttavity can be done by eliminating the projections over partially constrained elements in the coarse grid. In other words, the interpolation over the coarse grid would be done by performing projections also over patches consisting of one, two, three or even four elements. This procedure has been used in the proof on hp-convergence of Maxwell eigenvalues presented in [**?**]. The corresponding fine grid patches would consist then of up to sixteen small elements leading to a programmer's nightmare. So the only reasonable alternative seems to be to give up the global

conformity. Contrary to the determination of the geometry d.o.f. that must be globally conforming, the projection-based interpolation is used here to determine only optimal hp-refinements. We shall comment on this point again in the next section presenting the hp-algorithm.

20.2 The hp mesh optimization algorithm

We proceed now with the description of one step of the hp-algorithm. Along with the discussion on the algorithm, we shall commment on implementation issues. Implementing the hp algorithm for $H(\mathrm{curl})$-conforming elements involved modifying some of the existing routines in directory *adapt* using the preprocessing flags, and writing a collection of new routines that can be found in directory *adaptE*.

The overall logic of the algorithm, represented by the main driver *adapt/hp_strategy*, is identical for both elliptic and Maxwell problems. Starting with an initial mesh, we produce a sequence of optimally refined coarse hp grids and corresponding fine grids that deliver the exponential convergence. The number of iterations is controlled by a desired error tollerance for the coarse grid solution and a maximum number of iterations specified by the user.

The initial mesh is produced by the initial mesh generator using data from the input file, and may incorporate additional refinements done interactively by the user. In fact, the mesh optimization procedure can begin with any mesh, which allows for restarting the hp algorithm as many times as you wish.

Given a coarse mesh, we solve the problem on the mesh, and initiate a separate data structure to support the algorithm, see *module/hp_refinements*. We also shift the solution components in the main data structure arrays to make room for the fine grid solution (this is why the parameters *MAXNREQNS* and *MAXNREQNE* must be twice the value of *NREQNS* and *NREQNE*). We dump the coarse grid data structure to the disk. Next we refine the coarse grid globally in h and p, and solve the problem on the fine mesh. We evaluate now the norm of the difference between the coarse and fine grid solutions, storing at the same time the fine grid solution in *module/hp_refinements* (routine *adapt/compute_error*). If the difference is smaller than the desired tollerance, we terminate the mesh optimization algorithm. The fine grid solution is our final product. If not, we dump back in the coarse grid data structure, and continue with the algorithm described next.

Step 1: Determining optimal edge refinements and guaranteed error decrease rates. The logic of this step is identical with the logic of the 1D algorithm and the edge optimization for the elliptic problems. For each unconstrained edge of the coarse grid, we compute the projection-based interpolants of the fine grid solution on:

- the coarse grid edge space,

- p-refined coarse grid edge space,
- a sequence of piece-wise polynomial spaces corresponding to various h-refinements of the edge with edge-sons of order p_1, p_2.

The h-refinements need a detailed explanation. With the order of edge-sons limited to p, we have a total of p^2 combinations. For a hexahedral element of order p this number grows to p^8. Obviously, in general, we cannot afford checking all possibilities. Instead, we first check on *competetive refinements*. By a competetive h refinement we mean a refinement that adds the same number of d.o.f. as the p-refinement - just one for edges. More precisely,

$$(p_1 - 1) + (p_2 - 1) + 1 = p\,,$$

where p denotes the coarse grid edge order. This accounts for a total of p projections. Once we determine the best competetive h-refinement delivering the smallest interpolation error, we do not stop, but continue increasing the order of edge sons following the so called *biggest element error refinement path* until the original order p is reached for both edge sons. The concept was already discussed in the chapters on 1D and 2D adaptivity for elliptic problems, compare Fig. **??**.

The competetive h-refinements are compared with the p-refined element, to determine whether the edge should be p- or h-refinement. The continued h refinements beyond the competetive ones, are used to determine the *guaranteed errror decrease rate*. Recall the discussion about the "missing scales" from the previous chapters.

Step 2: Determining which edges should be refined. Step 1, described above, is purely local. The guaranteed error decrease rates, for each individual edge, are now compared with each other, to determine the *maximum edge error decrease rate*,

$$\text{max rate } = \max_e \{\text{rate of edge } e\}$$

Each edge whose guaranteed rate is within the 1/3 of the maximum rate is then selected for the h or the p-refinement. Notice that edges with the "missing scale" may be p-refined, even though the corresponding guaranteed edge rate may have corresponded to an h-refinement adding more than one d.o.f. This is related to the fact that those h-refinements have never been checked against p-refinements increasing order p by more than one. After Step 2, we know which edges to refine and we know whether we should increase their order or break them.

Step 4: Enforcing the desired edge h-refinements. Isotropy flags. An edge is broken only if all elements adjacent to the edge are h-refined. In the case of a quadrilateral, the element may be broken anisotropically into two element-sons, or isotropically into four element-sons. The decision

how to break a quad element is based on computing the so called *isotropy flags*. The flags are determined by examining the structure of the element error measured in the $H(\mathrm{curl})$ norm,

$$\underbrace{\int_{\hat{K}} |\frac{\partial e_2}{\partial \xi_1}|^2 J^{-1} + a_{11}|e_1|^2 \, d\boldsymbol{\xi}}_{C_{11}} + \underbrace{\int_{\hat{K}} |\frac{\partial e_1}{\partial \xi_2}|^2 J^{-1} + a_{22}|e_2|^2 \, d\boldsymbol{\xi}}_{C_{22}}$$
$$+ \underbrace{\int_{\hat{K}} (-\frac{\partial e_1}{\partial \xi_2}\frac{\partial \bar{e}_2}{\partial \xi_1} - \frac{\partial e_2}{\partial \xi_1}\frac{\partial \bar{e}_1}{\partial \xi_2}) J^{-1} + (a_{12} e_1 \bar{e}_2 + a_{21} e_2 \bar{e}_1) \, d\boldsymbol{\xi}}_{C_{12}} .$$

Here $\boldsymbol{e} = \boldsymbol{E}_{h/2,p+1} - \boldsymbol{E}_{h,p}$ is the difference between the fine and coarse grid solutions, and a_{mn} denotes the metric resulting from the change of coordinates,

$$a_{mn} = \sum_{k=1}^{2} \frac{\partial \xi_m}{\partial x_k} \frac{\partial \xi_n}{\partial x_k} J \, .$$

If absolute value of contribution C_{12} does not exceed one percent of $C_{11} + C_{22}$ and contribution C_{11} is greater than 99 percent of $C_{11} + C_{22}$, the element isotropy flag is set to 10, indicating the need of refining the element vertically only. Similarly, if contribution C_{22} prevails, the flag is set to 01 indicating the need of a horizontal refinement. Otherwise the flag is set to 11 indicating the isotropic refinement. The threshold value of 99 percent reflects the philosophy that the anisotropic refinements should be reserved for boundary layers only.

We loop now through all coarse grid elements, and determine the desired h-refinement flag implied by the edge refinement flags. If an element is set to be broken only anisotropically, we check the element *isotropy flag*. If the flag = 11, the desired h-refinement flag is upgraded to the isotropic $h4$-refinement.

Once the element flags have been determined, the elements are h-refined accordingly, executing the 1-irregular rule. The isotropy flags are used one more time to determine the minimal unwanted h-refinements, compare routine *meshmods/refine*.

At this point, the topology of the new coarse grid has been fully determined and it remains to set the order of approximation for the refined edges and element interiors.

Step 5: Determining an optimal order p for the h-refined edges. We visit all h-refined edges to determine the orders for the edge sons. These include edges that "have won the competition" with p-refinement, and those that have been refined involuntary, as a result of the 1-irregular meshes algorithm.

This part of the algorithm may be compared with an investment strategy. Suppose we have got unlimited funds to invest. We have visited a number of banks and have learned about the maximum interest rates they offer for a specific, fixed amount of funds, say one milion dollars. Given the unlimited resources, we decide to invest into every bank that has offered us a rate within a reference

rate that we fix at the level of the one third of the maximum rate. Obviously, in the bank that has offered us the best rate, we try to invest more than one million dollar. Upon learning that the rate decreases with the amount we deposit, we have to decide how much we want to invest. A strategy consistent with our 1/3 of the max rate policy, is to keep on investing into the bank until the offered rate drops below the reference rate. After all, we are already investing in less competetive banks that have met this criterion.

Our interest rates in the investment story are the rates of decrease of the interpolation error,

$$\frac{\|\boldsymbol{E}-\boldsymbol{E}_{ref}\|-\|\boldsymbol{E}-\boldsymbol{E}_{h,p}\|}{\text{nrdof added}}\,.$$

Here $\boldsymbol{E}$ is the fine grid solution, $\boldsymbol{E}_{ref}$ is the projection-based interpolant of the fine grid solution on a *reference grid*, and $\boldsymbol{E}_{h,p}$ is the interpolant on any grid with a number d.o.f. greater than in the reference grid, "tested for investment". Our choice of the reference grid reflects the assumption that the new coarse grid should always be at least as good as the old one - the interpolation error should not increase. This is automatically satisfied for any orthogonal projection, if the old coarse grid and new coarse grid spaces are nested. In the case of an h-refined edge, this is not necessarily the case. We begin with the most competetive distribution of orders determined in Step 1 and keep on increasing the orders of approximation for the edge-sons following the *biggest subelement error path* discussed earlier. The first grid that delivers an error smaller or equal than the old coarse grid[2] is selected as the reference grid. Once we have selected the reference grid, we start computing the rate and stop the "investment process" on the last grid that delivers a rate within the 1/3 of the max rate. Quite often, for *unwanted* h-refinements, the process stops simply at the reference grid. The selection process is done for all edges that have been h-refined, voluntary or not. For edges that have been selected for the p-refinement, we simply increase the order of approximation, and edges that were not selected for any refinement, and have not been involuntary refined, are left alone.

Step 6. Determining the optimal order of approximation for coarse grid element interiors. In this last step, we determine the order of approximation for elements in the new coarse grid. The logic is again fixed around the old coarse grid elements that may or may have not been h-refined. Possible scenarios are illustrated in Fig. 20.2. For an unrefined triangular element, we need to determine only one parameter - order p of its middle node. For a refined triangle, we need to determine four parameters - orders of the element-sons $p_1,\ldots,p_4$. The order of the three edges lying in the interior of the coarse grid element, is implied by the *minimum rule*. For quads the situation is more complicated. Each of the subelement orders involves two parameters: the "horizontal" p_h and "vertical" order p_v. The optimization process involves thus two parameters for an unrefined element, four parameters for an anisotropically refined element, and eight parameters for an element that has been refined into four element-sons. In 3D, for a hexahedral element, this

[2] In practice, to avoid the round off error effects, we match 0.99 percent of the old coarse grid error

number grows to $3 \times 8 = 24$ parameters ! This clearly indicates why a global search for the best distribution of polynomial orders is not feasible.

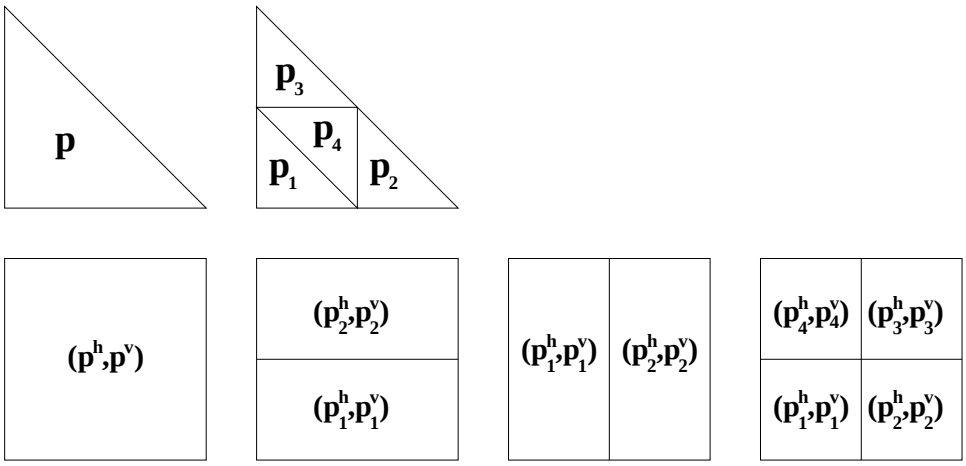


Figure 20.2: Polynomial orders for coarse grid element-sons determined in Step 6 of the mesh optimization procedure

We use exactly the same logic as for edges. We visit first all elements and compute the corresponding *guaranteed rates*. The procedure involves again following the maximum subelement error path, and determining the reference grid used for the computation of the rate. For quadrilateral elements, we have additionally to decide whether the order should be increased *isotropically*, i.e. from (p_h, p_v) to $(p_h + 1, p_v + 1)$, or *selectively* in one direction only, i.e. to $(p_h + 1, p_v)$ or $(p_h, p_v + 1)$. This is done be examining the directional structure of the error within the subelement, in the same way that we have used for computing the element *isotropy flags*. Also, in order to speed up the optimization process, the polynomial order is increased not just in one subelement with the largest error contribution but all subelements that deliver an error within 70 percent of the maximum subelement error (squared). The parameters controling the optimization process are completely arbitrary. An anisotropic increase in polynomial order may result in no new d.o.f. added. For instance, increasing order from $(2, 1)$ to $(3, 1)$ for a quad element keeps still the middle node "empty". In such a case, the order is increased isotropically to assure adding new d.o.f. The optimization procedure starts with orders implied by the *minimum rule* and the already known distribution of orders for the coarse element edges, see Fig. 20.3 for an example.

In the first sweep over the elements, we keep increasing the order for the element-sons until a maximum order is reached. For p-refined elements, the maximum order is fixed at level $p + 1$, for h-refined elements, at p level. For anisotropically refined elements, the maximum order is fixed in both directions independently. After the *guaranteed rate* has been determined for each element, we compute the maximum rate and follow our standard investment strategy. All elements are revisited, and the process of adding new degrees of freedom is stopped once the element rate drops below the

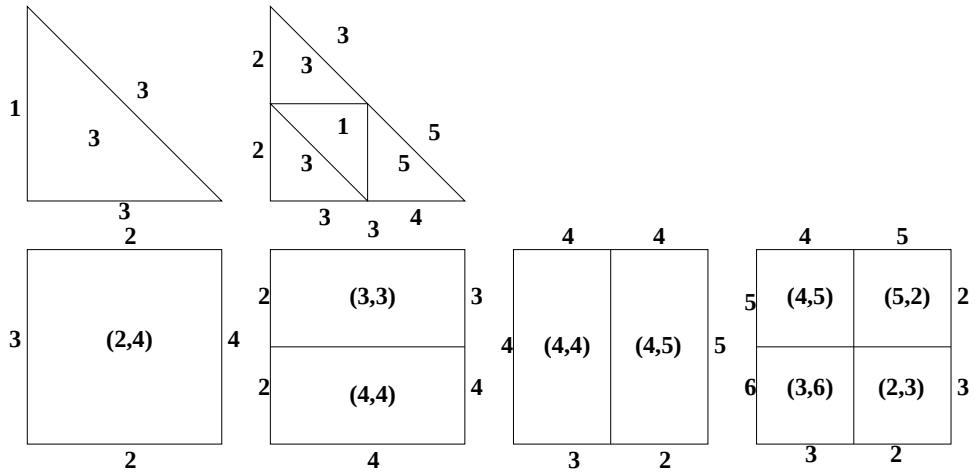


Figure 20.3: Step 6 of the hp algorithm: starting values for polynomial orders for element-sons derived from the orders for edges using the minimum rule

1/3 of the maximum rate.

Step 6 completes the mesh optimization process. The new coarse grid has been determined, and the optimization algorithm is executed recursively until either a prescribed error tollerance is met or specified maximum number of iterations reached. We always view the fine grid solution as our final product, although the error estimate is provided only for the coarse grid.

20.3 Example: The screen problem

We return now to the example discussed in the previous section – scattering of a plane wave on a half-line representing an infinite screen. Recall that the solution represents the most severe singularity encountered in 2D homogeneous problems. In order to avoid trouble with modeling an open boundary, we cut off a unit square computational domain around the diffracting edge, and use the exact solution to impose Dirichlet boundary conditions. The material data are $\epsilon = \mu = \omega = 1, \sigma = 0$.

Starting with an initial mesh of 4×4 quadrilateral elements of third order (remember that we refer always to the order of H^1-conforming elements), we push the algorithm to deliver a rather academic error of .01 percent. Figures 20.4, 20.5 present eight consecutive zooms on the diffracting edge in the final, optimal mesh.

Figures 20.6,20.7 present the corresponding zooms on the imaginary part of the first component of the electric field. Only at the level of the smallest elements, one can see the typical discontinuity artifacts related to the edge element discretization.

Finally, Fig. 20.8 presents the convergence history on an algebraic - log scale, for three different element projections discussed earlier:

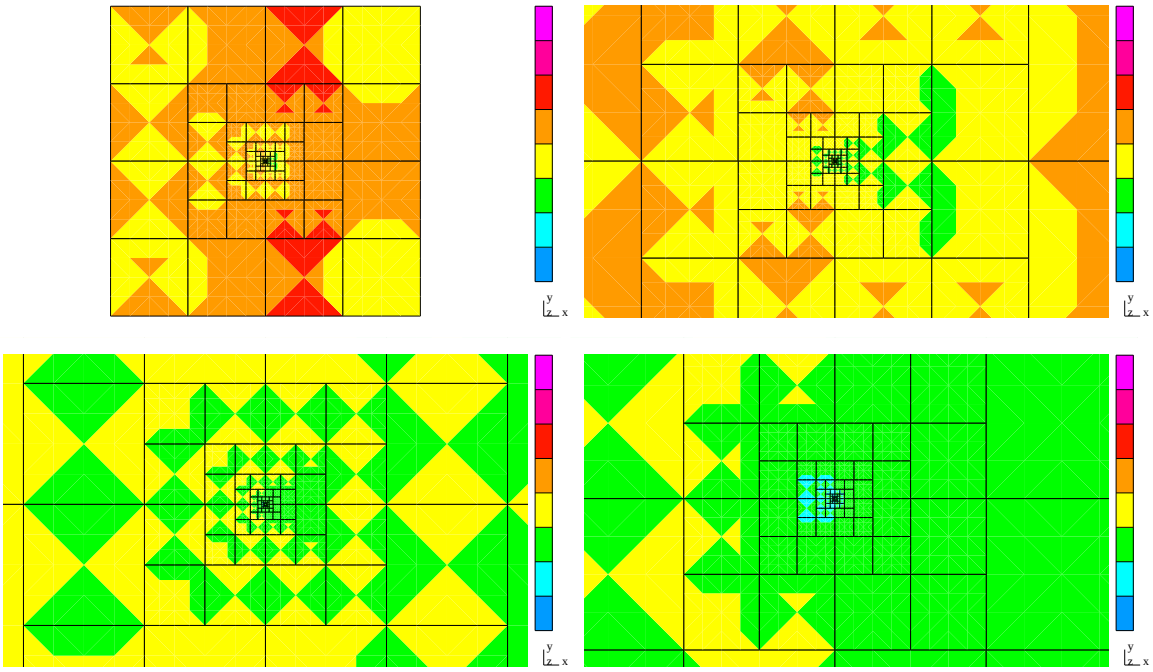


Figure 20.4: Diffraction of a plane wave on a screen. Final hp mesh, zooming on the diffracting edge (with factor 10)

- the $H(\mathrm{curl})$- projection (20.1.1),
- the $H(\mathrm{curl})$- stabilized projection (20.1.2),
- the actual projection defining the projection-based interpolation for $H(\mathrm{curl})$-conforming elements, (20.1.3).

In all three cases, the stabilized variational formulation has been used to solve the global problem. I have to admit that I have been surprised to see practically no difference in the influence of the choice of the element projections on the performance of the hp algorithm. The first version of the algorithm "died" in a routine performing the element projections due to the (numerical) singularity of the matrix. Replacing the $H(\mathrm{curl})$-projection with the stabilized $H(\mathrm{curl})$-projection eliminated that problem but the code stopped only a couple of meshes later in the global solve of the fine grid problem due to a small pivot. Replacing the stabilized $H(\mathrm{curl})$-projection with formulation (20.1.3) had no effect on either the solution of local problems or on the overall performance of the algorithm.

For comparison, Fig. 20.8 presents also the convergence history for the h-refinement strategy with cubic elements (see Section 14.4 for the discussion of the algorithm). Both h and hp algorithms display a strong deterioration of the convergence rates once the minimum element size drops below a certain level. The hp method seems to be more sensitive - the deterioration of the convergence can

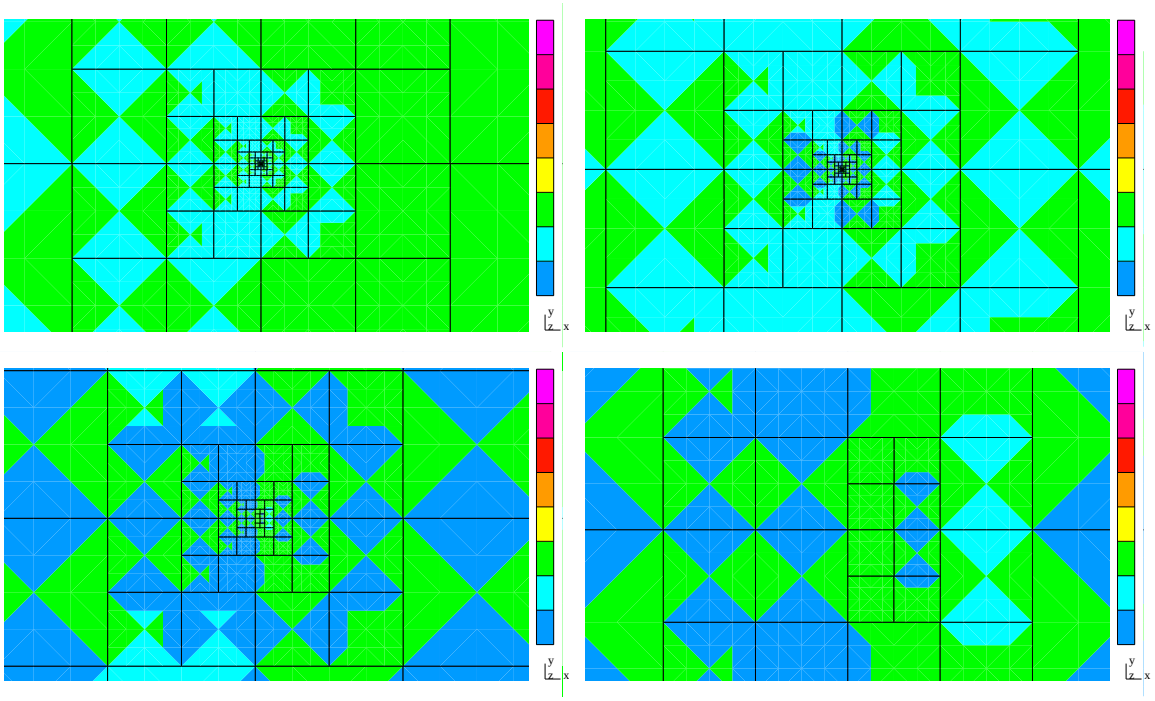


Figure 20.5: Diffraction of a plane wave on a screen. Final hp mesh, zooming on the diffracting edge (with factor 10)

be observed earlier. We attribute this to the round-off error effects. For two-dimensional problems, the "mass term" in the contribution to the element stiffness matrix scales as $O(1)$, but the curl-curl term scales as h^{-2}, in terms of element size h. With 15 significant digits, the mass term becomes "invisible" for h smaller that 10^{-7}.

Solutions to elliptic problems are singular only in terms of derivatives - the solution itself is finite. Singularity of the solution to the Maxwell problem is more severe - the solution itself blows up to infinity at the tip of the edge. We observe a need for more h-refinements which suggests that resolving the Maxwell singularities may call for more than a double precision. Practically identical results are obtained with triangular elements. Figures 20.9, 20.10 present eight consecutive zooms on the diffracting edge in the final, optimal mesh. The corresponding solution, this time the real part of the first component E_1, is displayed in Figures 20.11,20.12. Again, only at the level of the smallest elements, one can see the typical discontinuity artifacts related to the edge element discretization.

The corresponding convergence history is presented in Fig. 20.13. Use of triangular elements requires use of triangles in the geometrical modeling and, consequently, a preparation of a separate input file (*files/inputs/input_diffrt*).

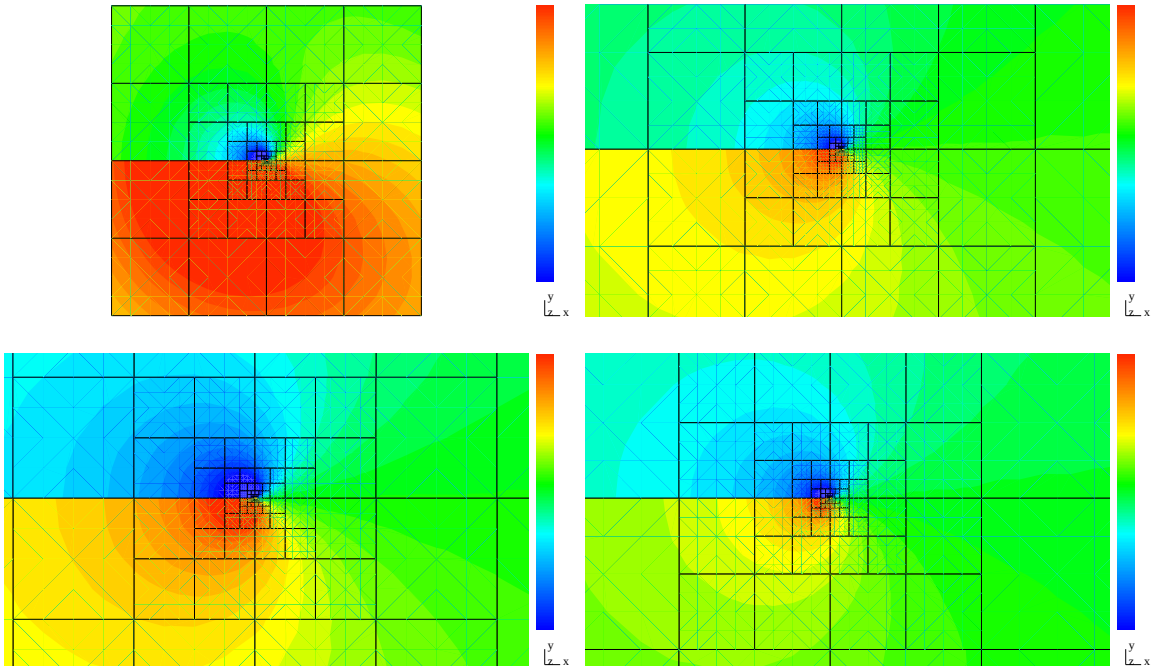


Figure 20.6: Diffraction of a plane wave on a screen. Imaginary part of E_1 component, zooming on the diffracting edge (with amplification factor 10)

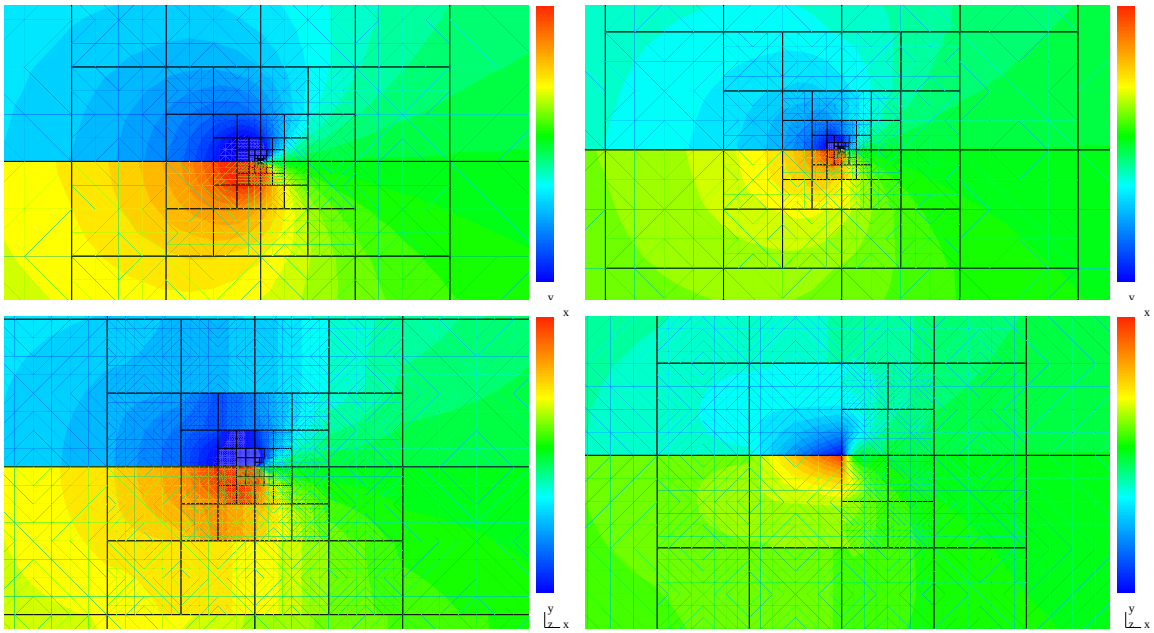


Figure 20.7: Diffraction of a plane wave on a screen. Imaginary part of E_1 component, zooming on the diffracting edge (cont.)

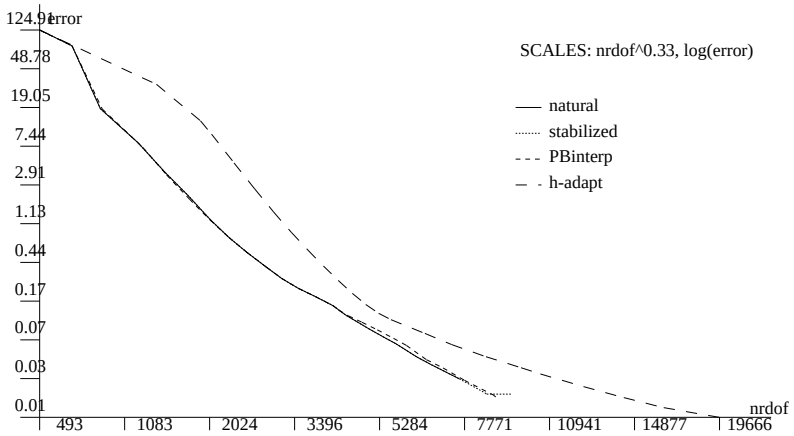


Figure 20.8: Diffraction of a plane wave. Convergence history

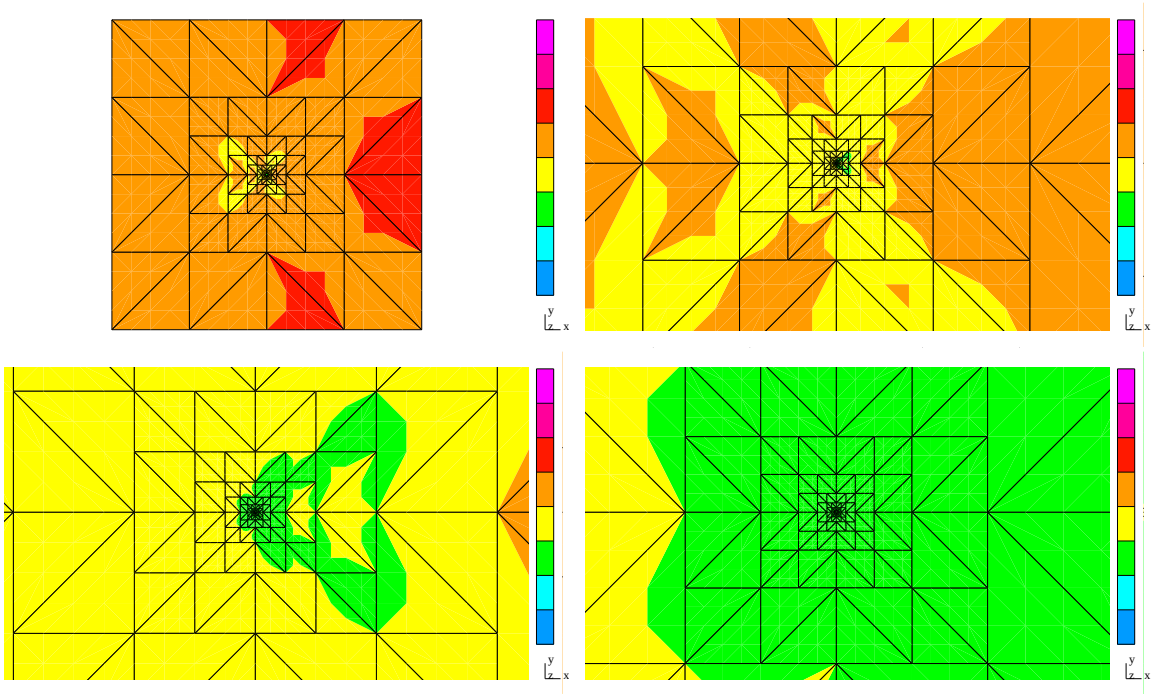


Figure 20.9: Diffraction of a plane wave on a screen. Final hp triangular mesh, zooming on the diffracting edge (with factor 10)

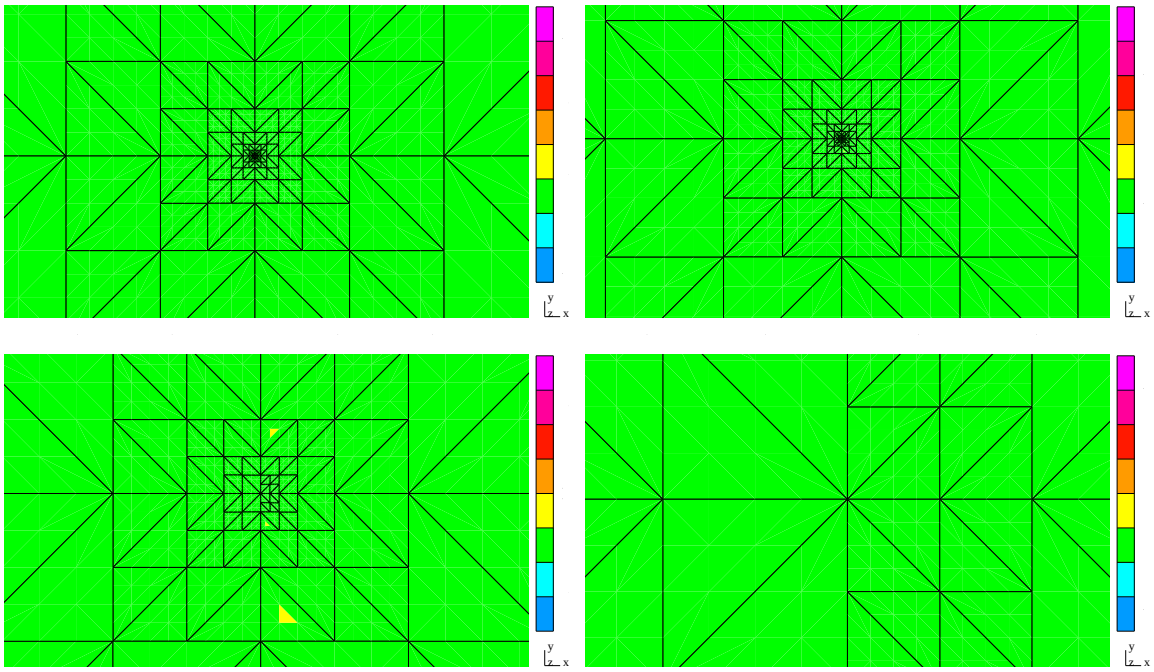


Figure 20.10: Diffraction of a plane wave on a screen. Final hp traingular mesh, zooming on the diffracting edge (with factor 10)

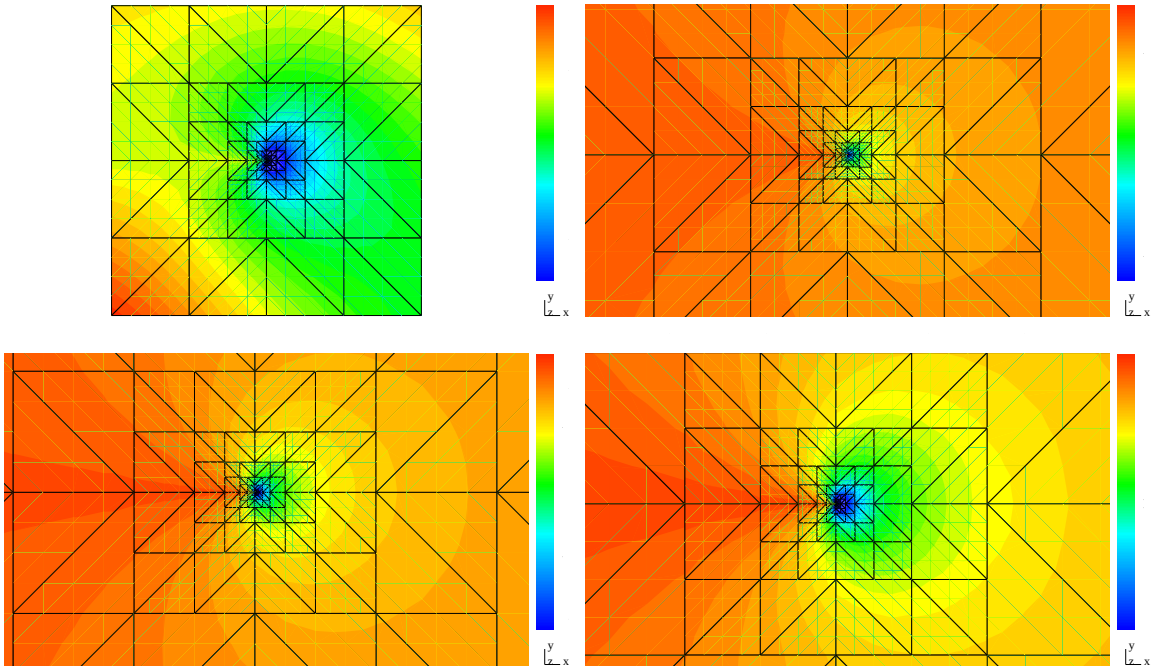


Figure 20.11: Diffraction of a plane wave on a screen. Real part of E_1 component, zooming on the diffracting edge (with amplification factor 10)

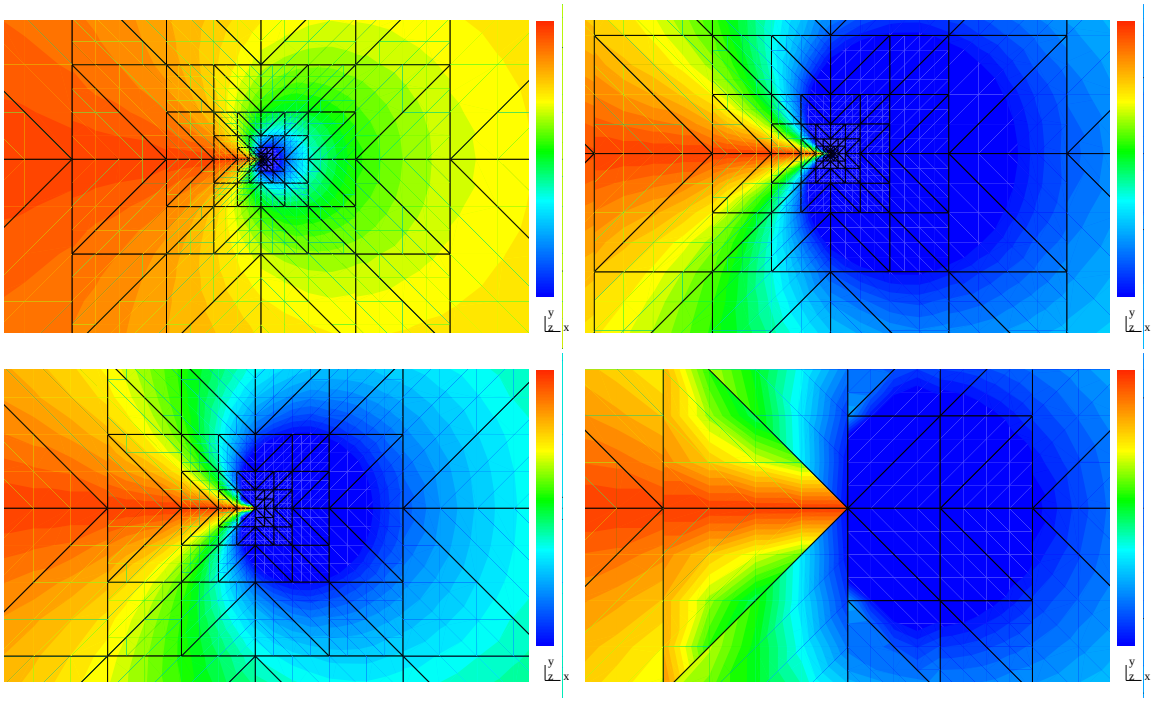


Figure 20.12: Diffraction of a plane wave on a screen. Real part of E_1 component, zooming on the diffracting edge (cont.)

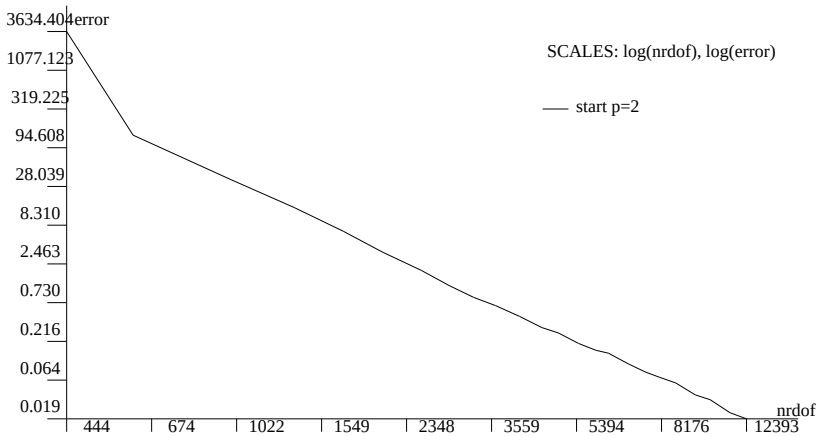


Figure 20.13: Diffraction of a plane wave. Convergence history for a mesh of triangular elements

Chapter 21

Exterior Maxwell Boundary-Value Problems

In this chapter we focus on a class of radiation/scattering problems for time-harmonic Maxwell equations, and develop further the idea of Infinite Element discretization discussed in Chapter 16. The presented work is motivated with the calculation of Radar Cross Sections (RCS) of objects with sharp edges and corners.

Let Ω^{int} be a bounded domain occupied by the scatterer, with Γ denoting its boundary, and let $\Omega = I\!R^n - \overline{\Omega^{int}}$ denote the exterior domain, with $n = 2, 3$. We truncate the exterior domain with a sphere (circle) $S_a = \{|\boldsymbol{x}| = a\}$ surrounding the scatterer, and split the domain Ω into a *near-field domain* $\Omega^a = \{\boldsymbol{x} \in \Omega : |\boldsymbol{x}| < a\}$, and a *far-field domain* $\Omega_a = \{|\boldsymbol{x}| > a\}$. Assuming for simplicity a *Perfect Conductor* (PEC) scatterer, we formulate the problem as follows.

Find electric field $\boldsymbol{E}$ that satisfies :

- Reduced wave equation in both near-field and far-field domains,

$$\boldsymbol{\nabla} \times (\boldsymbol{\nabla} \times \boldsymbol{E}) - k^2 \boldsymbol{E} = \boldsymbol{0}, \quad \boldsymbol{x} \in \Omega^a, \Omega_a \,, \tag{21.0.1}$$

- PEC boundary condition on boundary Γ,

$$\boldsymbol{n} \times \boldsymbol{E} = -\boldsymbol{n} \times \boldsymbol{E}^{inc}, \quad \boldsymbol{x} \in \Gamma \,, \tag{21.0.2}$$

- Interface boundary conditions on the truncating sphere,

$$\boldsymbol{n} \times [\boldsymbol{E}] = \boldsymbol{0}, \quad \boldsymbol{n} \times [\boldsymbol{\nabla} \times \boldsymbol{E}] = \boldsymbol{0}, \quad |\boldsymbol{x}| = a \,, \tag{21.0.3}$$

- Silver-Müller radiation condition at infinity,

$$\boldsymbol{e}_r \times (\boldsymbol{\nabla} \times \boldsymbol{E}) - ik\boldsymbol{E}_t \in L^2(\Omega^a) \,. \tag{21.0.4}$$

Here $\boldsymbol{n}$ denotes the normal to the boundary or interface, $k = \omega\sqrt{\epsilon_0\mu_0}$ is the free-space wave number, with ω denoting the angular frequency, and ϵ_0, μ_0 being the free space permittivity and permeability, $[\,]$ denotes the jump across the interface, $\boldsymbol{e}_r$ is radial unit vector corresponding to a spherical (polar) system of coordinates with a center inside of the scatterer, $\boldsymbol{E}^{inc}$ denotes an incident electric field, and $\boldsymbol{E}_t = -\boldsymbol{e}_r \times (\boldsymbol{e}_r \times \boldsymbol{E})$ is the tangential component of $\boldsymbol{E}$.

Infinite Elements. The idea of coupled Finite Element (FE)/Infinite Element (IE) approximations for exterior wave propagation problems dates back to the pioneering contributions of Bettess, and Bettess and Zienkiewicz, see [23] and the literature cited therein.

The works of Astley et al. [14], Cremers *et al.* [34], Givoli [72] and many others recognized the spectral character of the approximation and pointed to the necessity of multipole expansions. Burnett [26] revolutionized the approach from the practical point of view, by introducing a new, *symmetric* unconjugated formulation, and using prolate and oblate spheroidal elements.

Contrary to the concept of Perfectly Matched Layer [22], and other techniques based on Absorbing Boundary Conditions (ABC's), the conjugated element of Astley et al. [14], aims at obtaining the solution in the whole unbounded domain.

A conjugated Petrov-Galerkin infinite element for Maxwell equations was proposed in [56], and studied in [28, 29]. The idea of the construction was based on the assumption that the IE *test functions* come from a space belonging to an exact sequence involving standard grad-curl-div operators, discussed in the previous chapters. In simple terms, the test functions for the Maxwell Infinite Element include gradients of scalar potentials. In this way, any FE/IE solution to the Maxwell equations satisfies automatically the weak form of the continuity equation. The Petrov-Galerkin infinite element was applied to solve difficult three-dimensional scattering problems in [106].

In this chapter, we generalize the concept of a conjugated Bubnov-Galerkin infinite element for Helmholtz equation presented in Chapter 16. As for the Helmholtz equation, the main idea consists of interpreting the integral over the exterior domain in a Cauchy Principal Value (CPV) sense, and building the far-field pattern into the ansatz for the approximate solution. The resulting discretization allows for the use of the *same* trial and test functions but the discrete solution no longer satisfies the usual weak form of the continuity equation.

21.1 Variational formulation

We follow the standard procedure for the near-field domain. Taking a test function such that $\boldsymbol{n}\times\boldsymbol{F} = \boldsymbol{0}$ on Γ, we multiply the reduced wave equation (21.0.1) with complex conjugate $\bar{\boldsymbol{F}}$, and integrate by parts to obtain,

$$\int_{\Omega^a} (\boldsymbol{\nabla}\times\boldsymbol{E})(\boldsymbol{\nabla}\times\bar{\boldsymbol{F}}) - k^2\boldsymbol{E}\bar{\boldsymbol{F}}\,d\boldsymbol{x} = -\int_{S_a} \boldsymbol{e}_r\times(\boldsymbol{\nabla}\times\boldsymbol{E})\bar{\boldsymbol{F}}\,dS, \quad \forall\boldsymbol{F}, \boldsymbol{n}\times\boldsymbol{F} = 0 \text{ on } \Gamma\,. \quad (21.1.5)$$

Integration over the far-field domain will be interpreted in the Cauchy Principal Value (CPV) sense. We introduce a second truncating sphere $S_R = \{|\boldsymbol{x}| = R\}, R > a$ and consider the truncated far-field domain,

$$\Omega_a^R = \{\boldsymbol{x} \,:\, a < |\boldsymbol{x}| < R\}\,.$$

Performing the same operations for Ω_a^R as for the near-field domain, we obtain,

$$\int_{\Omega_a^R} (\boldsymbol{\nabla}\times\boldsymbol{E})(\boldsymbol{\nabla}\times\bar{\boldsymbol{F}}) - k^2\boldsymbol{E}\bar{\boldsymbol{F}}\,d\boldsymbol{x} + \int_{S_R} \boldsymbol{e}_r\times(\boldsymbol{\nabla}\times\boldsymbol{E})\bar{\boldsymbol{F}}\,dS = \int_{S_a} \boldsymbol{e}_r\times(\boldsymbol{\nabla}\times\boldsymbol{E})\bar{\boldsymbol{F}}\,dS, \quad \forall \boldsymbol{F}\,. \tag{21.1.6}$$

Using the Silver-Müller radiation condition (21.0.4), we replace $\boldsymbol{e}_r\times(\boldsymbol{\nabla}\times\boldsymbol{E})$ in the integral over S_R with $ik\boldsymbol{E}_t$ plus an unknown contribution which, due to the assumption on L^2-integrability, will vanish in the limit when $R\to\infty$,

$$\begin{aligned}&\int_{\Omega_a^R} (\boldsymbol{\nabla}\times\boldsymbol{E})(\boldsymbol{\nabla}\times\bar{\boldsymbol{F}}) - k^2\boldsymbol{E}\bar{\boldsymbol{F}}\,d\boldsymbol{x} + ik\int_{S_R}\boldsymbol{E}_t\bar{\boldsymbol{F}}\,dS + (\text{ a term that vanishes at } R\to\infty)\\ &= \int_{S_a} \boldsymbol{e}_r\times(\boldsymbol{\nabla}\times\boldsymbol{E})\bar{\boldsymbol{F}}\,dS, \quad \forall \boldsymbol{F}\,.\end{aligned} \tag{21.1.7}$$

Finally, we sum up contributions (21.1.5) and (21.1.7). Assuming that both solution $\boldsymbol{E}$ and test function $\boldsymbol{F}$ satisfy the interface conditions (21.0.3), we obtain our final variational formulation.

$$\left\{\begin{array}{l} \boldsymbol{n}\times\boldsymbol{E} = -\boldsymbol{n}\times\boldsymbol{E}^{inc},\\ \displaystyle\int_{\Omega^a} (\boldsymbol{\nabla}\times\boldsymbol{E})(\boldsymbol{\nabla}\times\bar{\boldsymbol{F}}) - k^2\boldsymbol{E}\bar{\boldsymbol{F}}\,d\boldsymbol{x}\\ \displaystyle + \lim_{R\to\infty}\left(\int_{\Omega_a^R} (\boldsymbol{\nabla}\times\boldsymbol{E})(\boldsymbol{\nabla}\times\boldsymbol{F}) - k^2\boldsymbol{E}\bar{\boldsymbol{F}}\,d\boldsymbol{x} + ik\int_{S_R}\boldsymbol{E}_t\bar{\boldsymbol{F}}\,dS\right) = 0, \quad \forall \boldsymbol{F}, \boldsymbol{n}\times\boldsymbol{F} = \boldsymbol{0} \text{ on } \Gamma\,. \end{array}\right. \tag{21.1.8}$$

Both solution $\boldsymbol{E}$ and test function $\boldsymbol{F}$ are assumed to "live" in $\boldsymbol{H}_{loc}(\Omega,\mathrm{curl})$ with extra assumptions to guarantee the existence of the limit to be discussed next.

The presented concept of the variational formulation based on interpreting the integral over the exterior domain in the CPV sense, is valid in any space dimension. We shall use it now as a basis for deriving the corresponding IE discretization. When it comes to Maxwell problems and the curl operator, discussion of the 3D case is more straightforward than its 2D counterpart, and we shall present it first, following with the 2D version next. Three-dimensional numerical experiments will be presented in the second volume of this monograph.

21.2 Infinite Element discretization in 3D

As for the Helmholtz equation, the main idea of the Infinite Element discretization for the Maxwell equation, is to incorporate into the IE ansatz the asymptotic form of the solution. Using separation

of variables, we arrive at a representation of the solution in terms of Hankel functions, which leads to the following formula for the far-field pattern, compare [33],

$$\boldsymbol{E}_t = \frac{e^{-ikr}}{r^\alpha}\boldsymbol{E}_t^{FFP}(\hat{\boldsymbol{x}}) + \boldsymbol{O}(r^{-(\alpha+1)}), \quad \boldsymbol{E}_r = O(r^{-(\alpha+1)}) .$$

Here $\alpha = 1$ for 3D problems, and $\alpha = 1/2$ in two space dimensions. Function $\boldsymbol{E}_t^{FFP}$ depends only upon the direction represented by point $\hat{\boldsymbol{x}}$ on the unit sphere centered at the origin, is called the *far-field pattern*. Notice that, asymptotically in r, the radial component E_r is negligible when compared with the tangential component $\boldsymbol{E}_t$. This is related to the fact that, for the spherical waves, the magnetic field vector is only asymptotically orthogonal to the electric field vector (for plane EM waves, $\boldsymbol{E}$ and $\boldsymbol{H}$ must be orthogonal).

Infinite element coordinates

Recall that for the Helmholtz equation, we had formulated the discretization in terms of radial coordinate r first, and only later switched to the "inverted radial coordinate" $x = r^{-1}$. In the case of the Maxwell problem, it is advantegeous to start using the inverted coordinate from the very beginning.

The curvilinear system of coordinates used to construct the infinite element discretization combines thus a parametrization of the truncating sphere with the inverted radial coordinate,

$$\boldsymbol{x}(\xi_\alpha, x) = x^{-1}\boldsymbol{x}_a(\xi_\alpha), \quad \alpha = 1,2, \quad |\boldsymbol{x}_a(\xi_\alpha)| = a, \quad 0 < x < 1 .$$

Here $\boldsymbol{x}_a(\xi_\alpha)$ is a parametrization of the sphere of radius a, centered at the origin of a Cartesian system of coordinates x_i.[1], see Fig. 21.1. The basis vectors are,

$$\boldsymbol{a}_\alpha = x^{-1}\frac{\partial \boldsymbol{x}_a}{\partial \xi_\alpha} = x^{-1}|\frac{\partial \boldsymbol{x}_a}{\partial \xi_\alpha}|\boldsymbol{e}_\alpha, \qquad \boldsymbol{a}_x = -x^{-2}\boldsymbol{x}_a = x^{-2}a\boldsymbol{e}_x .$$

We assume that the parameters α have been denumerated in such a way that $(\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_x)$ is a right triple. The cobasis vectors are given by,

$$\boldsymbol{a}^1 = x\mathrm{jac}_a^{-1}(\frac{\partial \boldsymbol{x}_a}{\partial \xi_2} \times \boldsymbol{e}_x), \quad \boldsymbol{a}^2 = -x\mathrm{jac}_a^{-1}(\frac{\partial \boldsymbol{x}_a}{\partial \xi_1} \times \boldsymbol{e}_x), \quad \boldsymbol{a}^x = x^2 a^{-1}\boldsymbol{e}_x ,$$

where,

$$\mathrm{jac}_a = (\frac{\partial \boldsymbol{x}_a}{\partial \xi_1} \times \frac{\partial \boldsymbol{x}_a}{\partial \xi_2}) \cdot \boldsymbol{e}_x .$$

Denoting by $\hat{\boldsymbol{a}}_\alpha, \hat{\boldsymbol{a}}^\alpha$ basis and cobasis vectors on sphere S_a,

$$\boldsymbol{a}_\alpha = x^{-1}\hat{\boldsymbol{a}}_\alpha, \quad \boldsymbol{a}^\alpha = x\hat{\boldsymbol{a}}^\alpha ,$$

[1] We shall use Greek letters for indices with range 1,2, and Roman letters for indices with range 1,2,3

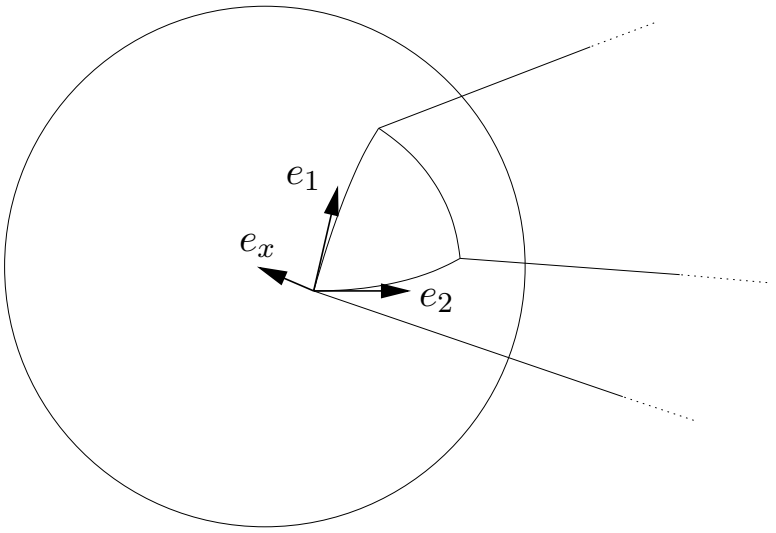


Figure 21.1: Curvilinear system of coordinates on the truncating sphere

we recall the standard formula for the gradient of a scalar-valued function u,

$$\boldsymbol{\nabla} u = x \frac{\partial u}{\partial \xi_\alpha} \hat{\boldsymbol{a}}^\alpha + x^2 a^{-1} \frac{\partial u}{\partial r} \boldsymbol{e}_x \; .$$

As usual, repeated indices indicate summation.

Consistently with the standard exact sequence property and the construction of parametric finite elements, compare Section 18.1.4, all vector-valued fields will be assumed in the form,

$$\boldsymbol{E} = x E_\alpha \hat{\boldsymbol{a}}^\alpha + x^2 a^{-1} E_x \boldsymbol{e}_x \; .$$

REMARK 2 Notice that, with components $E_\alpha, E_x = O(1)$, the choice of the system of coordinates guarantees the right asymptotic behavior in $r = x^{-1}$, dicussed earlier. ∎

Recalling the general transformation rule for the curl vector in three space dimensions [41],

$$(\mathrm{curl} \boldsymbol{E})_i = J^{-1} \frac{\partial x_i}{\partial \xi_n} (\hat{\mathrm{curl}} \hat{\boldsymbol{E}})_n \; ,$$

where J is the jacobian of transformation $x_i = x_i(\xi_n)$ (in our case $J = x^{-4} a \mathrm{jac}_a$), we can compute the curl of vector $\boldsymbol{E}$,

$$\boldsymbol{\nabla} \times \boldsymbol{E} = (x^{-4} a \mathrm{jac}_a)^{-1} \left\{ (\hat{\mathrm{curl}} \hat{\boldsymbol{E}})_\alpha x^{-1} \hat{\boldsymbol{a}}_\alpha + (\hat{\mathrm{curl}} \hat{\boldsymbol{E}})_3 x^{-2} a \boldsymbol{e}_x \right\} \; .$$

Here,

$$\hat{\mathrm{curl}} \hat{\boldsymbol{E}} = \left(\frac{\partial E_x}{\partial \xi_2} - \frac{\partial E_2}{\partial x}, \; -(\frac{\partial E_x}{\partial \xi_1} - \frac{\partial E_1}{\partial x}), \; \frac{\partial E_2}{\partial \xi_1} - \frac{\partial E_1}{\partial \xi_2} \right) \; .$$

REMARK 3 The curvilinear coordinates ξ_α discussed here are to be understood in two different ways. On the theoretical side, they provide a basis for the construction of the IE approximation; in this context, one can think about e.g. standard spherical coordinates. On the practical side, they can be directly interpreted as a parametrization corresponding to isoparametric finite elements used to approximate the truncating sphere. In such a case, they will correspond to a *local*, element-wise approximation of the sphere only. Formulas derived here can then be used directly for coding. ∎

Incorporating the far-field pattern. Consistently with the far-field pattern, we shall postulate the solution $\boldsymbol{E}$ in the form,

$$\boldsymbol{E} := e^{-ika(x^{-1}-1)}\boldsymbol{E}\,,$$

where the symbol $\boldsymbol{E}$ has been "overloaded". Recalling the elementary formula,

$$\boldsymbol{\nabla}\times(\phi\boldsymbol{E}) = \boldsymbol{\nabla}\phi\times\boldsymbol{E} + \phi\,\boldsymbol{\nabla}\times\boldsymbol{E}\,,$$

we obtain,

$$\boldsymbol{\nabla}\times\left(e^{-ika(x^{-1}-1)}\boldsymbol{E}\right) = e^{-ika(x^{-1}-1)}\left(+ik(\boldsymbol{e}_x\times\boldsymbol{E}) + \boldsymbol{\nabla}\times\boldsymbol{E}\right)\,. \tag{21.2.9}$$

Identical substitution for the (conjugated) test function $\bar{\boldsymbol{F}}$, leads to,

$$\boldsymbol{\nabla}\times\left(e^{+ika(x^{-1}-1)}\bar{\boldsymbol{F}}\right) = e^{+ika(x^{-1}-1)}\left(-ik(\boldsymbol{e}_x\times\bar{\boldsymbol{F}}) + \boldsymbol{\nabla}\times\bar{\boldsymbol{F}}\right)\,. \tag{21.2.10}$$

Substituting (21.2.9) and (21.2.10) into the limit term in (21.1.8), we obtain (with overloaded symbols $\boldsymbol{E}$ and $\boldsymbol{F}$)

$$\begin{aligned}\lim_{X\to 0}\ \Bigg(\int\int_X^1\ & \left\{k^2(\boldsymbol{e}_x\times\boldsymbol{E})\cdot(\boldsymbol{e}_x\times\bar{\boldsymbol{F}}) + (\boldsymbol{\nabla}\times\boldsymbol{E})\cdot(\boldsymbol{\nabla}\times\bar{\boldsymbol{F}})\right.\\ & \left.-ik\left[(\boldsymbol{\nabla}\times\boldsymbol{E})\cdot(\boldsymbol{e}_x\times\bar{\boldsymbol{F}}) - (\boldsymbol{e}_x\times\boldsymbol{E})\cdot(\boldsymbol{\nabla}\times\bar{\boldsymbol{F}})\right] - k^2\boldsymbol{E}\cdot\bar{\boldsymbol{F}}\right\}\,x^{-4}a\mathrm{jac}_a\,dxd\boldsymbol{\xi}\\ & +ik\int\boldsymbol{E}_t(\cdot,X)\bar{\boldsymbol{F}}_t(\cdot,X)X^{-2}\mathrm{jac}_a\,d\boldsymbol{\xi}\Bigg)\,.\end{aligned}$$

Here, parameter X in the limit corresponds to R^{-1} compared with the limit in (21.1.8). But,

$$k^2(\boldsymbol{e}_x\times\boldsymbol{E})\cdot(\boldsymbol{e}_x\times\bar{\boldsymbol{F}}) - k^2\boldsymbol{E}\cdot\bar{\boldsymbol{F}} = k^2x^2a^{-2}E_x\bar{F}_x\,,$$

so the term reduces to,

$$\begin{aligned}\lim_{X\to 0}\Bigg(\int\int_X^1\ & \left\{(\boldsymbol{\nabla}\times\boldsymbol{E})\cdot(\boldsymbol{\nabla}\times\bar{\boldsymbol{F}}) - ik\left[(\boldsymbol{\nabla}\times\boldsymbol{E})\cdot(\boldsymbol{e}_x\times\bar{\boldsymbol{F}}) - (\boldsymbol{e}_x\times\boldsymbol{E})\cdot(\boldsymbol{\nabla}\times\bar{\boldsymbol{F}})\right]\right.\\ & \left.-k^2x^2a^{-2}E_x\bar{F}_x\right\}\,x^{-4}a\mathrm{jac}_a\,dxd\boldsymbol{\xi} + ik\int\boldsymbol{E}_t(\cdot,X)\bar{\boldsymbol{F}}_t(\cdot,X)X^{-2}\mathrm{jac}_a\,d\boldsymbol{\xi}\Bigg)\,.\end{aligned}$$

Substitution of the far-field pattern into the ansatz for the solution and test function yields the limit finite for $E_\alpha, E_x = O(1)$. We obtain,

$$\int\int_0^1 \quad \left\{(\boldsymbol{\nabla}\times\boldsymbol{E})\cdot(\boldsymbol{\nabla}\times\bar{\boldsymbol{F}}) - ik\left[(\boldsymbol{\nabla}\times\boldsymbol{E})\cdot(\boldsymbol{e}_x\times\bar{\boldsymbol{F}}) - (\boldsymbol{e}_x\times\boldsymbol{E})\cdot(\boldsymbol{\nabla}\times\bar{\boldsymbol{F}})\right]\right.$$
$$\left. -k^2x^2a^{-2}E_x\bar{F}_x\right\}\; x^{-4}a\mathrm{jac}_a\, dxd\boldsymbol{\xi} + ik\int (E_\alpha(\cdot,0)\boldsymbol{a}^\alpha)(\bar{F}_\alpha(\cdot,0)\boldsymbol{a}^\alpha)\mathrm{jac}_a\, d\boldsymbol{\xi}\;. \tag{21.2.11}$$

We record the final formulas necessary for the calculation of the stiffness matrix.

$$\begin{aligned}\boldsymbol{\nabla}\times\boldsymbol{E} &= \mathrm{jac}_a^{-1}\left\{+\frac{x^3}{a}\left(\frac{\partial E_x}{\partial\xi_2}-\frac{\partial E_2}{\partial x}\right)\hat{\boldsymbol{a}}_1 - \frac{x^3}{a}\left(\frac{\partial E_x}{\partial\xi_1}-\frac{\partial E_1}{\partial x}\right)\hat{\boldsymbol{a}}_2 + x\left(\frac{\partial E_2}{\partial\xi_1}-\frac{\partial E_1}{\partial\xi_2}\right)\boldsymbol{e}_x\right\}\\ \boldsymbol{e}_x\times\boldsymbol{E} &= x\mathrm{jac}_a^{-1}\left(-E_2\hat{\boldsymbol{a}}_1 + E_1\hat{\boldsymbol{a}}_2\right)\;.\end{aligned}$$

Discretization. Solution components E_α, E_x and test function components F_α, F_x can now be discretized using standard Nédélec hexahedral (or prismatic) elements of the first type [92]. For the hexahedral element of order (p_α, p_x), we have,

$$E_1\in\mathcal{P}^{p_1-1}\otimes\mathcal{P}^{p_2}\otimes\mathcal{P}^{p_x},\quad E_2\in\mathcal{P}^{p_1}\otimes\mathcal{P}^{p_2-1}\otimes\mathcal{P}^{p_x},\quad E_x\in\mathcal{P}^{p_1}\otimes\mathcal{P}^{p_2}\otimes\mathcal{P}^{p_x-1}\;. \tag{21.2.12}$$

Shape functions depending upon ξ_α have to match shape functions for the standard quadrilateral element, see [3]. The leading term in x, similarly to the Helmholtz case (comp. Chapter 16), corresponds to term,

$$\int_0^1 x^2\frac{\partial E_\alpha}{\partial x}\frac{\partial\bar{F}_\beta}{\partial x}\,dx\;,$$

and suggests selecting for the shape functions in x integrals of Jacobi polynomials $P_j^{2,0}$,

$$\psi_j(x) = \begin{cases} 1 & j=0\\ \displaystyle\int_x^1 P_{j-1}^{(0,2)}(2t-1)\,dt & j\geq 1\;.\end{cases}$$

21.3 Infinite Element discretization in 2D

The reasoning in two dimensions is very similar.

2D IE coordinates

We use polar-like coordinates,

$$\boldsymbol{x}(x,\xi) = x^{-1}\boldsymbol{x}_a(\xi),\quad |\boldsymbol{x}_a(\xi)| = a,\quad x\in(0,1)\;,$$

where $\boldsymbol{x}_a(\xi)$ is a clockwise parametrization of the truncating circle. Basis and cobasis vectors are defined as,

$$\boldsymbol{a}_\xi = x^{-1}\frac{d\boldsymbol{x}_a}{d\xi} = x^{-1}\underbrace{|\frac{d\boldsymbol{x}_a}{d\xi}|}_{\mathrm{jac}_a}\boldsymbol{e}_\xi, \quad \boldsymbol{a}_x = -x^{-2}\boldsymbol{x}_a(\xi) = x^{-2}a\boldsymbol{e}_x, \quad \boldsymbol{a}^\xi = x\mathrm{jac}_a^{-1}\boldsymbol{e}_\xi, \quad \boldsymbol{a}^x = x^2a^{-1}\boldsymbol{e}_r,.$$

The formula for a gradient of function $u(x,\xi)$ is,

$$\boldsymbol{\nabla}u = \frac{\partial u}{\partial x}x^2a^{-1}\boldsymbol{e}_x + \frac{\partial u}{\partial \xi}x\mathrm{jac}_a^{-1}\boldsymbol{e}_\xi\ ,$$

with a corresponding representation for vector-valued fields,

$$\boldsymbol{E} = E_x x^2 a^{-1}\boldsymbol{e}_x + E_\xi x\,\mathrm{jac}_a^{-1}\boldsymbol{e}_\xi\ .$$

The curl is evaluated using the formula,

$$\mathrm{curl}\boldsymbol{E} = (x^{-3}a\mathrm{jac}_a)^{-1}\underbrace{\left(\frac{\partial E_\xi}{\partial x} - \frac{\partial E_x}{\partial \xi}\right)}_{\hat{\mathrm{curl}}\boldsymbol{E}}\ .$$

Incorporating the far-fi eld pattern

The anzatz incorporating the far field pattern is now slightly different,

$$\boldsymbol{E} := x^{-\frac{1}{2}}e^{-ika(x^{-1}-1)}\boldsymbol{E}\ .$$

Upon the substitution, and cancellation of Lebesgue non-integrable terms, we obtain an expression analogous to (21.2.11),

$$\int\int_0^1 \Big\{(a\mathrm{jac}_a)^{-1}\left[x^2\hat{\mathrm{curl}}\boldsymbol{E}\,\hat{\mathrm{curl}}\bar{\boldsymbol{F}} - (\frac{1}{2}x+ika)\hat{\mathrm{curl}}\boldsymbol{E}\bar{F}_\xi - (\frac{1}{2}x-ika)E_\xi\hat{\mathrm{curl}}\bar{\boldsymbol{F}} + \frac{1}{4}E_\xi\bar{F}_\xi\right]$$
$$-(ka)^2a^{-3}\mathrm{jac}_aE_x\bar{F}_x\Big\}\ dxd\xi + ika\int a^{-1}\mathrm{jac}_a^{-1}E_\xi(\cdot,0)\bar{F}_\xi(\cdot,0)\,d\xi\ .$$

Leading terms in x are identical as in 3D and suggest the use of the same radial shape functions.

21.4 Stability

Substituting in (21.2.11) $\boldsymbol{F} = \boldsymbol{\nabla}v$, where v is a scalar-valued test function with support in the far-field domain, we learn that any solution to variational formulation (21.1.8) satisfies automatically a variational compatibility condition, that implies the corresponding (second order) differential equation and a radiation condition at infinity. These compatibility conditions *do not* coincide with the

usual continuity equation obtained directly from the weak form of the Maxwell equations. Equivalently, we could substitute in original variational formulation,

$$\boldsymbol{F} = e^{-ika(x^{-1}-1)}\boldsymbol{\nabla} v = \boldsymbol{\nabla}(e^{-ika(x^{-1}-1)}v) + ike^{-ika(x^{-1}-1)}x^{-2}v\boldsymbol{e}_x \,. \tag{21.4.13}$$

Only substitution of a gradient for the test function $\boldsymbol{F}$ results in the weak form of the continuity equation. Notice that the second term in (21.4.13) modifies only the radial component of the gradient and, therefore it can also be done in a situation when the infinite elements are coupled with finite elements. In other words, we can extend (21.4.13) by a regular gradient into the finite element domain and substitute the resulting test function into the variational formulation extending over the whole domain.

The Nédélec element (21.2.12) belongs to the standard family of polynomials satisfying the exact sequence property,

$$W_p \stackrel{\text{grad}}{\longrightarrow} \mathbf{Q}_p \stackrel{\text{curl}}{\longrightarrow} \mathbf{V}_p \stackrel{\text{div}}{\longrightarrow} Y_p \,,$$

where,

$$\begin{aligned}
W^p &= \mathcal{P}^{p_1} \otimes \mathcal{P}^{p_2} \otimes \mathcal{P}^{p_x} \\
\mathbf{Q}_p &= (\mathcal{P}^{p_1-1} \otimes \mathcal{P}^{p_2} \otimes \mathcal{P}^{p_x}) \times (\mathcal{P}^{p_1} \otimes \mathcal{P}^{p_2-1} \otimes \mathcal{P}^{p_x}) \times (\mathcal{P}^{p_1} \otimes \mathcal{P}^{p_2} \otimes \mathcal{P}^{p_x-1}) \\
\mathbf{V}_p &= (\mathcal{P}^{p_1} \otimes \mathcal{P}^{p_2-1} \otimes \mathcal{P}^{p_x-1}) \times (\mathcal{P}^{p_1-1} \otimes \mathcal{P}^{p_2} \otimes \mathcal{P}^{p_x-1}) \times (\mathcal{P}^{p_1-1} \otimes \mathcal{P}^{p_2-1} \otimes \mathcal{P}^{p_x}) \\
Y_p &= \mathcal{P}^{p_1-1} \otimes \mathcal{P}^{p_2-1} \otimes \mathcal{P}^{p_x-1}
\end{aligned}$$

Denoting by Au the multiplication by the far-field pattern,

$$Au = e^{-ika(x^{-1}-1)}u \,,$$

we see that the infinite element shape functions incorporating the factor, belong to a modified exact sequence,

$$W_p^{exp} \stackrel{A\text{grad}A^{-1}}{\longrightarrow} \mathbf{Q}_p^{exp} \stackrel{A\text{curl}A^{-1}}{\longrightarrow} \mathbf{V}_p^{exp} \stackrel{A\text{div}A^{-1}}{\longrightarrow} Y_p^{exp} \,,$$

where spaces $W_p^{exp}, \mathbf{Q}_p^{exp}, \mathbf{V}_p^{exp}, Y_p^{exp}$, have been obtained by multiplying the polynomials from the original spaces with the exponential factor. This situation is similar to the Bloch approximations studied by Dobson and Pasciak [64].

The 2D case is fully analogous.

As usual we can build a stabilized variational formulation by introducing a Lagrange multiplier p,

$$\left\{ \begin{array}{lll} a(\boldsymbol{E},\boldsymbol{F}) + \overline{c(\boldsymbol{F},p)} & = l(\boldsymbol{F}), & \forall \boldsymbol{F} \\ c(\boldsymbol{E},v) & = l(\boldsymbol{\nabla} v), & \forall v \end{array} \right.$$

Here $a(\boldsymbol{E},\boldsymbol{F})$ and $l(\boldsymbol{F})$ denote the sesquilinear and antilinear forms corresponding to the variational formulation (21.1.8), and the sesquilinear form $c(\boldsymbol{E},v)$ has been obtained by using the substitution (21.4.13),

$$c(\boldsymbol{E},v) := k^{-1}a(\boldsymbol{E},\boldsymbol{F}) \,.$$

Incorporating one power of k into the Langrange multiplier, improves the stability properties for $k \to 0$ but it does yield a formulation uniformly stable in k as for bounded domains and non-conductive materials. The situation resembles the conductive case $\sigma > 0$ for the bounded domains. This is, in a sense, consistent with the concept of "radiation damping".

REMARK 4 Notice that the space for the Lagrange multiplier *does not* coincide with the corresponding space for the IE discretization for the Helmholtz equation [59]. This is related to the fact that the Lagrange multiplier enters the stabilized formulation only through its gradient. The gradient decays at infinity but the Lagrange multiplier *does not* and, therefore, the corresponding element cannot be used to approximate the Helmholtz problem. For the standard elements, space W is used to solve acoustics in terms of pressure, space $\boldsymbol{Q}$ is for Maxwell equations, and space $\boldsymbol{V}$ of $H(\mathrm{div})$-conforming elements is used when discretizing the acoustics in terms of velocity. This nice analogy for the discussed infinite elements is lost. ∎

21.5 Implementation

All routines implementing the Maxwell infinite element can be found in directory *infel*. A number of routines for the Helmohltz infinite element, has been upgraded for the Maxwell case, using the Maxwell preprocessing flag. Those include,

- module *module/infinite_elements* storing the infinite elements connectivities and d.o.f.,
- routine *infel/initiate_infinite_elements* initiating the temporary data structure, every time the coupled FE/IE problem is to be solved,
- routine *solve1(IE)* - the driver for the frontal solver. Notice that the name of the routine - *solve1* is identical with the original driver in directory *solver1*, but the name of the file containing it is different (letters *IE* added).
- routine *solin2(IE)* - interface routine with the frontal solver, "feeding" the frontal solver with IE stiffness matrices,
- routine *displayIE* to display a finite portion of the infinite element mesh (corresponding to the range of the inverted radial coordinate $x > 1/2$), useful for studying the quality of the IE discretization,
- routine *celeminf* incorporating sign factors in the infinite element matrices,
- routine *RCS* evaluating the far field pattern (radar crossection),
- routine *lsvisids(IE)* displaying the FE/IE mesh.

The IE stiffness matrices (load vectors are equal zero) are calculated in a new routine *eleminf_EM*. Integration of the infinite element stiffness matrix is done using the standard Gauss-Legendre quadrature, with the number of integration points equal to $p+1$ in the tangential, and $N+1$ in the radial directions, where p and N denote the order of the infinite element in the tangential and the radial directions, respectively.

Automatic hp-adaptivity. We refer to the previous chapter for details on the algorithm executing the automatic hp-adaptivity. Starting with an initial *coarse mesh*, we refine the mesh globally in both h and p, and solve the problem on the *fine mesh*. The next optimal coarse mesh is obtained by maximizing the rate with which the interpolation error decreases, as the current coarse grid undergoes selective, local hp-refinements. One of the primary goals of the presented research was to determine whether the strategy can be used for the scattering problems. The mesh optimization is restricted to the near-field (truncated) domain only, i.e. the infinite elements are treated as an implicit implementation of ABC's of arbitrary order.

Choice of radial order N. The infinite elements in the initial mesh are assumed to be *isotropic*, i.e. order N in the radial direction is set to the corresponding order p of the edge on the truncating circle. We always begin with elements of second order.

During the hp-refinements, edges on the truncating circle get p- or h-refined. Every time, the edge is p-refined, its IE radial order is updated, i.e. N is increased to $N+1$. We also increase the IE order when the edge is h-refined. Therefore, in presence of h-refinements, we encounter infinite elements with radial order N *greater* than the FE order p. This reflects the philosophy that any improvement in the approximation properties on the truncating circle, should be accompanied with the corresponding improvement in the radial direction as well. The IE radial order has been restricted to $N \leq 9$.

Evaluation of the error. In all reported experiments, the error is computed in the $H(\mathrm{curl})$–norm, integrated *only* over the near-field domain. This is in line with treating the infinite element as an implicit implementation of an Absorbing Boundary Condition (ABC) only. Evaluation of the error over the whole exterior domain should be done in a weighted Sobolev norm. Since, at present, we cannot prove any convergence result for the IE discretization, we restrict ourselves to the near-field domain only and do not claim any convergence over the whole exterior domain. The error is reported in percent of the semi-norm of the solution, defined over the same domain. The exact solution for the cylinder problem is computed using the standard formula involving Hankel functions [82], compare Exercise 21.1. For the wedge problem, the unknown solution is replaced with the solution on the hp-refined mesh i.e., as in the previous chapters, by error we mean the norm of the difference of the fine and the coarse grid solutions.

21.6 Numerical experiments

Scattering of a plane wave on a PEC cylinder. Verification of the code. We begin with the standard example of scattering a plane wave,

$$\boldsymbol{E}^{inc} = -\frac{1}{ik}\boldsymbol{\nabla} \times e^{-ik\boldsymbol{e}\cdot\boldsymbol{x}} ,$$

on a unit PEC cylinder. Here $\boldsymbol{e}$ specifies the direction of the incident wave, in our case $\boldsymbol{e} = (-1, 0)$.

We set the wave number to $k = \pi$, and truncate the infinite domain with a circle of radius $a = 3$. Fig. 21.2 displays convergence history for p-uniform and hp-adaptive refinements, starting with a mesh of 16 quadratic elements. The horizontal axis corresponds to the number of d.o.f. n displayed on the algebraic scale $n^{1/3}$, with the vertical axis presenting the error on the logarithmic scale. A straight line indicates the exponential convergence,

$$\text{error} \approx Ce^{\beta n^{1/3}} ,$$

predicted by the theory. Notice that the actual numbers displayed on the axes correspond to the quantities being displayed, i.e. the relative error in percent of the norm of the solution on the vertical axis, and the number of d.o.f. on the horizontal axis.

As expected, the uniform p refinements deliver exponential convergence, with the adaptive refinements delivering slightly worse results but the same rates. Fig. 21.3 shows the optimal hp mesh,

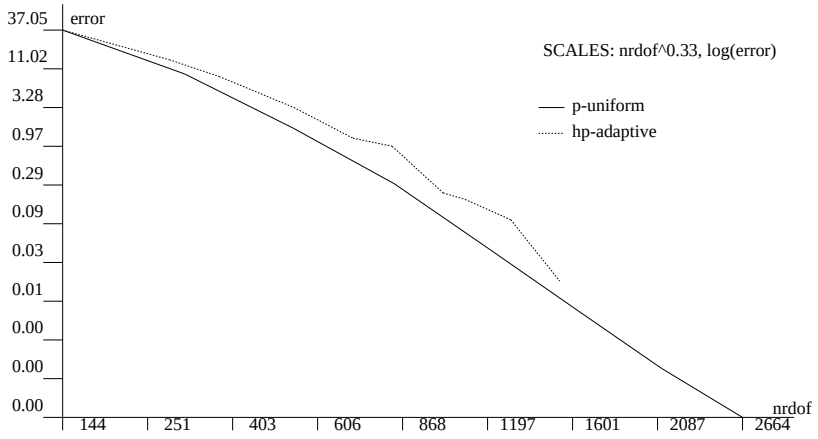


Figure 21.2: Scattering of a plane wave on a PEC cylinder. Convergence history for p-uniform and hp-adaptive refinements

corresponding to an error of 1 percent. Different colors indicate different (locally changing) order

of approximation p ranging from $p = 1$ to $p = 8$ (color scale on the right). The distribution of orders clearly reflects the pattern of the solution.

Fig. 21.4 presents contour lines of the real part of the error function, for a uniform mesh of quartic elements. The values, indicated by the color scale on the right, range from -0.02 to 0.02. Along with the FE mesh, the graph displays a portion of the infinite elements corresponding to $0.5 < x < 1$. The solution in the IE domain seems to be actually better than in the finite element domain which indicates that lower order infinite elements would have been sufficient.

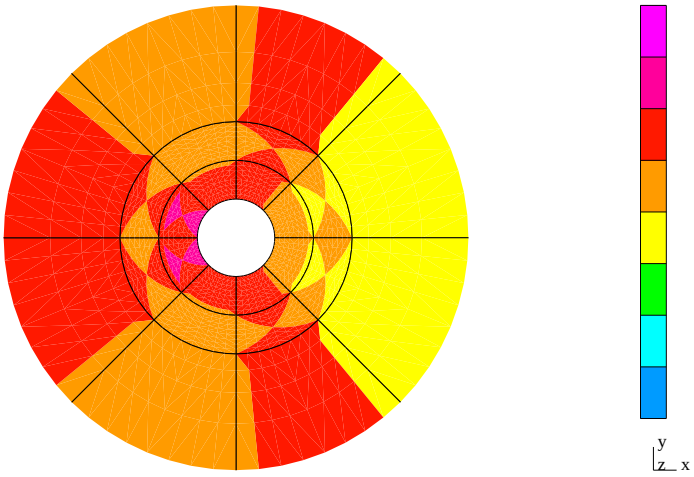


Figure 21.3: Scattering of a plane wave on a cylinder. Optimal hp mesh corresponding to 1 percent error. Different colors indicate different (locally changing) order of approximation p ranging from $p = 1$ to $p = 8$ (color scale on the right

Scattering of a plane wave on a PEC wedge. The second example illustrates the resolution of singularities using the coupled hp-adaptive FE/IE discretizations. We have divided the cylinder from the previous example into four equal quadrants and kept just one of them. We set the wave number $k = \pi$, i.e. the distance between the truncating boundary and the object is equal to one wavelength. We start with an example of a typical, automatically obtained hp mesh corresponding to the incident wave coming from the NE direction (angle θ = 45^o), and rather academic error level of 0.1 percent. Figures 21.5 and 21.6 present the optimal mesh, with three consecutive zooms showing details of the mesh around the lower corner. Fig. 21.7 presents convergence history for the problem. We start with an initial mesh of just seven elements of second order that clearly do not resolve the wave pattern (one second order element per wave length) to illustrate the principle that only the fine grid must be in the asymptotic convergence region. Consequently, the convergence curve consists roughly of two straight lines, the first one corresponding to the preasymptotic region,

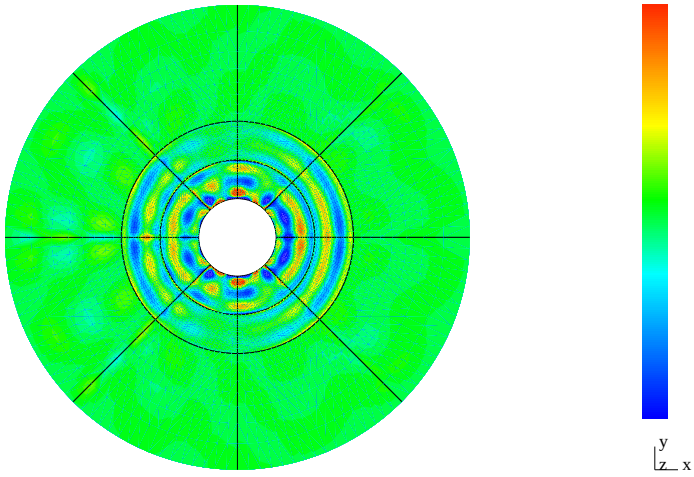


Figure 21.4: Scattering of a plane wave on a cylinder. Real part of the second component of the error function for a uniform mesh of quartic elements

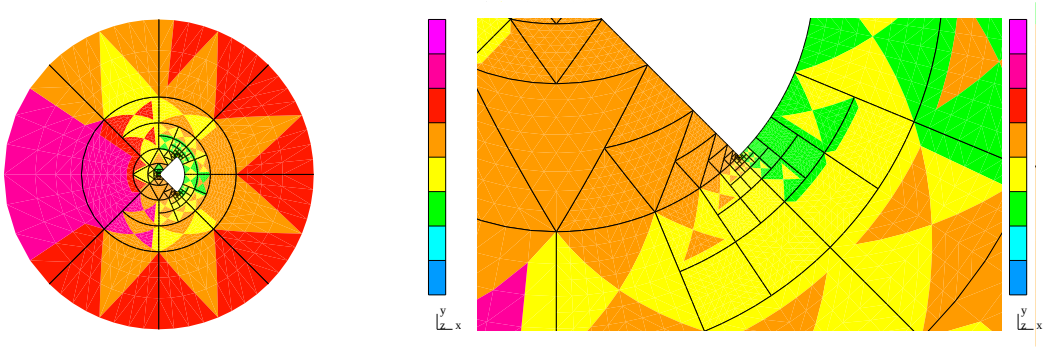


Figure 21.5: Scattering of a plane wave on a PEC wedge, $\theta = 45^o$. Optimal hp mesh for 0.1 percent error, with a 10 times zoom on the lower corner

and the second one reflecting the actual exponential convergence.

Evaluation of RCS. We come to the final experiment reflecting the impact of adaptivity on evaluation of Radar Cross Sestion (RCS). For two-dimensional problems, the monostatic RCS reflects simply the far-field pattern of the solution, and it is defined as follows.

$$\lim_{r\to\infty} |\boldsymbol{E}(r\hat{\boldsymbol{x}})| r^{\frac{1}{2}} \ .$$

Here $\hat{\boldsymbol{x}}$ is a point on the unit circle corresponding to the direction of the incoming plane wave $\boldsymbol{e} = -\hat{\boldsymbol{x}}$, and $r^{\frac{1}{2}}$ compensates the decay rate of the solution.

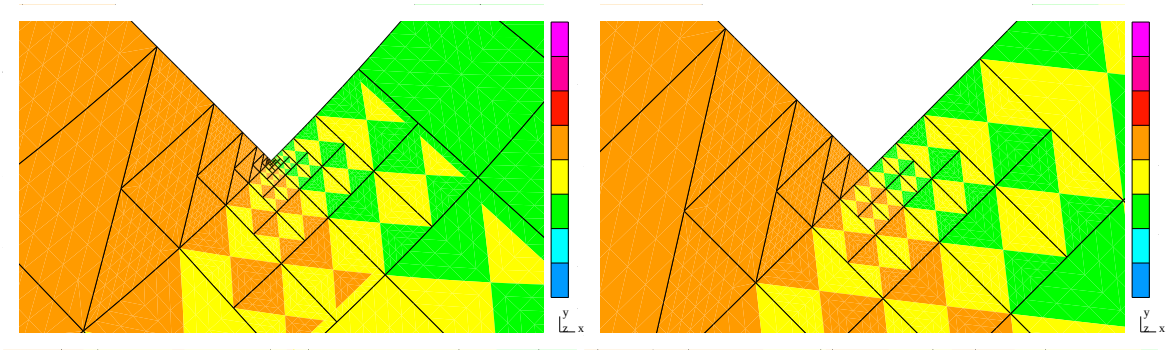


Figure 21.6: Scattering of a plane wave on a PEC wedge, $\theta = 45^o$. Optimal hp mesh for 0.1 percent error. Zooms on the lower corner with 100 and 1000 magnifications



Figure 21.7: Scattering of a plane wave on a PEC wedge, $\theta = 45^o$. Convergence history for adaptive hp refinements

Similarly as for the Helmholtz equation, the infinite element discretization offers an inexpensive way of computing the RCS without the usual costly postprocessing involving integration of surface electric and magnetic currents on any closed surface surrounding the scatterer. As the far-field pattern is built into the ansatz for the approximate solution, one simply evaluates the approximate solution at $r = \infty$ $(x = 0)$.

Fig. 21.8 presents RCS for the wedge problem evaluated using a uniform mesh of quartic elements and and hp-adaptive mesh. The choice of the uniform mesh reflects the usual practice of selecting a mesh that reproduces the wave form of the solution (two quartic elements per wave-

length) and delivers an error in the range of 3-4 percent. The second curve corresponds to RCS evaluated using hp-adaptivity.

The hp meshes were obtained by requesting a two percent error level, at which several levels of hp-refinements resolve the structure of the singularities in the solution. For each direction of the incoming wave ($\theta = 180, 179, \ldots, 0$, left to right), the hp-adaptive algorithm was run starting with the optimal mesh for the previous direction, with the optimization procedure restarted from the initial mesh every 10 degrees. Except for a slight shift in the RCS level, the results are practically

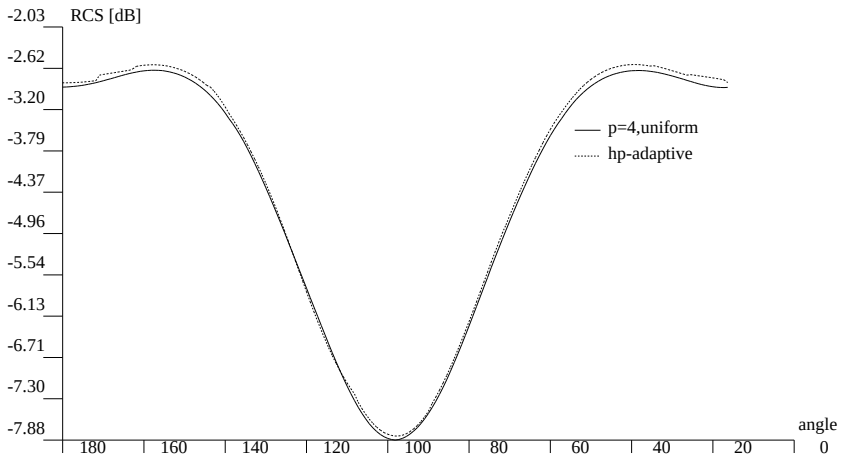


Figure 21.8: Scattering of a plane wave on a wedge. RCS in dB vs. the direction of the incident wave in degrees, for the uniform mesh of quartic elements (3-4 percent error range level) and hp-adaptive mesh (2 percent error)

identical.

For the presented example, the resolution of the singularities seems to have no impact on the quality of the RCS computations.

Exercises

Exercise 21.1 Derive the solution

Bibliography

[1] M. Abramowitz and I.A. Stegun, editors. *Handbook of Mathematical Functions, with Formulas, Graphs and Mathematical Tables*, volume 55. National Bureau of Standards, Washington, D.C., (for sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C., 20402), 1966. Fifth printing, with corrections.

[2] H. Adams. *Sobolev Spaces*. Academic Press, New York, 1978.

[3] M. Ainsworth and J. Coyle. Hierarchic hp-edge element families for maxwell's equations on hybrid quadrilateral/triangular meshes. *Comput. Methods Appl. Mech. Engrg.*, 190:6709–6733, 2001.

[4] M. Ainsworth and J.T. Oden. *A Posteriori error Estimation in Finite Element Analysis*. Wiley and Sons, Inc., New York, 2000.

[5] M. Ainsworth and B. Senior. Aspects of an adaptive hp-finite element method: Adaptive strategy, conforming approximation, and efficient solvers. *Comput. Methods Appl. Mech. Engrg.*, 150:65–87, 1997.

[6] B. Andersson. Computational methods for reliable fatigue and damage tolerance dimensioning of aircraft structures. In *Proceedings of Fifth Nordic Seminar on Computational Mechanics*, Aalborg, 1992.

[7] B. Andersson. Calculation of fracture mechanics parameters from measured surface displacements in composite specimens. In *Proceedings of 6th Nordic Seminar on Computational Mechanics*, pages 181–186, 1993.

[8] B. Andersson, I. Babuska, and U. Falk. Accurate and reliable determination of edge and vertex stress intensity factors in three-dimensional elastomechanics. In *Proceedings of 17th Congress of the International Council of the Aeronautical Sciences*, pages 1730–1746, 1990.

[9] B. Andersson and U. Falk. Reliable stress and fracture mechanics analysis of complex aircraft components. In *Proceedings of the 17th Symposium of the International Committe on Aeronautical Fatigue*, pages 389–411, Stockholm, Sweden, 1993.

[10] B. Andersson, U. Falk, I. Babuska, and T. von Petersdorff. Reliable stress and fracture mechanics analysis of complex components using an hp version of fem. *Int. J. Num. Meth. Eng.*, 38:2135–2163, 1995.

[11] B. Andersson, A. Sjogren, and L. Berglund. Micro- and mesolevel residual stresses in glass-fiber/vinylester composites. *Composites Science and Technology*, 60:2011–2028, 2000.

[12] D.N. Arnold, B. Boffi, and R.S. Falk. Quadrilateral $h(div)$ finite element. *I.A.N.-C.N.R*, 1283:1–23, 2002. (http://www.imati.cnr.it/ian/PUBBLICAZIONI/Pubblicazioni2002.html).

[13] R.J. Astley and J.P. Coyette. Conditioning of infinite element schemes for wave problems. *Communications in Numerical Methods in Engineering*, 17:31–41, 2001.

[14] R.J. Astley, G.J. Macaulay, and J.P. Coyette. Mapped wave envelope elements for acoustical radiation and scattering. *Journal of Sound and Vibration*, 170(1):97–118, 1994.

[15] B. Babuska, B. Andersson, P.J. Smith, and K. Levin. Damage analysis of fiber composites. *Comput. Methods Appl. Mech. Engrg.*, 172:27–77, 1999.

[16] I. Babuska and Th. Strouboulis. *The Finite Element Method and its Reliability*. Clarendon Press, Oxford, 2001.

[17] I. Babuška. Error bounds for finite element method. *Numerische Mathematik*, 16:322–333, 1971.

[18] I. Babuška and B.Q. Guo. Approximation properties of the hp version of the finite element method. *Comput. Methods Appl. Mech. Engrg.*, 133:319–346, 1996.

[19] I. Babuška, R.B. Kellogg, and J. Pitkäranta. Direct and inverse error estimates for finite elements with mesh refinement. *Numer. Math.*, 33:447–471, 1979.

[20] I. Babuška and J. Osborn. Eigenvalue problems. In P.G. Ciarlet and J.L. Lions, editors, *Hanbook of Numerical Analysis*. Elsevier Science Publishers B.V.(North Holland), 1991.

[21] A. Bedford and K.M. Liechti. *Mechanics of Materials*. Prentice Hall, 2000.

[22] J.-P. Bérenger. A perfectly matched layer for the absorption of electromagnetic waves. *Journal of Computational Physics*, 114:185–200, 1994.

[23] P. Bettess. *Infinite Elements*. Penshaw Press, 1992.

[24] A.F. Blom and B. Andersson. On the semi-elliptical surface crack problem: Detailed numerical solutions for complete elastic stress fields. Standard Technical Publication 1060, American Society for Testing and Materials, 1990.

[25] T.Z. Boulmezaoud. Inverted finite elements: a new method for solving elliptic problems in unbounded domains. *Mathematical Modeling and Numerical Analysis (M^2AN)*, 2005. to appear.

[26] D.S. Burnett. A three-dimensional acoustic infinite element based on a prolate spheroidal multipole expansion. *Journal of the Acoustical Society of America*, 96:2798–2816, 1994.

[27] J. Cea. cea's lemma. *Annales de l'Institut Fourier*, 1964.

[28] W. Cecot, L. Demkowicz, and R. Rachowicz. A two-dimensional infinite element for Maxwell's equations. *Comput. Methods Appl. Mech. Engrg.*, 188:625–643, 2000.

[29] W. Cecot, L. Demkowicz, and W. Rachowicz. An hp-adaptive finite element method for electromagnetics. part 3: A three-dimensional infinite element for Maxwell's equations. *Int. J. Num. Meth. Eng.*, 57:899–921, 2003.

[30] Y.C. Chang and L. Demkowicz. Solution of viscoelastic scattering problems using coupled hp be/fe methods. *Int. J. Num. Meth. Eng.*, 44(12):1885–1907, 1999.

[31] A.J. Chapman. *Heat Transfer*. Macmillan Publishing Company, New York, 1984.

[32] Ph. G. Ciarlet. *The Finite Element Methods for Elliptic Problems*. North Holland, New York, 1994.

[33] D. Colton and R. Kress. *Inverse Acoustic and Electromagnetic Scattering*. Springer Verlag, Berlin, 1992.

[34] L. Cremers, K.R. Fyfe, and J.P. Coyette. A variable order infinite acoustic wave envelope element. *Journal of Sound and Vibrations*, 17(4):483–508, 1994.

[35] F. Cugnon. *Automatisation des Caculs Éléments Finis dans le Cadre de la Méthode p*. PhD thesis, Universite de Lieg, 2000.

[36] L. Demkowicz. *Adaptive Finite Element Methods*. Cracow University of Technology Press, Monographia 46, Cracow, 1986. (Habilitation Thesis, in Polish).

[37] L. Demkowicz. Asymptotic convergence in finite and boundary element methods. part 1: Theoretical results. *Computers & Mathematics with Applications*, 27(12):69–84, 1994.

[38] L. Demkowicz. Viscoelastic vibrations in fluid. analysis of a model problem. Technical Report 12, TICAM, 1996.

[39] L. Demkowicz. Edge finite elements of variable order for Maxwell's equations. In D. Hecht U. van Rienen, M. Günther, editor, *Scientific Computing in Electrical Engineering, Lecture Notes in Computational Science and Engineering* **18**, pages 15–34. Springer Verlag, Berlin,

2000. (Proceedings of the 3rd International Workshop, August 20-23, Warnemuende, Germany).

[40] L. Demkowicz. 2D hp-adaptive finite element package (2Dhp90). version 2.0. Technical Report 06, TICAM, 2002.

[41] L. Demkowicz. Finite element methods for Maxwell equations. In E. Stein R. de Borst, T.J.R. Hughes, editor, *Encyklopedia of Computational Mechanics*, chapter 26, pages 723–737. John Wiley & Sons, Ltd, Chichester, 2004.

[42] L. Demkowicz. Babuška $\Leftrightarrow$ Brezzi ? Technical report, ICES, 2005.

[43] L. Demkowicz and I. Babuška. p interpolation error estimates for edge finite elements of variable order in two dimensions. *SIAM J. Numer. Anal.*, 41(4):1195–1208 (electronic), 2003.

[44] L. Demkowicz, A. Bajer, W. Rachowicz, and K. Gerdes. 3D hp-adaptive finite element package. fortran 90 implementation. Technical Report 29, TICAM, 1999.

[45] L. Demkowicz and A. Buffa. H^1, $H(\mathrm{curl})$ and $H(\mathrm{div})$-conforming projection-based interpolation in three dimensions. quasi-optimal p-interpolation estimates. *Comput. Methods Appl. Mech. Engrg*, 194:267–296, 2005.

[46] L. Demkowicz, K. Gerdes, Ch. Schwab, A. Bajer, and T. Walsh. Hp90: A general and flexible fortran 90 hp-FE code. *Computing and Visualization in Science*, 1:145–163, 1998.

[47] L. Demkowicz and F. Ihlenburg. Analysis of a coupled finite-infinite element method for exterior Helmholtz problems. *Numer. Math.*, 88:43–73, 2001.

[48] L. Demkowicz, A. Karafiat, and J.T. Oden. Solution of elastic scattering problems in linear acoustics using hp boundary element method. *Comput. Methods Appl. Mech. Engrg.*, 101:251–282, 1991. (Proceedings of Second Workshop on Reliability and Adaptive Methods in Computational Mechanics, Cracow, October 1991, eds. L. Demkowicz, J.T. Oden, and I. Babuska).

[49] L. Demkowicz, P. Monk, L. Vardapetyan, and W. Rachowicz. de Rham diagram for hp finite element spaces. *Comput. Math. Appl.*, 39(7-8):29–38, 2000.

[50] L. Demkowicz and J.T. Oden. Application of hp-adaptive be/fe methods to elastic scattering. *Comput. Methods Appl. Mech. Engrg.*, 133(3-4):287–318, 1996. a special issue on p and hp methods, edited by I. Babuska and J.T. Oden.

[51] L. Demkowicz, J.T. Oden, M. Ainsworth, and P. Geng. Solution of elastic scattering problems in linear acoustics using hp boundary element methods. *Journal of Computational and Applied Mathematics*, 36:29–63, 1991.

[52] L. Demkowicz, J.T. Oden, and Ph. Devloo. On h-type mesh refinement strategy based on a minimization of interpolation error. *Comput. Methods Appl. Mech. Engrg.*, 53:67–89, 1985.

[53] L. Demkowicz, J.T. Oden, and R. Rachowicz. A new finite element method for solving compressible navier-stokes equations based on an operator splitting method and hp adaptivity. *Comput. Methods Appl. Mech. Engrg.*, 84:275–326, 1990.

[54] L. Demkowicz, J.T. Oden, and W. Rachowicz. Toward a universal hp adaptive finite element strategy, part 1. constrained approximation and data structure. *Comput. Methods Appl. Mech. Engrg.*, 77:79–112, 1989.

[55] L. Demkowicz, J.T. Oden, W. Rachowicz, and O. Hardy. An hp Taylor-Galerkin finite element method for compressible Euler equations. *Comput. Methods Appl. Mech. Engrg.*, 88(3):363–396, 1991.

[56] L. Demkowicz and M. Pal. An infinite element for Maxwell's equations. *Comput. Methods Appl. Mech. Engrg.*, 164:77–94, 1998.

[57] L. Demkowicz, D. Pardo, and W. Rachowicz. 3D hp-adaptive finite element package (3Dhp90). version 2.0. Technical Report 24, TICAM, 2002.

[58] L. Demkowicz, W. Rachowicz, and Ph. Devloo. A fully automatic hp-adaptivity. *Journal of Scientific Computing*, 17(1-3):127–155, 2002.

[59] L. Demkowicz and J. Shen. A few new (?) facts about infinite elements. Technical Report 60, ICES, 2004.

[60] L. Demkowicz and L. Vardapetyan. Modeling of electromagnetic absorption/scattering problems using hp-adaptive finite elements. *Comput. Methods Appl. Mech. Engrg.*, 152(1-2):103–124, 1998.

[61] L. Demkowicz, T. Walsh, K. Gerdes, and A. Bajer. 2D hp-adaptive finite element package. fortran 90 implementation. Technical Report 14, TICAM, 1998.

[62] Ph. Devloo. Object oriented tools for scientific computing. *Engineering with Computers*, 16:63–72, 2000.

[63] S. Dey, M.S. Shephard, and J.E. Flaherty. Geometry-based issues associated with p-version finite element computations. *Comput. Methods Appl. Mech. Engrg.*, 150:39–50, 1997.

[64] D.C. Dobson and J.E. Pasciak. Analysis of an algorithm for computing electromagnetic Bloch modes using Nedelec spaces. *Comp. Meth. Appl. Math.*, 1(2):138–153, 2001.

[65] D.A. Dunavant. High-degree efficient symmetric gaussian quadrature rules for the triangle. *Int. J. Num. Meth. Eng.*, 21:1129–1148, 1985.

[66] A. Düster. *High Order Finite Elements for Three-Dimensional, Thin-Walled Nonlinear Continua*. Shaker Verlag, Aachen, 2002.

[67] A. Eriksson, C. Pacoste, and A. Zdunek. Numerical analysis of complex instability behaviour using incremental-iterative strategies. *Comput. Methods Appl. Mech. Engrg.*, pages 265–305, 1999.

[68] S. Fawaz and B. Andersson. Accurate stress intensity factor solutions for unsymmetric corner cracks at a hole. In *Proceedings of Fourth Joint DoD/FAA/NASA Conference on Aging Aircraft*, St Louis, Missouri, USA., 2000.

[69] Ph. Frauenfelder and Ch. Lage. Concepts – an object oriented software package for partial differential equations. *Mathematical Modelling and Numerical Analysis (M2AN)*, 36(5):937–951, 2002.

[70] Y-C Fung. *Foundations of Solid Mechanics*. Englewood Cliffs, N., 1965.

[71] K. Gerdes and L. Demkowicz. Solution of 3D Laplace and Helmholtz equation in exterior domains using hp infinite elements. *Comput. Methods Appl. Mech. Engrg.*, 137:239–274, 1996.

[72] D. Givoli. *Numerical Methods for Problems in Infinite Domains*. Elsevier, Amsterdam, 1992.

[73] J. Gopalakrishnan, L.E. García-Castillo, and L. Demkowicz. Nédeélec spaces in affine ccordinates. *Computares and Mathematics with Applications*, 49:1285–1294, 2005.

[74] W.J. Gordon and C.A. Hall. Transfinite element methods: Blending function interpolation over arbitary curved element domain. *Numer. Math.*, 21:109–129, 1973.

[75] W. Gui and I. Babuska. The h, p and hp versions of the finite element method in one dimension. part 1: The error analysis of the p-version, part 2: The error analysis of the h- and hp-versions. part 3: The adaptive hp vesrion. *Numer. Math.*, 49:577–683, 1986.

[76] G. Guo and I. Babuska. The hp version of the finite element method. part 1: The basic approximation results. part 2: General results and applications. *Comput. Mech.*, 1:21–41,203–220, 1986.

[77] R. Hiptmair. Canonical construction of finite elements. *Mathematics of Computation*, 68, 1999. 1325-1346.

[78] R. Hiptmair. Higher order Whitney forms. Technical Report 156, Universität Tübingen, 2000.

[79] P. Houston and E. Suli. hp-adaptive discontinuous galerkin finite element methods for first order hyperbolic problems. *SIAM Journal on Scientific Computing*, 23(4):1226–1252, 2001.

[80] T.J.R. Hughes. *The Finite Element Method - Linear Static and Dynamic Finite Element Analysis*. Dover Publishers, 1998, 2000.

[81] F. Ihlenburg. *Finite Element Analysis of Acoustic Scattering*. Springer-Verlag, New York, 1998.

[82] D.S. Jones. *Acoustic and Electromagntic Waves*. Oxford Science Publications, 1986.

[83] M.C. Junger and D. Feit. *Sound, Structures, and Their Interaction*. MIT Press, Boston, 1986.

[84] G.E. Karniadakis and A. Beskok. *Microflows: Fundamentals and Simulation*. Springer Verlag, 2001.

[85] G.E. Karniadakis and S.J. Sherwin. *Spectral/hp Element Methods for CFD*. Oxford University Press, 1999.

[86] T. Kato. Estimation of iterated matrices, with application to the von neumann condition. *Numer. Math.*, 2:22–29, 1960.

[87] D. Kincaid and W. Cheney. *Numerical Analysis*. Brooks/Cole Publishing Company, 1996. 2nd ed.

[88] P.D. Lax and R.S. Phillips. *Scattering Theory*. Academic Press, Inc., 1989.

[89] P.D. Ledger. *An hp-Adaptive Finite Element Procedure for Electromagnetic Scattering Problems*. PhD thesis, University of Wales, Swansea, Nov. 2001.

[90] L.P. Meissner. *Fortran 90*. PWS Publishing Company, Boston, 1995.

[91] J.M. Melenk. *hp-Finite Element Methods for Singular Perturbations*. Springer Verlag, 2002.

[92] J.C. Nedelec. Mixed finite elements in $\mathbb{R}^3$. *Numer. Math.*, 35:315–341, 1980.

[93] J.C. Nedelec. A new family of mixed finite elements in $\mathbb{R}^3$. *Numer. Math.*, 50:57–81, 1986.

[94] K.F. Nilsson and B. Andersson. Analysis methodology for fatigue crack propagation and residual strength of joints with widespread fatigue damage. In *USAF Aircraft Structural Integrity Program (ASIP) Conference*, San Antonio, Texas, USA, 1999.

[95] A.A. Novotny, J.T. Pereira, E.A. Fancello, and C.S. de Barcellos. A fast hp adaptive finite element mesh design for 2d elliptic boundary value problems. *Comput. Methods Appl. Mech. Engrg.*, 190(1-2):133–148, 2000.

[96] J.T. Oden, L. Demkowicz, T. Liszka, and W. Rachowicz. hp adaptive finite element methods for compressible and incompressible flows. *Computing Systems in Engineering*, 1(2-4):523–534, 1990.

[97] J.T. Oden, L. Demkowicz, R. Rachowicz, and T.A. Westermann. Toward a universal hp adaptive finite element strategy, part 2. a posteriori error estimation. *Comput. Methods Appl. Mech. Engrg.*, 77:113–180, 1989.

[98] J.T. Oden and L.F. Demkowicz. *Applied Functional Analysis for Science and Engineering*. CRC Press, Boca Raton, 1996.

[99] J.T. Oden, A. Patra, and Y. Feng. An hp adaptive strategy. In A.K. Noor, editor, *Adaptive Multilevel and Hierarchical Computational Strategies*, pages 23–46. ASME Publication 157, 1992.

[100] J.T. Oden, W. Wu, and M. Ainsworth. Three step hp adaptive strategy for the incompressible navier–stokes equations. In J.E. Flaherty I. Babuška, editor, *Modeling, Mesh Generation and Adaptive Numerical Methods for Partial Differential Equations*. IMA Minnesota, 1995.

[101] A. Patra and A. Gupta. A systematic strategy for simultaneous adaptive hp finite element mesh modification using nonlinear programming. *Comput. Methods Appl. Mech. Engrg.*, 190:3797–3818, 2001.

[102] W. Rachowicz. *An hp finite element method for one-irregular meshes, error estimation and mesh refinement strategy*. PhD thesis, The University of Texas at Austin, 1989.

[103] W. Rachowicz and L. Demkowicz. "an hp-adaptive finite element method for electromagnetics. part 1: Data structure and constrained approximation. *Comput. Methods Appl. Mech. Engrg.*, 187:625–643, 2000.

[104] W. Rachowicz and L. Demkowicz. An hp-adaptive finite element method for electromagnetics. II. A 3D implementation. *Internat. J. Numer. Methods Engrg.*, 53(1):147–180, 2002. p and hp finite element methods: mathematics and engineering practice (St. Louis, MO, 2000).

[105] W. Rachowicz, J.T. Oden, and L. Demkowicz. Toward a universal hp-adaptive finite element strategy, part 3: Design of hp meshes. *Comput. Methods Appl. Mech. Engrg.*, 77:181–212, 1989.

[106] W. Rachowicz and A. Zdunek. An hp-adaptive finite element method for scattering problems in computational electromagnetics. *International Journal for Numerical Methods in Engineering*, 2005.

[107] E. Rank. A zooming technique using a hierarchical hp-version of the finite element method. In J. Whiteman, editor, *The Mathematics of Finite Elements and Applications - Highlights 1993*. Elsevier, 1993.

[108] J.E. Remacle, J.-F. Flaherty and M.S. Shephard. Parallel algorithm oriented mesh database. *Engineering with Computers*, 2001.

[109] W.C. Rheinboldt and C.K. Mesztenyi. On a data structure for adaptive finite element mesh refinements. *ACM Transactions on Mathematical Software*, 6(2):166–187, 1980.

[110] A. Safjan, L. Demkowicz, and J.T. Oden. Adaptive finite element methods for hyperbolic systems with application to transient acoustics. *Int. J. Num. Meth. Eng.*, 32:677–707, 1991.

[111] A. Safjan and M. Newman. On two-dimensional infinite elements utilizing basis functions with compact support. *Comput. Methods Appl. Mech. Engrg.*, 190(48):6399–6424, 2001.

[112] J. Schon and B. Andersson. Interaction between a delamination crack and a matrix crack", proceedings of the american society for composites. In *Proceedings of the American Society for Composites, 13'th Annual Technical Conference*, Baltimore MD, USA, 1998.

[113] Ch. Schwab. *p and hp-Finite Element Methods*. Clarendon Press, Oxford, 1998.

[114] L. Schwartz. *Cours d'analyse*. Hermann, Paris, 1967. (in French).

[115] T. B. A. Senior and J. L. Volakis. *Approximate Boundary Conditions in Electromagnetics*. IEEE Press, New York, 1995.

[116] S. Shephard, M.S. Dey and J.E. Flaherty. A straightforward structure to construct shape functions for variable p-order meshes. *Comput. Methods Appl. Mech. Engrg.*, 147:209–233, 1997.

[117] B.A. Szabo and I. Babuška. *Finite Element Analysis*. Wiley, New York, 1991.

[118] S.V. Tsynkov. Numerical solution of problems on unbounded domains. a review. *Applied Numerical Mathematics*, 27:465–532, 1998.

[119] L. Vardapetyan and L. Demkowicz. hp-adaptive finite elements in electromagnetics. *Comput. Methods Appl. Mech. Engrg.*, 169:331–344, 1999.

[120] M. Vogelius and I. Babuska. An analysis of the p-version of the finite element method for nearly incompressible materials. uniformly valid optimal error estimates. *Num. Math.*, pages 39–53, 1983.

[121] J.P. Webb. "hierarchical vector based funtions of arbitrary order for triangular and tetrahedral finite elements. *IEEE Antennas Propagat. Mag.*, 47(8):1244–1253, 1999.

[122] D. Xue, L. Demkowicz, and A. Zdunek. An interface between geometrical modeling package (gmp) and mesh-based deometry (mbg). Technical Report 20, ICES, 2003.

[123] L. Xue, D.and Demkowicz. Geometrical modeling package. version 2.0. Technical Report 30, TICAM, 2002.

[124] A. Zdunek and W. Rachowicz. A three-dimensional hp-adaptive finite element approach to radar scattering problems. In *Fifth World Congress on Computational Mechanics*, Vienna, Austria, 2002.

[125] G. Zumbusch. *Simultaneous hp Adaption in Multilevel Finite Elements*. Shaker Verlag, Aachen, 1996.