HP90: a general & flexible fortran 90 hp-FE code

Eidgenössische Technische Hochschule Zürich

Seminar für Angewandte Mathematik SAM

HP90: A general & flexible Fortran 90 *hp*-FE code

L. Demkowicz[1], K. Gerdes[2], C. Schwab, A. Bajer[1] and T. Walsh[1]

Research Report No. 97-17
December 1997

Seminar für Angewandte Mathematik
Eidgenössische Technische Hochschule
CH-8092 Zürich
Switzerland

[1]TICAM, The University of Texas at Austin, Austin, TX 78712, USA
[2]email: gerdes@sam.math.ethz.ch

HP90: A general & flexible Fortran 90 *hp*-FE code

L. Demkowicz[1], K. Gerdes[2], C. Schwab, A. Bajer[1] and T. Walsh[1]

Seminar für Angewandte Mathematik
Eidgenössische Technische Hochschule
CH-8092 Zürich
Switzerland

Research Report No. 97-17 December 1997

Abstract

A general 2D-*hp*-adaptive Finite Element (FE) implementation in Fortran 90 is described. The implementation is based on an abstract data structure, which allows to incorporate the full *hp*-adaptivity of triangular and quadrilateral finite elements.

The h-refinement strategies are based on $h2$-refinement of quadrilaterals and $h4$-refinement of triangles. For p-refinement we allow the approximation order to vary within any element. The mesh refinement algorithms are restricted to 1-irregular meshes. Anisotropic and geometric refinement of quadrilateral meshes is made possible by additionally allowing double constrained nodes in rectangles.

The capabilities of this *hp*-adaptive FE package are demonstrated on various test problems.

Keywords: *hp* finite element method, spectral element method, constrained approximation, anisotropic mesh refinement

Subject Classification: Primary: 65N30, 65N35, Secondary: 65N50

[1]TICAM, The University of Texas at Austin, Austin, TX 78712, USA
[2]email: gerdes@sam.math.ethz.ch

1 Introduction

The Finite Element Method (FEM) is today the most widely used discretization technique in solid and fluid mechanics. The classical approach still dominant in today's industrial applications is to partition the domain into many small subdomains of diameter $O(h)$ and to approximate the unknown solution by a piecewise polynomial of low order p (typically, in applications, $p = 1$ or $p = 2$). Convergence is achieved through mesh refinement, i.e. by letting $h \to 0$.

In the early eighties, the so-called p-Version FEM emerged where the polynomial degree $p \to \infty$ on a fixed mesh in order to produce convergent sequences of FE-solutions (see, e.g., [1] and the references there). It has been shown in the meantime that p-FEM are substantially less susceptible to locking phenomena than the abovementioned h-type FEM and, moreover, give *exponential* rates of convergence for many problems in solid and fluid mechanics for piecewise analytic solutions typically arising in practice [17, 26].

Early implementations of p-FEM were PROBE of Noetic Tech. and, more recently, STRESSCHECK of ESRD Res. Corp. in St. Louis, Mo. where the mesh was fixed and the polynomial degree was assumed uniform.

The first implementation of a *general hp*-FEM where the mesh as well as the elemental polynomial degree are completely general was available for two dimensional problems in the late 80ies [3, 4]. There general meshes consisting of triangles and/or quadrilaterals and even certain types of irregular ("hanging") nodes were admissible. A commercial version of this code became available under the name PHLEX of COMCO Corp. in Austin, Tx. The early work had been strongly determined by considerations to save RAM and could also accomodate standard, continuous FE-spaces.

In recent years, however, the advantages of nonstandard mixed and hybrid variational formulations of boundary value problems and of the FEM based on such formulations have been widely recognized [6] and also *hp*-variants of many mixed elements (such as the Raviart-Thomas, MITC [30], Nedelec-edge elements [19]) have been analyzed, but computational experience with such *hp*-versions for problems in Mechanics is lacking. The situation is similar with regard to the classical Stokes and Navier-Stokes problem (although equal order interpolants for velocity and pressure together with SUPG stabilization proved to be quite successful [25])

Thus there arises the need for a *flexible* implementation of *general hp*-FEM. Here *flexible* means that meshes consisting of triangles and/or quadrilaterals with possibly hanging nodes are admissible. On such meshes we admit arbitrary polynomial degree distributions, and even allow variable polynomial degree within elements. *General hp*-FEM is to be understood in the sense that not only C^0-conforming, "classical" elements are to be considered, but also mixed element pairs where no or only partial continuity constraints are enforced between elements (as is the case with $H(div, \Omega)$ and $H(curl, \Omega)$ conforming elements or the MITC-family). This flexibility is mandatory, for example, in the context of *multiple field problems* where elliptic systems of governing equations of possibly quite different mathematical character are to be discretized simultaneously (as, e.g., incompressible fluid flow with heat exchange, electromagnetic scattering coupled with elasticity etc.)

To present a general class of FE-spaces capable of realizing *hp*-versions for nearly all finite elements and to describe its successful implementation on general meshes in two dimensions is the purpose of the present paper. Departing from discontinuous, piecewise polynomial vector functions of possibly different degrees on the abovementioned general class of meshes, we describe a general constraining strategy which allows to realize all possible interelement continuity conditions — from completely discontinuous to C^0, even in the presence of hanging

nodes. We describe in detail how these elements are implemented in FORTRAN 90.

Frequently, in applications, *adaptive* mesh refinement leads to superior performance of FE-codes. Our implementation can accomodate very general types of refinements – in particular also anisotropic refinements which are imperative to resolve boundary layers (as, e.g. the viscous boundary layer in incompressible fluid flow or the skin-effect in electromagnetics) at an exponential rate of convergence.

The outline of this paper is as follows. In section 2 we define the $hp-$FE meshes and the mesh refinement algorithms. The corresponding FE spaces are presented in section 3 together with examples of variational formulations. In section 4 we give a general overview of the capabilities of this FE package. The new dynamic data structure is introduced in section 5. Section 6 deals with the constrained approximation procedure and section 7 reveals details about the element computations and the generalized assembling procedure. Numerical examples are presented in section 8 and we finish with a summary in section 9.

2 The *hp*-FE Meshes

In this section we describe the meshes that our FE package can generate and handle. We start from the assumption that the initial triangulation is always regular. At first we focus on the geometrical description, i.e. on describing the triangulations and the possible subdivisions of such triangulations. Then we introduce the polynomial spaces on the master elements and define the element mappings that map each element from the triangulation onto its corresponding master element. For each element we also define the polynomial degree and these approximation orders together with the triangulation define then the *hp*-mesh.

2.1 The Initial Triangulation

By $\Omega \subset \mathbb{R}^N, N = 2,3$ we denote a bounded two dimensional linear manifold with piecewise analytic Lipschitz boundary $\Gamma = \partial\Omega$, i.e. the sides Γ_j are analytic curves. A partition $\mathcal{T}$ of Ω with subdomains $K \in \mathcal{T}$ is called an *initial triangulation* of Ω if

- each subdomain $K \in \mathcal{T}$ is either a curvilinear triangle or quadrilateral,
- the intersection of any two $K, K' \in \mathcal{T}$, $K \neq K'$, is either empty, a vertex, or a side,
- each side belongs to at most two subdomains K.

In this case $K \in \mathcal{T}$ is called an *element*, the vertices of $\mathcal{T}$ are the *nodes* of $\mathcal{T}$.

Remark 2.1 We assume that the initial triangulation is always regular, and the so called hanging nodes can only be introduced by subdividing elements.

Nodes that are introduced by subdivision in the interior of a side are called *irregular* or *hanging* or *constrained* nodes, otherwise regular nodes or simply nodes. A triangulation containing only regular nodes is called a regular triangulation.

Remark 2.2 Many practitioners question the usefulness of hanging nodes. We allow them for flexibility and efficiency. For example, on typical geometric triangulations used for the resolution of corner singularities (see, e.g., Figures 1 and 2) the number of DOF per element is $p^2/2 + O(p)$ in both cases (if the serendipity- or "trunk"-space $\mathcal{Q}'_p$ is used on quadrilaterals), but hanging nodes allow geometric subdivisions with considerably fewer elements.

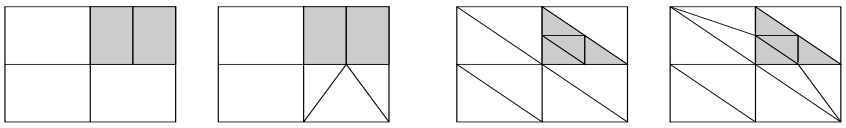


Figure 1: Irregular and regular subdivision of a quadrilateral and a triangle

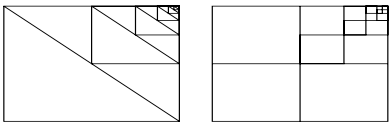


Figure 2: Subdivision into a corner.

2.2 Subdivisions

2.2.1 Motivation

In many practical situations it is necessary to subdivide the FE triangulation to account for singularities or boundary layers. In the case of boundary layers it is additionally desirable to have the ability of anisotropic subdivisions. This is incorporated in the algorithm that subdivides quadrilaterals. In the following we describe the algorithms that subdivides elements, i.e. in the case of a triangular element we describe the $h4$-subdivision and in the case of a quadrilateral element the $h2$-subdivision. The capabilities of these algorithms are visualized in Figure 3.

2.2.2 $h4$-subdivision

The $h4$-subdivision procedure generates 4 new triangles out of a triangle. This is done by introducing new edges that intersect the edges of the original triangle in the middle, as shown in Figure 4. In particular, this algorithm allows a successive subdivision of triangles into a corner as shown in Figure 3. In general, we have to assure that the irregularity rule for the mesh is satisfied. This means, that an algorithm has to determine which elements need to be subdivided first before the element in question can be subdivided. The algorithm that enforces the irregularity rule for triangular and quadrilateral elements is described in section 2.3.

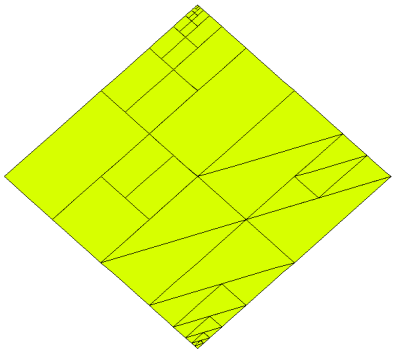
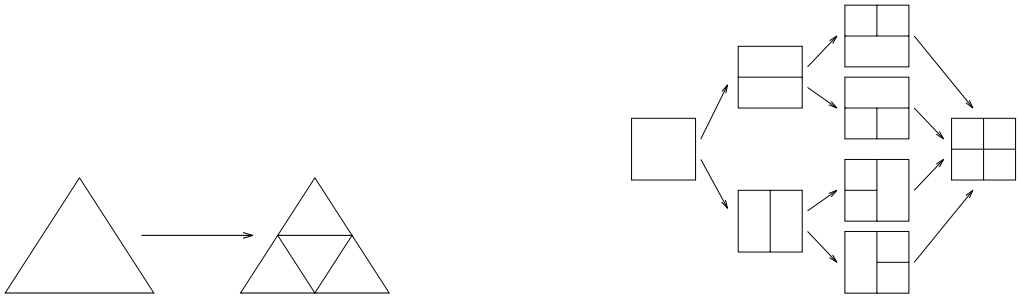


Figure 3: FE Triangulation with constrained and double constrained nodes.



(a) $h4$-subdivision (b) $h2$-subdivisions

Figure 4: Subdivisions of triangles and quadrilaterals

2.2.3 Anisotropic $h2$-subdivision

The anisotropic $h2$-subdivision creates 2 new rectangles out of a rectangle. Both horizontal and vertical subdivisions are possible, as shown in Figure 4. From Figure 4 it is also evident that a successive application of the anisotropic $h2$-subdivision gives a result similar to an $h4$-subdivision. We emphasize again, that the anisotropic $h2$-subdivision procedure allows for both uniform and anisotropic subdivisions and also for subdivisions towards a corner, as shown in Figure 3. Again, the algorithm in section 2.3 determines any necessary additional subdivisions.

2.3 Subdivision algorithm

The algorithm that decides which elements have to be subdivided first is based on an algorithm proposed in [9, 23] for quadrilateral elements, which is also path independent. The algorithm presented here is an extension of the original version in [9] to quadrilateral and triangular elements. The input data that have to be provided are the element `Nel` and the kind of subdivision `Nref`, and the subdivision of element `Nel` is then performed by following the following guidelines:

```
set nel = Nel; nref = Nref
10 continue
if nel is a triangle then
   if nel is constrained then store the element on the waiting list; identify the
      element to be subdivided first and the kind of subdivision required, i.e. set new
      nel, nref; goto 20
   endif
   h4-subdivision of triangle nel
   get next element from shelf, i.e. set new nel,nref
   goto 10
elseif nel is a quadrilateral then
   if nel is an initial mesh element then goto 20
   if kind of subdivision of the father of nel = nref then
      if all nodes of the element are active then goto 20
      else store the element on the waiting list; identify the element to be subdivided
         first and the kind of subdivision required, i.e. set new nel, nref; goto 10
```

endif
else
if all nodes of the element's father are active then goto 20
else store the element on the waiting list; identify the element to be subdivided
first and the kind of subdivision required, i.e. set new nel, nref; goto 10
endif
endif
h2-subdivision of quadrilateral nel
endif
20 if the waiting list is empty then stop
else get the last element from the waiting list, i.e. take new nel and nref
from the list; goto 10
endif

2.4 Polynomial spaces on the master elements

In the following we define the master elements $\hat{K}_t$ and $\hat{K}_q$ that correspond to a triangular or quadrilateral element $K \in \mathcal{T}$.

For each element $K \in \mathcal{T}$ we define the element polynomial degree p_K by

$$p_K = \left(p_K^1, \cdots, p_K^L\right), \tag{2.1}$$

with $L = 4$ for a triangle and $L = 5$ for a quadrilateral. $p_K^i,\ 1 \le i \le L-1$, is the polynomial degree on the i-th edge of the element, and p_K^L the polynomial degree in the interior of the element. For a quadrilateral, we allow p_K^5 to be anisotropic, i.e.

$$p_K^5 = p_K^h * 10 + p_K^v, \tag{2.2}$$

where p_K^h, p_K^v are the horizontal and vertical polynomial degrees in the quadrilateral. Whenever we refer in the following to p_K^5, then we implicitly mean the horizontal and vertical polynomial degrees and not the in (2.2) decoded value.

The polynomial degree of an element can be in principle arbitrary, but we require that

$$p_K^L \ge \max\left\{p_K^1, \cdots, p_K^{L-1}\right\} \quad \forall\ K \in \mathcal{T}. \tag{2.3}$$

We describe a particular set of nodal shape functions for the triangular and quadrilateral master elements that are associated with these master elements but want to emphasize that the user can replace these shape functions by any other set of nodal based shape functions. By slightly modifying the code it is also possible to incorporate shape functions such as integrated Legendre polynomials [31] that are not nodal based.

Also, we want to point out that the choice of the higher order shape functions is somewhat arbitrary, provided that the following conditions are satisfied:

- The mid-side node side shape functions corresponding to the i-th side of $\hat{K}$ vanish on the remaining sides. They must span polynomials of order $p_{\hat{K}}^i$ and vanish on the side endpoints.
- The middle node interior shape functions vanish along the whole element boundary.
- The element shape functions span polynomials of order p, where $p = \min\{p_{\hat{K}}^1, \cdots, p_{\hat{K}}^L\}$.

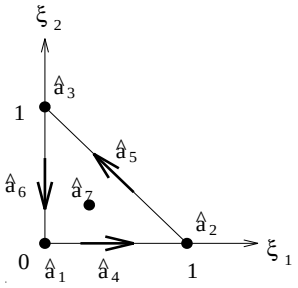


Figure 5: Triangular master element $\hat{K}_t$ of order $p = (p^1, p^2, p^3, p^4)$.

2.4.1 The triangular master element

The triangular master element is defined on the basis of the right triangle $\hat{K}_t$ shown in Figure 5. The nodes $\hat{a}_1, \cdots, \hat{a}_7$ are introduced to represent the dof corresponding to the master element. We divide the nodes into three categories:

1. vertex nodes $\hat{a}_1, \hat{a}_2, \hat{a}_3$
2. mid-side nodes $\hat{a}_4, \hat{a}_5, \hat{a}_6$
3. middle node $\hat{a}_7$

Each of the mid-side nodes and the middle node may have a different corresponding order of approximation $p^1, \cdots, p^4$ respectively. The vertex nodes are linear and therefore have approximation order 1. The shape functions corresponding to the triangular master element are defined for each node by using the area coordinates $\lambda_1, \lambda_2, \lambda_3$ with

$$\lambda_1 = 1 - \xi_1 - \xi_2, \quad \lambda_2 = \xi_1, \quad \lambda_3 = \xi_2. \tag{2.4}$$

Further, a set of points $x^i_j,\ j = 0, \cdots, p^i$, is introduced for each approximation order $p^i,\ i = 1, \cdots, 4$. It is required that $x^i_0 = 0$ and $x^i_{p^i} = 1$ and that $x^i_j < x^i_{j+1}$. It is further understood that the points x^2_j are scaled by a factor $\sqrt{2}$ to account for the length of the second edge of the master triangle. Any system of nodes satisfying above requirements can be choosen to define the corresponding Lagrange type shape functions. Possible choices for x^i_j include equidistant points or the Gauss-Lobatto points. In general the Gauss-Lobatto points are better conditioned than the equidistant points. The Lagrange type shape functions $\hat{\chi}(\xi_1, \xi_2)$ for the master triangle are then defined for each node by (compare [8])

- vertex nodes
$$\hat{\chi}_j(\xi_1, \xi_2) = \lambda_j, \quad j = 1, 2, 3, \tag{2.5}$$
- mid-side nodes
$$\hat{\chi}^1_j(\xi_1, \xi_2) = \frac{\prod\limits_{l=0; l\neq j}^{p^1-1} (\lambda_2 - x^1_l)\lambda_1}{\prod\limits_{l=0; l\neq j}^{p^1-1} (x^1_j - x^1_l)(1 - x^1_j)}, \quad j = 1, \cdots, p^1 - 1, \tag{2.6}$$

  the formulas for $\hat{\chi}^2_j(\xi_1, \xi_2)$ and $\hat{\chi}^3_j(\xi_1, \xi_2)$ are obtained by permuting indices

p	vertex dof	side dof	interior dof	total # dof
1	3	0	0	3
2	3	3	0	6
3	3	6	1	10
4	3	9	3	15
5	3	12	6	21
6	3	15	10	28
7	3	18	15	36
8	3	21	21	45
9	3	24	28	55

Table 1: Number of triangular element shape functions for uniform p.

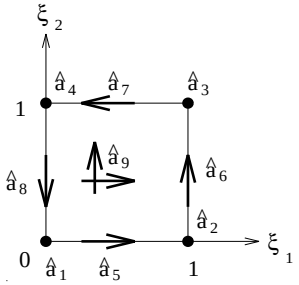


Figure 6: Quadrilateral master element $\hat{K}_q$ of order $p = (p^1, p^2, p^3, p^4, p^5)$.

- middle node

$$\hat{\chi}^4_{i,j,k}(\xi_1, \xi_2) = \frac{\prod_{m=0}^{i-1}(\lambda_1 - x_m^4)\prod_{n=0}^{j-1}(\lambda_2 - x_n^4)\prod_{l=0}^{k-1}(\lambda_3 - x_l^4)}{\prod_{m=0}^{i-1}(1 - x_j^4 - x_k^4 - x_m^4)\prod_{n=0}^{j-1}(x_j^4 - x_n^4)\prod_{l=0}^{k-1}(x_k^4 - x_l^4)}, \tag{2.7}$$

with $\quad 1 \le i, j, k \le p^4 - 1;\ i + j + k = p^4.$

There are $(p^4 - 1) \times (p^4 - 2)/2$ shape functions associated with the middle node. Table 1 summarizes the number of shape functions or the number of dof for the triangular master element with uniform p. In general the number of shape functions is determined by

$$\#\text{dof} = p^1 + p^2 + p^3 + \frac{(p^4 - 1)(p^4 - 2)}{2}. \tag{2.8}$$

2.4.2 The quadrilateral master element

The quadrilateral master element is defined on the basis of the unit square $\hat{K}_q$ shown in Figure 6. The nodes $\hat{a}_1, \cdots, \hat{a}_9$ represent the dof corresponding to the quadrilateral master element and are again devided into three categories:

1. vertex nodes $\hat{a}_1, \hat{a}_2, \hat{a}_3, \hat{a}_4$
2. mid-side nodes $\hat{a}_5, \hat{a}_6, \hat{a}_7, \hat{a}_8$

3. middle node $\hat{a}_9$

Similar to the triangular master element can each mid-side node and the middle node have a different corresponding approximation order $p^1, \cdots, p^5$ respectively. Additionally, we allow the approximation order p^5 of the middle node to vary horizontally and vertically. This is indicated in Figure 6.

The points $x_j^i,\ j = 0, \cdots, p^i;\ i = 1, 2, 3, 4$ are defined as in Section 2.4.1 and the Lagrange type master element shape functions $\hat{\chi}(\xi_1, \xi_2)$ are defined as tensor products of one-dimensional shape functions. These 1D shape functions of order p corresponding to a system of points x_i are then defined by

- vertex nodes
$$\hat{\chi}_1(\xi) = 1 - \xi, \quad \hat{\chi}_2(\xi) = \xi \tag{2.9}$$
- middle node
$$\hat{\chi}_{j+2}(\xi) = \prod_{l=1; l\neq j}^{p-1} \frac{\xi - x_l}{x_j - x_l} \times \frac{\xi(1-\xi)}{x_j(1-x_j)}, \quad j = 1, \cdots, p-1. \tag{2.10}$$

The tensor product gives then for the quadrilateral master element shape functions

- vertex nodes
$$\hat{\chi}_k(\xi_1, \xi_2) = \hat{\chi}_i(\xi_1)\hat{\chi}_j(\xi_2), \quad 1 \leq i, j \leq 2 \tag{2.11}$$
- mid-side nodes
$$\hat{\chi}_k^1(\xi_1, \xi_2) = \hat{\chi}_{j+2}(\xi_1)\hat{\chi}_1(\xi_2), \quad j = 1, \cdots, p^1 - 1 \tag{2.12}$$
with formulas for $\hat{\chi}_k^2(\xi_1, \xi_2), \hat{\chi}_k^3(\xi_1, \xi_2), \hat{\chi}_k^4(\xi_1, \xi_2)$ obtained by permuting indices
- middle node
$$\hat{\chi}_k(\xi_1, \xi_2) = \hat{\chi}_{i+2}(\xi_1)\hat{\chi}_{j+2}(\xi_2), \quad i = 1, \cdots, p_h^5 - 1,\ j = 1, \cdots, p_v^5 - 1,\ p^5 = p_h^5 \times 10 + p_v^5. \tag{2.13}$$

In general the number of shape functions or dof of the quadrilateral is given by

$$\#\text{dof} = p^1 + p^2 + p^3 + p^4 + (p_h^5 - 1) * (p_v^5 - 1). \tag{2.14}$$

The number of shape functions is summarized in Table 2 for uniform p.

2.4.3 Interpretation of the dof

The definition of the corresponding dof is somewhat arbitrary and relies on a particular, corresponding interpolation procedure [8]. We will now shortly introduce the interpolation procedure that is currently used and emphasize that this particular procedure can easily be modified or removed to incorporate another interpolation procedure.

Let K be any element. For $u \in C^0(\overline{K})$ we define the corresponding interpolant u_{hp} in three steps:

1. Evaluate the usual bilinear interpolant u_{hp}^1 of u using the bilinear vertex shape functions
$$u_{hp}^1 = \sum_{i=1}^{N} u(a_i)\hat{\chi}_i \tag{2.15}$$
with $N = 3$ for a triangle (and $N = 4$ for a quadrilateral).

p	vertex dof	side dof	interior dof	total # dof
1	4	0	0	4
2	4	4	1	9
3	4	8	4	16
4	4	12	9	25
5	4	16	16	36
6	4	20	25	49
7	4	24	36	64
8	4	28	49	81
9	4	32	64	100

Table 2: Number of quadrilateral element shape functions for uniform p.

2. Subtract u^1_{hp} from u and for each of the sides evaluate the corresponding side interpolants of $u - u^1_{hp}$ using the mid-side node shape functions

$$u^2_{hp} = \sum_{i=1}^{N} \sum_{j=1}^{p^i-1} (u - u^1_{hp})(\hat{a}_{i,j})\hat{\chi}_{i,j} \tag{2.16}$$

where $\hat{a}_{i,j}$ are the points on the i-th side that are used to construct the side shape functions.

3. Evaluate the Lagrange interpolant of $u - u^1_{hp} - u^2_{hp}$ using the points, that are used to construct the shape functions corresponding to the middle node

$$u^3_{hp} = \sum_{i,j,k=1; i+j+k=p^{N+1}}^{p^{N+1}-1} (u - u^1_{hp} - u^2_{hp})(\hat{a}_{i,j,k})\hat{\chi}_{N+1,i,j,k} \tag{2.17}$$

The Lagrange interpolant u_{hp} of u is then defined as

$$u_{hp} = u^1_{hp} + u^2_{hp} + u^3_{hp}. \tag{2.18}$$

This interpolation procedure is consistent with the hp convergence analysis in the sense that when combined with the standard Cea's lemma argument, it yields optimal rates of convergence. The definition of the hp-interpolation differs in the evaluation of u^2_{hp} and u^3_{hp} because they are obtained by performing local H^1_0-projections along the element sides or over the element, respectively. We emphasize here that these local H^1_0-projections should be used if the Lagrange type shape functions are based on equidistant points. On the other hand, if the Gauss-Lobatto points are used in Section 2.4.1 and 2.4.2 then the optimal convergence rates can still be obtained by avoiding the local H^1_0-projections. In any way, the particular interpolation procedure implies the definition of specific dof. These are functionals $\hat{\phi}_i$ that constitute a dual basis to the element shape functions.

2.5 The hp-FE meshes

We emphasize that the actual computations are not performed on the hp-meshes, which we define in the following. The actual computations are performed on the master elements and therefore we first have to introduce an element mapping that maps from the hp-mesh onto the corresponding master element.

The element mapping F_K, $K \in \mathcal{T}$, is defined by

$$F_K : \hat{K} \to K, \tag{2.19}$$

which is assumed to be a C^1 diffeomorphism. Furthermore, we assume that

1. $\det F_K' \geq \alpha > 0 \; \forall \; K \in \mathcal{T}$.
2. The part of an edge that neighboring elements $K_i, K_j \in \mathcal{T}$ have in common does have the same parametrization "from both sides". Let $\gamma_{ij} = \overline{K_i} \cap \overline{K_j}$ be the common part of the edge with possibly irregular endpoints (vertices) P_1, P_2. Then for any point $P \in \gamma_{ij}$, we have

$$\operatorname{dist}\left(F_{K_i}^{-1}(P), F_{K_i}^{-1}(P_l)\right)/L_i = \operatorname{dist}\left(F_{K_j}^{-1}(P), F_{K_j}^{-1}(P_l)\right)/L_j, \quad l = 1, 2,$$

   where L_i, L_j denote the length of the edges of the reference elements corresponding to γ_{ij}.

Remark 2.3 Above assumptions are also applicable in the situation of neighboring triangular and quadrilateral elements. Further, condition 2 is essential for the construction of conforming (continuous) finite element spaces.

Additionally, we also require that the polynomial degree is the same across an edge of two neighboring elements. This requirement is essential in the construction of continuous finite element spaces. The element mappings F_K and the element polynomial degrees p_K, defined in (2.1), are collected in the degree vector $\boldsymbol{p}$ and the mapping vector $\boldsymbol{F}$ by

$$\boldsymbol{p} := \{p_K \; : \; K \in \mathcal{T}\} \; , \qquad \boldsymbol{F} := \{F_K \; : \; K \in \mathcal{T}\} \, . \tag{2.20}$$

The triangulation $\mathcal{T}$ together with the approximation order on each element in $\mathcal{T}$, given by the degree vector $\boldsymbol{p}$, and the mapping vector $\boldsymbol{F}$, define the *hp*-**mesh** $\mathcal{M}$ and we shortly write

$$\mathcal{M} := (\mathcal{T}, \boldsymbol{p}, \boldsymbol{F}) \, . \tag{2.21}$$

In the case of a uniform polynomial degree p throughout the mesh $\mathcal{M}$ we simply replace $\boldsymbol{p}$ by p.

Remark 2.4 The above definitions are applicable to L^2 and C^0 finite element formulations and can be naturally extended to vector valued problems. But, in the case of mixed formulations, these definitions have to be changed to account for a different interpretation of the degrees of freedom, as it is for example the case with the generalization of the edge elements [6] that are described in [11].

3 Examples of variational formulations and the construction of corresponding *hp*-FE Spaces

In the following we give examples of problems of engineering interest that can be discretized using the *hp*-FE spaces that we define in this section. We state the weak formulations of our examples in the most general sense and do not discuss the stability issue in detail here. We only point out that the trial and test spaces have to be chosen appropriately, i.e. restricted, so that the formulation is stable. Such appropriate FE spaces are given in the remainder of this section, which will indeed lead to stable approximations.

3.1 Example problems and their variational formulations

1. The linearized **elasticity** problem (displacement formulation), see e.g. [2], has the following weak formulation: Let $\Omega \subset I\!R^2$ be a Lipschitz domain and let λ, μ be the Lamé constants and $\boldsymbol{\epsilon}_{ij}$ be the linearized strain tensor, then:

$$\begin{aligned} &\text{Find } \boldsymbol{v} \in H_0^1(\Omega) \text{ such that} \\ &\int_\Omega \lambda \operatorname{div}\boldsymbol{u} \operatorname{div}\boldsymbol{v} + 2\mu \sum_{i,j=1}^{2} \boldsymbol{\epsilon}_{ij}(\boldsymbol{u})\boldsymbol{\epsilon}_{ij}(\boldsymbol{v})\, \partial\Omega = \int_\Omega \boldsymbol{f}\cdot\boldsymbol{v}\, \partial\Omega \quad \forall\, \boldsymbol{v} \in H_0^1(\Omega) \end{aligned} \tag{3.1}$$

2. Scalar elliptic equation, e.g. the **exterior Helmholtz** problem: Let $\Omega \subset I\!R^3$ be the exterior of the unit sphere S and

$$\begin{aligned} \Delta u - k^2 u &= 0 && \text{in } \Omega, \\ \frac{\partial u}{\partial n} &= g && \text{on } S, \\ \left| \frac{\partial u}{\partial R} - iku \right| &= O\left(\frac{1}{R^2}\right) && \text{for } R \to \infty. \end{aligned} \tag{3.2}$$

   The corresponding weak formulation is:

$$\begin{aligned} &\text{Find } u \in H^+_{1,\omega}(\Omega) \text{ such that} \\ &\int_\Omega \boldsymbol{\nabla} u \cdot \boldsymbol{\nabla}\overline{v} - k^2 u\overline{v}\, d\Omega = \int_{\partial\Omega} g\overline{v}\, dS \quad \forall\, v \in H^+_{1,\omega^*}(\Omega), \end{aligned} \tag{3.3}$$

   where $H^+_{1,\omega}$ are weighted spaces of "outgoing waves". Although the exterior Helmholtz problem is a three-dimensional problem, it actually can be solved within the framework of this FE package by a hierarchical modeling approach and a tensor product Ansatz for the trial and test spaces, for details and the definition of $H^+_{1,\omega}$ we refer to [12, 13, 14].

3. The **Stokes** problem: Let $\Omega \subset I\!R^n$, $n = 2$, and

$$\begin{aligned} -\nu\boldsymbol{\Delta}\boldsymbol{u} + \boldsymbol{\nabla} p &= \boldsymbol{f} && \text{in } \Omega, \\ \operatorname{div}\boldsymbol{u} &= 0 && \text{in } \Omega, \\ \boldsymbol{u} &= 0 && \text{on } \partial\Omega. \end{aligned} \tag{3.4}$$

   The corresponding weak formulation is:

$$\begin{aligned} &\text{Find } (\boldsymbol{u}, p) \in \left[H_0^1(\Omega)\right]^n \times L_0^2(\Omega) \text{ such that} \\ &\int_\Omega \nu\boldsymbol{\nabla}\boldsymbol{u}\cdot\boldsymbol{\nabla}\boldsymbol{v}\, d\Omega - \int_\Omega p \operatorname{div}\boldsymbol{v}\, d\Omega = \int_\Omega \boldsymbol{f}\cdot\boldsymbol{v}\, d\Omega \quad \forall\, \boldsymbol{v} \in \left[H_0^1(\Omega)\right]^n, \\ &\int_\Omega q \operatorname{div}\boldsymbol{u}\, d\Omega = 0 \qquad\qquad \forall\, q \in L_0^2(\Omega). \end{aligned} \tag{3.5}$$

   The Stokes problem has a stable weak formulation if $S_0^{1,n}(\boldsymbol{p}, \mathcal{T}; \Omega) \times S_0^0(\boldsymbol{p} - \boldsymbol{2}, \mathcal{T}; \Omega)$ is chosen for the trial and test spaces. In this case the triangulation $\mathcal{T}$ can be either a regular mesh with triangles and quadrilaterals [27, 29] or an irregular quadrilateral mesh, i.e. a quadrilateral mesh with local geometric refinements and hanging nodes [25]. The above pairs of spaces are stable on these *hp*-meshes, i.e. satisfy the inf-sup condition.

4. The **electromagnetics** problem (Maxwell equations) can be written under appropriate assumptions as a (reduced) wave equation [11, 18]:

$$\begin{aligned} curl\left(\frac{1}{\mu} curl\boldsymbol{E}\right) - (\omega\epsilon - j\omega\sigma)\,\boldsymbol{E} &= -j\omega\boldsymbol{J}^{imp} \\ div\left((j\omega\epsilon + \sigma)\,\boldsymbol{E}\right) &= 0 \end{aligned} \tag{3.6}$$

The weak formulation is then

$$\begin{aligned}
&\text{Find } (\boldsymbol{E}, p) \in H(curl;\Omega) \times H^1(\Omega) \text{ such that} \\
&\int_\Omega \frac{1}{\mu}\,(curl\boldsymbol{E}) \cdot \left(curl\overline{\boldsymbol{F}}\right)\, d\Omega \;-\; \int_\Omega \left(\omega^2\epsilon - j\omega\sigma\right)(\boldsymbol{E} + \boldsymbol{\nabla} p) \cdot \overline{\boldsymbol{F}}\, d\Omega \\
&\qquad\qquad = \; -j\omega \int_\Omega \left(J^{imp} \cdot \overline{\boldsymbol{F}}\right)\, d\Omega \quad \forall\, \boldsymbol{F} \in H(curl;\Omega), \\
&\int_\Omega \left(\omega^2\epsilon - j\omega\sigma\right) \boldsymbol{E} \cdot \boldsymbol{\nabla}\overline{q}\, d\Omega \;=\; 0 \quad \forall\, q \in H^1(\Omega).
\end{aligned} \tag{3.7}$$

In [11] it is shown that the above weak formulation is stable in the case of lossless media, i.e. $\sigma = 0$.

5. The **Poisson** problem $-\boldsymbol{\Delta} u + f = 0$ can be written formally in a mixed formulation as

$$\begin{aligned}
\boldsymbol{\nabla} u &= \sigma, \\
f + div\ \sigma &= 0.
\end{aligned} \tag{3.8}$$

A stable mixed formulation [5, 6] is

$$\begin{aligned}
&\text{Find } (\sigma, u) \in H(div;\Omega) \times L^2(\Omega) \text{ such that} \\
&\int_\Omega \sigma \cdot \tau\, d\Omega + \int_\Omega u\ \mathrm{div}\ \tau\, d\Omega \;=\; 0 \qquad\qquad \forall\, \tau \in H(div;\Omega), \\
&\qquad\qquad \int_\Omega v\ \mathrm{div}\ \sigma\, d\Omega \;=\; -\int_\Omega f\, v\, d\Omega \quad \forall\, v \in L^2(\Omega).
\end{aligned} \tag{3.9}$$

3.2 The *hp*-FE spaces

Now we give a precise mathematical definition of the *hp*-FE spaces that are admissible in our code, i.e. we define the global *hp* spaces that are admissible within the framework of the code. Among those are the traditional H^1-conforming (scalar- or vector-valued) piecewise polynomials, and various $H(div)$ and $H(curl)$ conforming spaces of piecewise polynomials, where the *curl*-operator [16] is defined by

$$curl\ \boldsymbol{v} = \begin{cases} \dfrac{\partial v_2}{\partial x_1} - \dfrac{\partial v_1}{\partial x_2} & n = 2 \\ \left(\dfrac{\partial v_3}{\partial x_2} - \dfrac{\partial v_2}{\partial x_3}, \dfrac{\partial v_1}{\partial x_3} - \dfrac{\partial v_3}{\partial x_1}, \dfrac{\partial v_2}{\partial x_1} - \dfrac{\partial v_1}{\partial x_2}\right) & n = 3 \end{cases} \tag{3.10}$$

The framework of the code is general and flexible enough to include most settings of engineering interest.

The condition (2.3) makes it possible to define the *hp*-spaces independent of a particular set of shape functions, which allows us to view the element definition in the abstract setting of Ciarlet [2].

Definition 3.1 *Let $\mathcal{M} = (\mathcal{T}, \boldsymbol{p})$ be a given hp-mesh on $\Omega \subset \mathbb{R}^n$, $n = 2$ or 3, and let $m \geq 1$ be the number of field variables. Further, let $S_p(K)$ be the polynomial space of degree p on the element K which is either $\mathcal{P}_p(K)$ if K is a triangle or $\mathcal{Q}_p(K)$ if K is a quadrilateral. Let $e_i, 1 \leq i \leq L-1$, be the edges of the element K. Then we define*

1. *the space of piecewise discontinuous vector-valued polynomials*

$$S^{0,m}(\mathcal{M};\Omega) := \left\{\boldsymbol{u};\ \boldsymbol{u} \in \left[L^2(\Omega)\right]^m;\ \boldsymbol{u}|_K \in \left[S_{p_K^L}(K)\right]^m;\ \boldsymbol{u}|_{e_i} \in \left[S_{p_K^i}(e_i)\right]^m \quad \forall\, K \in \mathcal{T}\right\} \tag{3.11}$$

2. *the space of piecewise discontinuous vector valued-polynomials with vanishing mean value*

$$S_0^{0,m}(\mathcal{M};\Omega) := \left\{ \boldsymbol{u};\ \boldsymbol{u} \in S^{0,m}(\mathcal{M};\Omega);\ \int_\Omega \boldsymbol{u}\, d\Omega = \mathbf{0} \right\} \tag{3.12}$$

3. *the space of piecewise continuous vector-valued polynomials*

$$S^{1,m}(\mathcal{M};\Omega) := \left\{ \boldsymbol{u};\ \boldsymbol{u} \in \left[H^1(\Omega)\right]^m;\ \boldsymbol{u}|_K \in \left[S_{p_K^L}(K)\right]^m;\ \boldsymbol{u}|_{e_i} \in \left[S_{p_K^i}(e_i)\right]^m \quad \forall\ K \in \mathcal{T} \right\} \tag{3.13}$$

4. *the space of piecewise continuous vector-valued polynomials with vanishing trace on the boundary*

$$S_0^{1,m}(\mathcal{M};\Omega) := \left\{ \boldsymbol{u};\ \boldsymbol{u} \in S^{1,m}(\mathcal{M};\Omega);\ \boldsymbol{u}|_{\partial\Omega} = \mathbf{0} \right\} \tag{3.14}$$

5. *the space of $H(div)$-conforming vector-valued piecewise polynomials*

$$\begin{aligned} S^{div}(\mathcal{M};\Omega) \ :=\ & \left\{ \boldsymbol{u};\ \boldsymbol{u} \in \left[L^2(\Omega)\right]^m;\ div\, \boldsymbol{u} \in L^2(\Omega);\ \boldsymbol{u}|_K \in [S_{p_K}(K)]^m; \right. \\ & \left. \boldsymbol{u}|_{e_i} \in \left[S_{p_K^i}(e_i)\right]^m \quad \forall\ K \in \mathcal{T} \right\} \end{aligned} \tag{3.15}$$

6. *the space of $H(curl)$-conforming vector-valued piecewise polynomials*

$$\begin{aligned} S^{curl}(\mathcal{M};\Omega) \ :=\ & \left\{ \boldsymbol{u};\ \boldsymbol{u} \in \left[L^2(\Omega)\right]^m;\ curl\, \boldsymbol{u} \in \left[L^2(\Omega)\right]^m;\ \boldsymbol{u}|_K \in [S_{p_K}(K)]^m; \right. \\ & \left. \boldsymbol{u}|_{e_i} \in \left[S_{p_K^i}(e_i)\right]^m \quad \forall\ K \in \mathcal{T} \right\} \end{aligned} \tag{3.16}$$

Remark 3.2 Frequently we include the degree vector in the notation, i.e. we write $S^0(\boldsymbol{p},\mathcal{T};\Omega)$ instead of $S^0(\mathcal{M};\Omega)$. In the case of a uniform polynomial degree we simply write $S^0(p,\mathcal{T};\Omega)$.

Remark 3.3 For a vector valued problem it is admissible to use any combination of these *hp*-FE spaces, with possibly different $\boldsymbol{p}$, but on the same $\mathcal{T}$.

Remark 3.4 The BDM-spaces and $BDFM$-spaces defined e.g. in [6] fall into the category of spaces that approximate $S^{div}(\mathcal{M};\Omega)$. Further, the edge elements by Nedelec [20, 21] fall into the category of elements that approximate $S^{curl}(\mathcal{M};\Omega)$.

4 Overview of the FE package

The whole FE package consists of several independent modules. The interfaces between the various packages shown in Figure 7 are well defined and consistent with the underlying data structure that is used (compare section 5). In the following we shortly describe the various parts of the code.

Geometric modeling package (GMP): The GMP [7] provides parametrizations which describe the domain that has to be meshed by the mesh generator. The parametrizations currently possible include such reference figures as curved triangles and quadrilaterals and also 2-dimensional linear manifolds in the 3-dimensional space that are bounded by 3 or 4 curves. It is easily possible to add additional parametrizations to the GMP if the existing ones cannot describe the domain Ω of interest. Therefore, Ω is a 2-dimensional linear manifold that is a

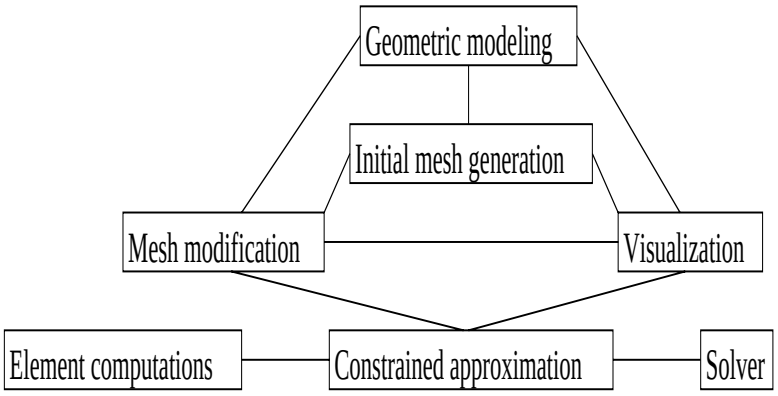


Figure 7: Structure of 2D-*hp*-adaptive FE package.

union of a finite number of triangular and quadrilateral linear manifolds that have edges in common.

Initial mesh generation (IMG): The IMG creates the triangular/quadrilateral finite elements in each triangular/quadrilateral reference figure. The initial mesh is generated by using the idea of an algebraic mesh generator and *hp*-interpolation. Given, for the reference figure, numbers m and n of divisions in the "horizontal" and "vertical" directions (compatible for neighboring reference figures, the initial mesh is always regular) the reference figures are covered with uniform, regular grids consisting of finite elements $\tilde{K}$. By constructing a composition of the standard affine map $\boldsymbol{\eta}$ transforming the master element $\hat{K}$ onto element $\tilde{K}$, and (the restriction of) the block parametrization $\boldsymbol{x}_b$, a map is constructed from the master element $\hat{K}$ onto a curvilinear element K, identified as the image of element $\tilde{K}$ under the particular parametrization $\boldsymbol{x}_b$,

$$K = \boldsymbol{x}(\hat{K}) = \boldsymbol{x}_b(\tilde{K}), \qquad \boldsymbol{x} = \boldsymbol{x}_b \circ \boldsymbol{\eta}. \tag{4.1}$$

In principle, this map could be used directly to define the curvilinear element, i.e. in the element calculations. In practice, it is approximated using the idea of *isoparametric* approximation. More precisely, given a particular order of approximation for element K (may vary for different nodes), transformation $\boldsymbol{x}$ is replaced with its *hp-interpolation*.

The idea of the *hp*-interpolation follows from the convergence theory for *hp*-approximations and has been introduced in [10]. Roughly speaking, the *hp*-interpolation combines the classical interpolation for vertex nodes with local H_0^1-projections for higher order nodes. Given a sufficiently regular function, the corresponding *hp*-interpolant exhibits the same orders of convergence (in terms of both h and p) as the corresponding global H_0^1-projection (solution to the Laplace equation with Dirichlet boundary conditions imposed using the H_0^1-projection on the boundary).

The initial *hp*-FE mesh can have polynomial degree p with $1 \leq p \leq 9$. This limitation on p can easily be removed by providing even higher order shape functions and the necessary storage space. All data corresponding to the *hp*-FE mesh are stored in the abstract data structure that is introduced in section 5.

Mesh modification (MM): The MM package handles the various types of mesh refinement procedures. It is possible to increase the polynomial degree of an element (p-refinement) or to refine a triangle into 4 triangles ($h4$-refinement) or to refine a quadrilateral into 2 quadrilaterals ($h2$-refinement). In the case of h-refinement we restrict ourselves to 1-irregular meshes and further allow double constrained vertex nodes in quadrilateral elements. A vertex node is called irregular or constrained or hanging if there exists an adjacent element for which the node is not a vertex node. A quadrilateral contains a double constrained node if the node

is constrained by nodes of which at least one is irregular. This will be dealt with in more detail in section 6. Figure 3 shows a mesh with constrained and double constrained nodes. In the following sections we describe in detail the implementation aspects related to the mesh refinement, constrained approximation and element computations.

Element computations (EC): The calculation of the element stiffness matrix and the element load vector are done on the master elements that are introduced in section 2.4. The physical element is therefore mapped in the usual way onto the corresponding master element. The integration is performed by numerical integration and the Gaussian quadrature is applied although other choices of integration procedures are possible too. It is assumed that the user of the FE package provides the routines that perform the element computations and therefore it is up to the user to decide on which integration procedure is used.

Constrained approximation (CA): The CA enforces the continuity of the approximation on irregular meshes. This is done by requiring a common order of approximation along two neighboring elements and by constraining degrees of freedom (dof) that correspond to an irregular node. This is described in detail in section 6.

Solver: The Solver interfaces with the CA. The corresponding element stiffness matrices and load vectors are transformed into an appropriate data structure format for the employed linear system solver. Currently a direct solver is used but it is straight forward to interface with any other solver, e.g. with iterative solvers.

Visualization: The currently used graphics package is based on XWindows graphics and displays the corresponding elements which are stored in the data structure. This graphics package could easily be replaced by interfacing with any other graphics/CAD program.

5 The data structure

Similar to computer languages C, C++ it is possible to define abstract data types in Fortran 90. The data that are necessary for an *hp*-adaptive FE implementation can be classified into two categories, nodal information and element information. Therefore, we define the two data types node and element by

```
type node
   character(len=4) :: type
   integer :: order
   double precision, dimension(:,:), pointer :: coord
   double precision, dimension(:,:), pointer :: dofs
endtype node
type element
   character(len=5) :: type
   integer :: nodes(9), neig(4), bcond, father, sons(4), ref_kind
endtype element
```

Type `node` contains all the information about a particular node, where `type`='vert' for a vertex node, ='msid' for a midside node, ='mdlt' for a bubble node in a triangle and ='mdlq' for a bubble node in a quadrilateral. The variable `order` determines the approximation order at the node. It is 1 for the vertex nodes, since the vertex nodes are by definition linear. The degrees of freedom (dof) on the edges and inside the element are handled by higher order nodes. This simplifies dealing with higher approximation orders, because the number of nodes stays the same for cubic or higher approximation orders. The geometry dof corresponding to

the particular node are stored in the variable `coord` and the corresponding solution dof are stored in the variable `dofs`.

In the type `element` the variable `type` determines the type of the element by its value 'trian' or 'quadr'. The node numbers, i.e. pointers onto all the nodal information are stored in the variable `node`, which is an array of length 9. The last two entries of `node` have no meaning for a triangular element. The neighboring elements are stored in variable `neig`. The boundary condition flags are stored in variable `bcond`, which determines which edges or vertices of an element are part of the boundary of the domain Ω. The variables `father`, `sons` and `ref_kind` contain information that is generated during the h-refinement. In detail, `father` points to the father element of the present element, `sons` points to the elements that are generated by h-refining the element and `ref_kind` determines the refinement kind. `ref_kind` can be equal to 1 or 2 for horizontal or vertical $h2$-refinement of a quadrilateral and equal to 4 for $h4$-refinement of a triangle.

The data types `node` and `element` are used to define the dynamic arrays `NODES` and `ELEMS` by the statements

type(node), allocatable :: NODES(:)

type(element), allocatable :: ELEMS(:)

It should be emphasized here that the necessary memory for storing the element and nodal information is allocated on demand during the runtime of the code. Therefore, it is not necessary to reserve large memory space ahead of time. Also, memory can be deallocated at run time if certain elements and nodal information can be deleted due to refinement/unrefinement.

It should be pointed out here that the data types defined above do not incorporate directly a global denumeration of the elements and that they only provide information about equal size or larger size neighboring elements. Therefore, it is necessary to establish an order of the elements and to provide information about the actual active neighboring elements. This is done by two algorithms, *nelcon* and *neigbr*.

An order of the elements is established by an algorithm that determines the so called *natural order of elements*. This algorithm is based on the denumeration of the initial mesh elements and the refinement of the initial mesh element. The element number `nel` of the n-th element is found by the algorithm *nelcon* that has as input data the previous element `Nel`:

set nel = Nel

10 continue

if (nel=0) then nel = 1; go to 20

elseif initial mesh element then nel = nel + 1; go to 20

else get family information

  if nel is not last son then nel = brother(#son+1); go to 20

  else nel = father(nel); go to 10

  endif

endif

20 continue

if element has not been refined then Nel1 = nel; return

else nel = first son; go to 20

endif

For solving a particular problem with the FE code it is necessary to loop over all the active elements. This can easily be done by using the natural order of elements. A typical loop over all the active elements in the code will then look as follows

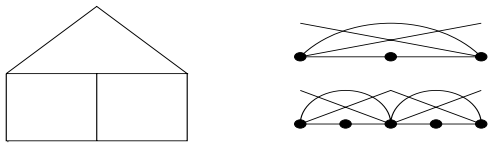


Figure 8: Large element neighboring two small elements.

```
nel = 0
do iel = 1, number of elements
  call nelcon(nel, nel)
      ⋮
enddo
```

The neigbor information is generated in the code by the procedure *neigbr*, which explicitly follows the refinement of each of the elements that are stored in the array `neigh`.

6 Constrained approximation for C^0 discretizations

The h-refinement strategies can lead to irregular meshes, compare Figure 3, and this brings up the issue of enforcing continuity of the numerical approximation. We are assuming in this section that the global shape functions are in H^1, and so continuity must be enforced if it does not occur naturally. We have already required that the polynomial degree across element edges be the same and therefore we only need to deal with the case of having a large element on one side of the edge and two small elements on the other side. This situation and the shape functions corresponding to a quadratic approximation along the common edge are illustrated in Figure 8. From Figure 8 it is evident that similar imcompatibilities will occur for all polynomial degrees when elements with different sizes are adjacent. Nevertheless, the continuity can be accomplished by forcing the son's shape functions to conform to the father's at the constrained nodal points. This, however, means that the dof corresponding to constrained nodes are not dof since they are dependent upon other dof. It is evident that the constraints that have to be determined must be related to properties of one dimensional shape functions, since the restriction of a shape function φ to an edge e results in a one dimensional shape function. This requires that the connectivities of the elements to each other are known. That is, for each constrained local node we need to determine lists of the associated global dof and the so-called constraints coefficients.

In the following we describe how to reconstruct this information at the local level, and show how it is used in the generalized assembling procedure to impose the continuity of the approximation. We emphasize that this procedure is similar for triangles and quadrilaterals and can also be extended to the 3D case, although the situation becomes much more complex in three dimensions.

6.1 Triangular elements

We will begin with a simple example of the construction of the constraint information for a triangle with single-constrained nodes. Figure 9 shows an initially unrefined triangle and the corresponding four sons generated by an $h4$ refinement. The constraint information for the hatched element is generated in three steps: reconstruction of the nodal constraints, identification of the modified element, and reconstruction of the constraints for the local dof. These

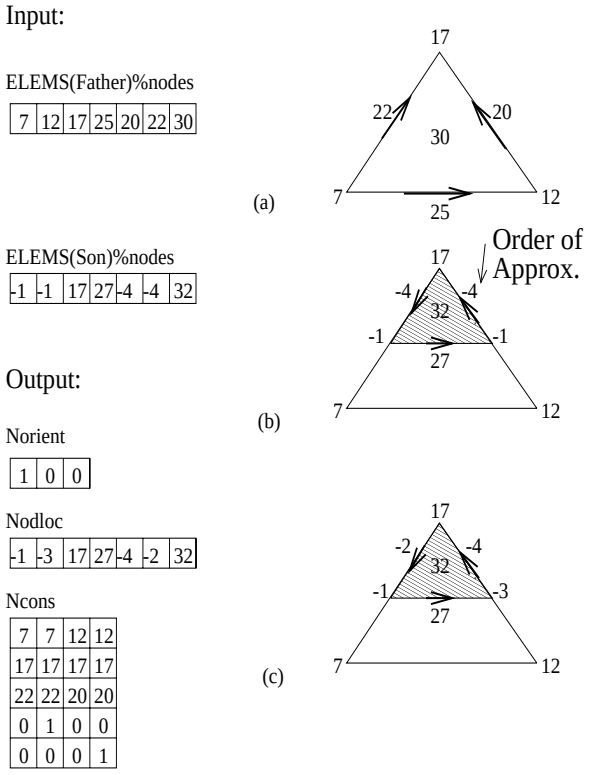


Figure 9: An example of the construction of the nodal constraint information from the data structure arrays for a triangle with single-constrained nodes.

procedures will now be illustrated with an example.

Step1: Reconstruction of the nodal constraints. The nodal information available in the data structure for the initial triangle and the hatched son after $h4$ refinement is shown in the left column of Figure 9. The ELEMS(father)%Nodes and ELEMS(son)%Nodes arrays simply contain a counterclockwise listing of the node numbers of the father and son elements, respectively. By comparing these arrays with their corresponding elements, it is seen that the first three entries are a counterclockwise listing of the vertex nodes, the next three entries are the mid-side nodes, and the last is reserved for the middle node. For the hatched element and its father, then, this is the nodal information that is available in the data structure after the $h4$ refinement.

The four nodes of the triangle in Figure 9b that are marked with a negative sign are constrained nodes, i.e. they do not include active dof. These nodes are not associated with any node numbers in the data structure. Instead, the order of approximation of the node is stored, preceded by a negative sign. Since constrained nodes do not include dof, there is no need to endow them with any explicit node number. Previous experience shows that the mesh modification routines are simplified significantly when nodal constraint information is not stored explicitly in the data structure. Indeed, with this approach there is no need to update nodal arrays for constrained nodes during the refinement/unrefinement processes.

The assembling procedure requires a local listing of node numbers, and although node numbers are not stored for constrained nodes, this information is easily reconstructed element by element. The array Nodloc, shown in Figure 9, contains a listing of the node numbers for the hatched element, using the same pattern as the ELEMS arrays. The four constrained nodes are given a local denumeration, 1-4, again preceded by a negative sign. However, in this

case the numbers are actually pointers to the respective columns in the array Ncons, to be described next.

Due to continuity requirements, the dof of constrained nodes depend on the dof along the father's common edge. Thus, for a given constrained node, a listing of the corresponding constraining nodes along the father's edge must be constructed. This information is stored columnwise in the array Ncons, the first column containing the information for constrained node -1, the second column for node -2, and so on. A given column in Ncons contains the vertex and midside node numbers of the father's common side in the first three entries. For example, the vertex node -1 is constrained by nodes 7, 17, and 22, which are the first three entries in the first column of Ncons. The final two entries in any column of Ncons contain the so-called *integer coordinates*, relating the position of the element side with respect to the father side. Figure 9 shows that these entries are nonzero only for midside nodes. To define the integer coordinates, a local 1D coordinate system from -1 to 1 is constructed along the father's common side, in the direction of the orientation of the father's midside node. Considering the orientation of the hatched element's midside node, the positions of the two corressponding vertex nodes along the hatched element's side are identified as the integer coordinates. For example, the first constrained midside node in Figure 10(a) is -2, which connects nodes 17 and -1. With respect to the coordinate system defined by the father's common side (which in this case is in the opposite direction of the hatched element's orientation), the integer coordinates are identified as $1, 0$. These are seen to be the final two entries in the second column of Ncons. The remaining entries are constructed in a similar manner.

The orientations of the hatched element's active midside nodes need to be known in general. For a given unconstrained edge of the hatched element, the array Norient stores a value of 1 if the local orientation is consistent with the global orientation, a value of -1 if they are opposite, and a value of zero if the edge is constrained.

Step2: Definition of the modified element. The next two arrays of interest, Nod1 and Nod2, contain listings of the vertex and midside node numbers of the so-called modified element. These arrays are constructed from the arrays Ncons and Nodloc. To illustrate, the first entry in Nodloc is -1, indicating that it is constrained. The corresponding vertex and midside nodes along the father's common edge, 7, 17, and 22, are obtained from the first column of Ncons. Thus, 7 and 17 become the first two entries in Nod1, and 22 becomes the first entry in Nod2. The remaining entries are obtained in a similar way. Finally, Nod1 and Nod2 contain a listing of the vertex and midside nodes associated with the hatched element. These arrays define the so-called *modified element*, or *element with legs*, [24], which can be useful for visualizing the constraint situation. The modified element corresponding to the hatched element is shown at the bottom of Figure 10. It is a graphical representation of the locations of the nodes which constrain the element of interest (in this case the hatched element). The constraining vertex and midside nodes along the father's common edges constitute the "legs" of the modified element.

Once the above arrays are constructed, we have a complete description of the nodal information for the hatched element (and the modified element), including the nodal constraints. The next step is to do the same for the dof.

Step3: Construction of the constraints for the local dof. The previous section dealt only with nodal information. Next, we will consider the constraints for the dof. In particular, we will construct arrays that describe the dof for the modified element, as well as the constraint coefficients that enforce continuity of the approximation across element boundaries.

Norient

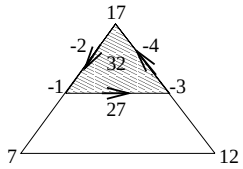
1	0	0

Nodloc

-1	-3	17	27	-4	-2	32

Ncons

7	7	12	12
17	17	17	17
22	22	20	20
0	1	0	0
0	0	0	1



(a)

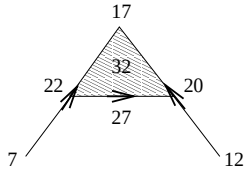
'Modified element'
'Element with legs'

Nod1

7	17	12

Nod2

22	20	27



(b)

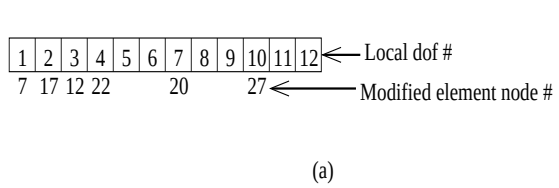
Figure 10: The nodal constraint information and associated modified element for a triangle with single-constrained nodes.

In a similar manner as for the nodal information, we need to consider each dof for the hatched element. If it is constrained, we need to determine a listing of the constraining dof along the father's common side. This information is stored in the matrix NAC. Thus, NAC contains, for each local dof of the hatched element, a corresponding column of local dof numbers of the modified element which constrain that dof. If a given dof is unconstrained, then its column in NAC will only contain one entry.

As shown in Figure 11a, Nod1 and Nod2 can be concatenated to form an ordered list of the vertex and midside node numbers for the modified element. Corresponding to each vertex node is one dof, and to each mid-side node are $p-1$ dof's, where p is the order of approximation. Thus, Figure 11(a) establishes a correspondence between local node numbers and local dof's for the modified element.

With this correspondence, the array NAC can be constructed as follows: for the i-th local dof of the hatched element, determine the node numbers of the constraining nodes from the matrix Ncons. Next, locate the constraining nodes on the list in Figure 11(a), and extract the corresponding local dof numbers of the modified element. These dof become the entries in the i-th column of NAC, corresponding to the i-th dof of the hatched element. For example, the first dof of the hatched element in this case is constrained by nodes 7, 17, and 22. Referring to Figure 11(a), we see that these correspond to dof 1, 2, 4, 5, and 6. As can be seen in Figure 11(b), these are precisely the entries in the first column of NAC.

At this point, it is instructive to draw an analogy between the arrays NAC and CONSTR, and the classical finite element approach. In either approach, within a given element, the values of geometrical coordinates and dof's at a point are represented as linear combinations of the nodal dof's (each of which is associated with a shape function). However, if one of the nodal dof's is constrained, then it is actually dependent upon other dof's within the element (as illustrated by the array NAC). Thus, in the constrained case values of geometrical coordinates and dof's at a point are actually 'nested' linear combinations of the nodal dof's.



(a)

Dof constraint information for the hatched element (NAC)

1	2	3	4	5	6	7	8	9	10	11	12
1	2	2	10	11	12	7	7	7	4	4	4
2	3					8	8	8	5	5	5
4	7					9	9	9	6	6	6
5	8										
6	9										

← Local dof #

(b)

Figure 11: An example of the construction of the dof constraint information for a triangle with single-constrained nodes.

With this in mind, the arrays NAC and CONSTR are recognized to be the second level of these nested summations. In fact, if an element were unconstrained, then each column in NAC would have only one entry, and the corresponding constraint coefficient would be 1.0. Thus, *the classical FEM approach is seen to be a special case of the more generalized procedure discussed here*. The element shown in Figure 9 has only single constrained nodes. If higher degrees of irregularity were allowed, then there would be a hierarchy of summations involved.

6.2 Quadrilateral elements

Step1: Reconstruction of the nodal constraints. We will begin with a simple example of the construction of the constraint information for a quadrilateral with only single constrained nodes, followed by an example with double-constrained nodes. Figure 12 shows a nine-node quadrilateral which has been refined horizontally, resulting in two constrained son elements. Figures 12(b) and 12(a), respectively, show the information available in the data structure for the hatched element and its father, and Figure 12(c) shows the reconstructed constraint arrays.

As in the case for triangles, notice that the first four entries in the ELEMS and Nodloc arrays of Figure 12 correspond to a counterclockwise listing of the vertex nodes, and the subsequent four entries are a counterclockwise listing of the mid-side nodes. The last entry, of course, is for the middle node. Negative entries in Nodloc indicate constrained nodes.

In this case, Ncons has four columns, each containing information for one of the constrained nodes listed in Nodloc. As in the case for triangles, the negative entries in Nodloc simply point to the respective column in Ncons. Thus, Figure 12 is the direct analog to Figure 9.

Steps2 and 3: Construction of the constraints for the local dof and the modified element. Figures 13(b) and 14 show the modified element and the reconstructed constraints for the dof, respectively. As in the case for triangles, Nod1 and Nod2 contain listings of the vertex and midside nodes of the modified element. For each local dof of the hatched element, NAC contains a column of local dof numbers of the modified element that constrain that local dof. The modified element for this case is shown in Figure 13(b), and it is the analog of the

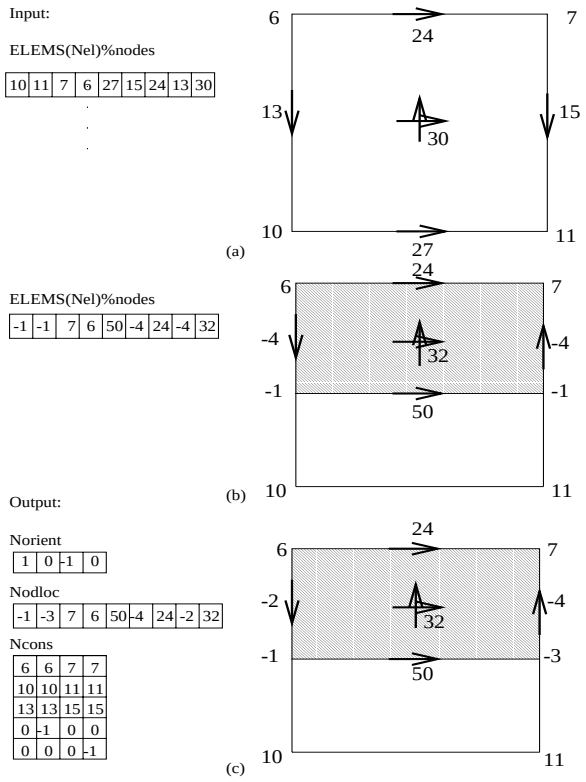


Figure 12: An example of the construction of nodal constraint information from the data structure arrays for a quadrilateral with single constrained nodes.

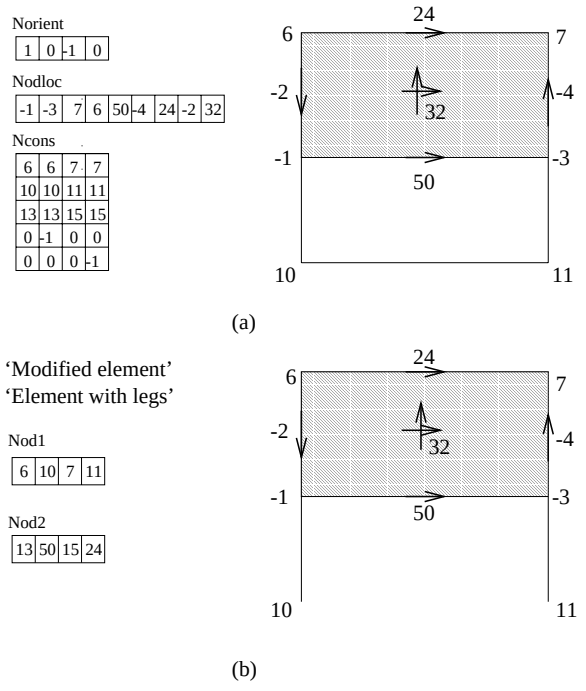


Figure 13: The nodal constraint information and associated modified element for a quadrilateral with single constrained nodes.

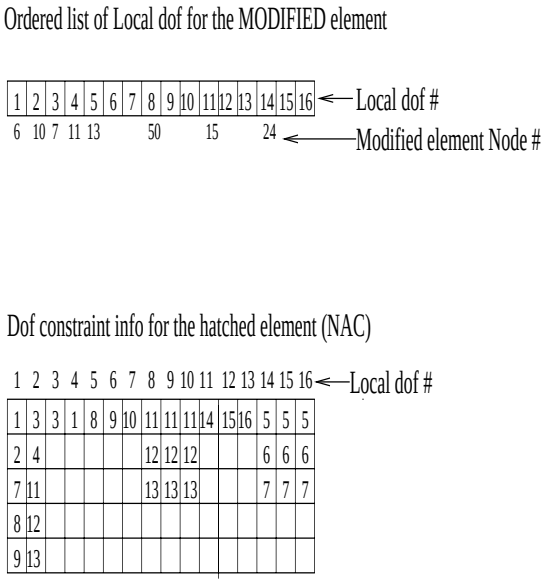


Figure 14: An example of the construction of the dof constraint information for a quadrilateral with single-constrained nodes.

case of a triangle with a single constrained node.

Condensation Process: Both the triangle and quadrilateral examples presented thus far have contained only single constrained nodes. Consequently, the arrays in Figures 9 and 12 differ only in the number of entries in the arrays.

Quadrilaterals, however, have the added technicality that they can accomodate *double constrained nodes*, i.e. nodes that are constrained by constrained nodes. Figures 15 and 16 illustrate the data structure information, corresponding nodal constraint arrays and modified element for a quadrilateral with a double constrained node resulting from two successive refinements. The most notable difference from the triangle case is the presence of negative entries (constrained nodes) in the array Ncons and Nod1, indicating the presence of double constrained nodes.

An additional loop through Nod1 to check for negative values is necessary to determine nodes of the modified element. For each negative entry in Nod1, the corresponding parent vertex and midside nodes are extracted from Ncons and added to Nod1 and Nod2 if they are not already there. After completion of this process, Nod1 will contain a listing of all *unconstrained* vertex nodes for the modified element, and Nod2 will contain the *unconstrained* midside nodes. The corresponding condensation process for the NAC and Ncons arrays is described in section 6.5.

As stated previousely, double constraints are the highest degree of irregularity allowed in the code. The mesh modification routines have built-in protocols that prevent higher degrees of irregularity.

6.3 Constrained approximation coefficients (CAC)

The coefficients that are used to impose the constraints, i.e. CONSTR, can be computed ahead of time or they can even be hardwired into the FE code. In any case, these coefficients are computed for all polynomial degrees p. For a typical situation, as shown in Figure 8, we simply

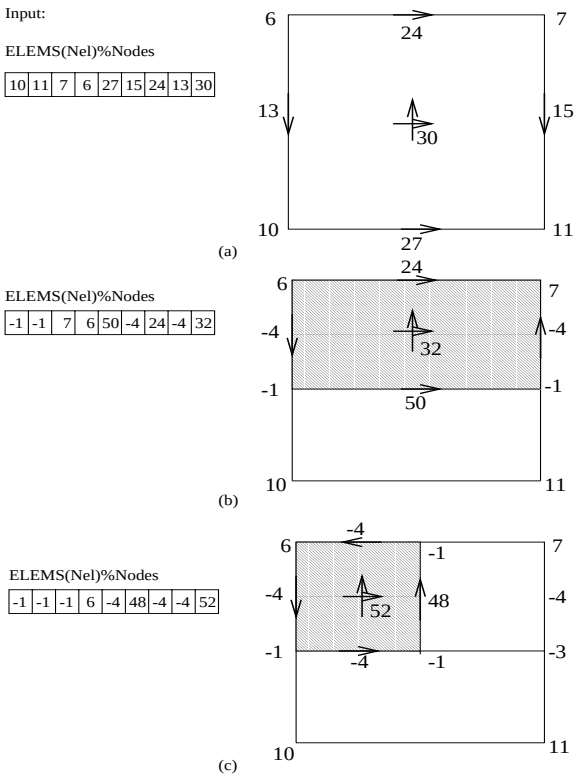


Figure 15: An example of the data structure information for a quadrilateral with double-constrained nodes.

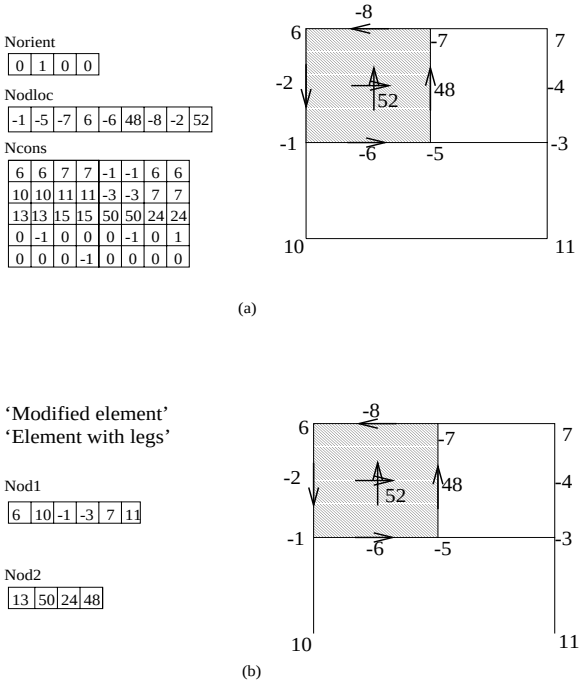


Figure 16: The nodal constraint information and the modified element for a quadrilateral with double constrained nodes.

need to determine the values of the big element shape functions at the nodes x_j of the small neighbors. The nodes x_j have already been defined in section 2.4 and we only need to establish a numbering convention for these coefficients, which are collected in an array $CAC(19, 9, 9)$. We interprete them so that $CAC(j, i, p-1)$ is the value of the i-th big element shape function at the j-th node of the small neighbor with the convention that $2 \le p \le 10$, and that the small element nodes are numbered with $j = 1$ for the node that coincides with the mid-side of the large element and with $2 \le j \le p$ for the first small element nodes and $p+1 \le j \le 2p-1$ for the second small element nodes. The coefficients are then easily be determined by the following algorithm:

```
do p=2,10
  determine shape functions φ_i at x = 0.5
  do i=1,p-1; CAC(1,i,p-1) = vshap(2+i); enddo
  do j=1,p-1
    determine shape functions φ_i at x_j
    do i=1,p-1
      CAC(1+j,i,p-1) = vshap(2+i) - CAC(1,i,p-1)*2.d0*x_j
    enddo
  enddo
  do j=1,p-1
    determine shape functions φ_i at x_j + 0.5
    do i=1,p-1
      CAC(p+j,i,p-1) = vshap(2+i) - CAC(1,i,p-1)*2.d0*(1-x_j)
    enddo
  enddo
enddo
```

6.4 Selection of the constrained coefficients

In section 6.3 we already described how to obtain the coefficients CAC for all possible polynomial degrees p and in sections 6.1 and 6.2 we introduced the modified element information. Now we have to select particular coefficients from the list CAC that correspond to the approximation orders of the modified element. The selected coefficients are then stored in the list $CONSTR(j, i)$, with $j = 1, \cdots, NRCON(i)$, $i = 1, \cdots,$ number of local element dof. Therefore, $CONSTR(j, i)$ contains the constraint coefficients corresponding to the i-th local dof. Obviously, in the case of any unconstrained dof we have $CONSTR(1, k) = 1$, for $k =$ local dof number. This is in particular the case for all the dof associated with the middle node, because the middle node is always unconstrained. In the case of a constrained local vertex node with local dof number k we have

$$CONSTR(1, k) = \frac{1}{2}; \quad CONSTR(2, k) = \frac{1}{2}$$

and

$$CONSTR(2 + j, k) = CAC(1, j, p-1), \quad j = 1, \cdots, p-1$$

if the orientation of the parent mid-side node is consistent with the local orientation, and otherwise

$$CONSTR(2 + j, k) = CAC(1, p-j, p-1), \quad j = 1, \cdots, p-1$$

where p is the approximation order of the parent mid-side node. In the case of a constrained mid-side node we additionally have to take the position of the small element with respect to the father element into account. Therefore we have for the local mid-side node with parent mid-side node of order p:

```
do j=1,p-1
   set k
   do jp = 1,p-1
    if small element is located at first half of the edge of the large element then
       if edge orientation is consistent then CONSTR(jp,k) = CAC(1+j,jp,p-1)
       else CONSTR(jp,k) = CAC(2*p-j,jp,p-1)
       endif
    else
       if edge orientation is consistent then CONSTR(jp,k) = CAC(p+j,jp,p-1)
       else CONSTR(jp,k) = CAC(p+1-j,jp,p-1)
       endif
    endif
  enddo
enddo
```

Thus, the constrained approximation coefficients are determined for an element with variable approximation order.

6.5 Treatment of double constrained nodes in a quadrilateral. The condensation process

The next step in the constrained approximation procedure is to eliminate the constrained nodes that are still present. We have already discussed that this is only possible in the case of a quadrilateral with double constrained vertex nodes. The following algorithm is based purely on algebraic considerations and takes care of the double constrained nodes. The particular orientation of an edge and the corresponding mid-side node is not important at this stage and only the data in the lists $NRCON$, NAC and $CONSTR$ are updated. Therefore, we first have to determine the modified element and then select the corresponding constraint coefficients, according to the algorithm in section 6.4. Additionally, it is also necessary to determine the local `Nrloc` and global `Nrglob` number of dof in the modified element. If these two numbers are equal, then there is no double constrained node present and the following algorithm need not be executed. Otherwise the double constraints are condensated out:

```
naf(i) = location of dof i after condensation for an unconstrained dof
       = -number of constraint for a constrained dof i
do k = Nrloc + 1, Nrglob      (renumerate dof)
  do kp = 1, NRCON(k)
    NAC(kp,k) = naf(NAC(kp,k))
  enddo
enddo
do k = 1, Nrloc      (condense connected dof)
  kc = 0; k1 = 0
  do kp = 1, NRCON(k)      (making a list of connected constrained dof)
```

```
      if (naf(NAC(kp,k)) ¿ 0) then
        k1 = k1 + 1; NAC(k1,k) = naf(NAC(kp,k))
        CONSTR(k1,k) = CONSTR(kp,k)
      else
        kc = kc + 1; nacl(kc) = -naf(NAC(kp,k))
        constrl(kc) = CONSTR(kp,k)
      endif
    enddo
    nrconl = kc
    do kc = 1, nrconl
      do kcp = 1, NRCON(nacl(kc))      (loop through connected dof)
        n=0
        if connected dof is not on list NAC then
          k1 = k1 + 1; n = k1; NAC(k1,k) = NAC(kcp,nacl(kc))
        enddo
        CONSTR(n,k) = CONSTR(n,k) + constrl(kc) * CONSTR(kcp,nacl(kc))
      enddo
    enddo
    NRCON(k) = k1
  enddo
```

7 Generalized assembling procedure and element computations

We first review shortly the regular assembling procedure before considering the generalized assembling procedure. This generalized assembling procedure uses the modified element information and the constrained approximation coefficients to ensure a continuous approximation across element edges. Then we reveal details about the element computations and the application of general boundary conditions, which are enforced at the local element stiffness matrix level.

7.1 Generalized assembling

For a regular mesh the continuity of the approximation is established by enforcing the local dof to coincide with the corresponding global dof. In practice this is done by introducing the connectivity list $NA(i)$, $i = 1, \cdots, N$, which assigns for each local dof i the corresponding global dof $NA(i)$. The usual assembling procedure for the global load vector $LOAD$ and global stiffness matrix $STIFF$ is then given by

```
do for each local dof i1
  LOAD(NA(i1)) = LOAD(NA(i1)) + load(i1)
  do for each local dof i2
    STIFF(NA(i1),NA(i2)) = STIFF(NA(i1),NA(i2)) + stiff(i1,i2)
  enddo
enddo
```

Here *load* and *stiff* are the local load vector and local stiffness matrix that correspond to

the local unconstrained element.

For a finite element mesh with constraint nodes this standard procedure fails to enforce the continuity of the approximation. Instead of the list NA the list NAC has to be used, which corresponds to the constrained case. This allows then to apply the constraint coefficients $CONSTR$. The algorithm for assembling the global load vector and stiffness matrix is then:

```
do for each local dof i1
  do ip1=1,NRCON(i1) (for each connected dof)
    LOAD(NAC(ip1,i1)) = LOAD(NAC(ip1,i1)) + load(i1)*CONSTR(ip1,i1)
    do for each local dof i2
      do ip2=1,NRCON(i2) (for each connected dof)
        STIFF(NAC(ip1,i1),NAC(ip2,i2)) = STIFF(NAC(ip1,i1),NAC(ip2,i2)) +
          stiff(i1,i2)*CONSTR(ip1,i1)*CONSTR(ip2,i2)
      enddo
    enddo
  enddo
enddo
```

This generalized assembling procedure, as it is given here, is written for a scalar problem. In the case of a vector valued problem this algorithm can easily be extended to account for the block structure of the corresponding stiffness matrix. It is also possible to take advantage of symmetries by slightly modifying the procedure. The spaces $S^{div}(\mathcal{M},\Omega)$ and $S^{curl}(\mathcal{M},\Omega)$ in (3.15) and (3.16) require a further generalization, we refer to [11] for a possible extension of the generalized assembling procedure that is well suited for the electromagnetic scattering problem. The presented methodology is very general and can be applied to various problems of engineering interest, provided that the appropriate modifications are incorporated into the generalized assembling procedure.

7.2 Element computations

The element stiffness matrix and load vector are computed by mapping the element onto the master element and performing all calculations on the master element. In particular, these computations are done by assuming that the local element is completely unconstrained. This means that the element computations are done in the standard way and that the user only needs to provide an element routine that corresponds to the problem that is being solved. Again, the user does not have to worry about the constrained approximation and the enforcement of the continuity of the approximation. At this point we also do not include the boundary conditions into the element computations (general boundary conditions are treated in section 7.3). Also, the possibly variable polynomial approximation order within an element does not introduce any difficulties into the element computations because the corresponding data are automatically provided by the data structure and the supporting routines. The integration over an element and possibly over element edges are done by using the standard Gauss integration rules for quadrilaterals and triangles. In the code we provide Gauss rules that are exact for polynomials up to degree 19. If the user wishes to use other integration schemes then they can easily be added to the code.

The element computations are similar for quadrilaterals and triangles and are performed in the following manner:

- determine the features of the element: type of element, order of approximation, geometry, etc.

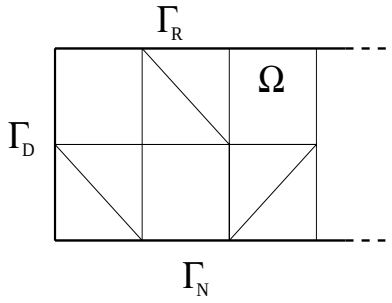


Figure 17: Dirichlet, Neumann and Robin boundary condition.

- establish order of Gauss Quadrature
- loop through the integration points
  - compute values and derivatives of the shape functions at the integration point
  - evaluate material parameters
  - determine the jacobian of the mapping from the master element onto the actual element
  - accumulate contributions of the integration point to subsequent entries of the element stiffness matrix and load vector

7.3 Application of general boundary conditions

We allow Dirichlet (essential), Neumann (natural) and Robin (mixed) boundary conditions to be imposed on the boundary of the 2D domain Ω that is being discretized. The only requirement on the boundary conditions is that they do not overlap. That is that the boundary Γ of the domain Ω is the disjoint union of the Dirichlet, Neumann and Robin boundaries, i.e. $\Gamma = \Gamma_D \cup \Gamma_N \cup \Gamma_R$.

In the following we describe how these general boundary conditions are implemented in the code. Figure 17 shows a typical situation and it is obvious that element edges can be part of the boundary Γ and that for a triangular element it is additionally possible that single vertices are part of the boundary. In any case, we only require that an edge of an element can only be part of one of the three different boundary parts. This assumption need not hold for vertex nodes. The abstract data structure introduced in section 5 provides for each element the information about its connection to the boundary and the type of boundary conditions to be imposed. The Neumann and Robin boundary conditions result in a modification of element load vector and/or element stiffness matrix and, therefore, they are implemented at the local element level, *before* the constraint approximation modifications are imposed. The Dirichlet boundary conditions, however, concern the *actual* degrees of freedom and, therefore, they have to be applied to the modified element. The methodology is illustrated using as a model problem the Poisson equation with all three different types of boundary conditions.

$$\begin{array}{lrcll} \text{Poisson problem:} & -\Delta u & = & f & \text{on } \Omega \\ \text{Dirichlet BC:} & u & = & u_D & \text{on } \Gamma_D \\ \text{Neumann BC:} & \dfrac{\partial u}{\partial n} & = & u_N & \text{on } \Gamma_N \\ \text{Robin BC:} & \dfrac{\partial u}{\partial n} + a\,u & = & u_R & \text{on } \Gamma_R,\ 0 < c \le a,\ a \in L^\infty(\Gamma_R) \end{array} \tag{7.2}$$

Obviously, the corresponding weak formulation is obtained by multiplying with a test function $v \in H_0^1(\Omega)$ and integrating by parts:

$$\begin{aligned}&\text{Find } u \in H_D^1(\Omega) \text{ such that } \forall\ v \in H_0^1(\Omega)\\ &\int_\Omega \boldsymbol{\nabla} u \cdot \boldsymbol{\nabla} v\ d\Omega + \int_{\Gamma_R} auv\ dS = \int_\Omega fv\ d\Omega + \int_{\Gamma_N} u_N v\ dS + \int_{\Gamma_D} \frac{\partial u}{\partial n} v\ dS + \int_{\Gamma_R} u_R v\ dS\end{aligned} \tag{7.3}$$

with $H_0^1(\Omega) = \{v \in H^1(\Omega); v = 0 \text{ on } \Gamma_D\}$ and $H_D^1(\Omega) = \{u \in H^1(\Omega); u = u_D \text{ on } \Gamma_D\}$. From (7.3) we note that the Neumann and Robin boundary conditions do not introduce any complications into the element computations. They just add some boundary integrals to the element stiffness matrix and the load vector, which can be computed similar to the element computations.

The only technical difficulty arises from the Dirichlet boundary condition. Let us first assume that we already have computed the element stiffness matrix and the element load vector with the corresponding Neumann and Robin boundary conditions, i.e.

$$\begin{aligned}\boldsymbol{B} = (\boldsymbol{B}_{ij}),\ \ \boldsymbol{B}_{ij} &= \int_K \boldsymbol{\nabla}\varphi_i \cdot \boldsymbol{\nabla}\varphi_j\ d\Omega + \int_{\partial K \cap \Gamma_R} a\,\varphi_i\,\varphi_j\ dS && 1 \le i,j \le N_e\\ \boldsymbol{F} = (\boldsymbol{F}_j),\ \ \boldsymbol{F}_j &= \int_K f\,\varphi_j\ d\Omega + \int_{\partial K \cap \Gamma_N} u_N\,\varphi_j\ dS + \int_{\partial K \cap \Gamma_R} u_R\,\varphi_j\ dS && 1 \le j \le N_e\end{aligned} \tag{7.4}$$

where N_e is the local number of dof for element K and φ_i are the local element shape functions. If $\partial K \cap \Gamma_D \neq \emptyset$ then we construct an interpolant of the Dirichlet data on $\partial K \cap \Gamma_D$ by using the element shape functions φ_i, $1 \le i \le N_e$, i.e.

$$I_h u_D = \sum_{i=1}^{N_e} u_D^i\,\varphi_i. \tag{7.5}$$

In practice we construct a Lagrange interpolant by requiring that

$$u_D(y_j) = \sum_{i=1}^{N_e} u_D^i\,\varphi_i(y_j)\ , \quad 1 \le j \le p+1, \tag{7.6}$$

where p is the approximation order on the edge that is part of the Dirichlet boundary. In principle any set of points y_j could be used, including equidistant points, but we emphasize that theoretical results [28] show that the Gauss Lobatto points should be used, to obtain a robust interpolant. Based on the interpolation points, the coefficients are collected in the vector $\boldsymbol{u}_D$, which is then used to modify the load vector $\boldsymbol{F}$, i.e.

$$\boldsymbol{F} = \boldsymbol{F} - \boldsymbol{B}\,\boldsymbol{u}_D. \tag{7.7}$$

The components of $\boldsymbol{F}$ that correspond to dof on the Dirichlet boundary are then set to the corresponding Dirichlet data, i.e. let $\boldsymbol{F}_i$ correspond to a dof that is located at y_i, then we set

$$\boldsymbol{F}_i = u_D(y_i). \tag{7.8}$$

In the case of a triangular element and only a vertex being part of Γ_D, the same procedure applies, but the determination of $\boldsymbol{u}_D$ becomes completely trivial.

We emphasize that the modification of the element load vector takes place for the *modified element*, i.e. after the constrained approximation has been applied.

After modifying $\boldsymbol{F}$ we only need to modify $\boldsymbol{B}$ by putting zeros in every column and row that corresponds to a dof that is located on Γ_D and then setting the corresponding diagonal entry of $\boldsymbol{B}$ to one. With this procedure we can impose homogeneous and nonhomogeneous Dirichlet boundary conditions in a numerically stable way. Further, it is evident that this methodology assures that the test and trial functions are consistent with the test and trial spaces.

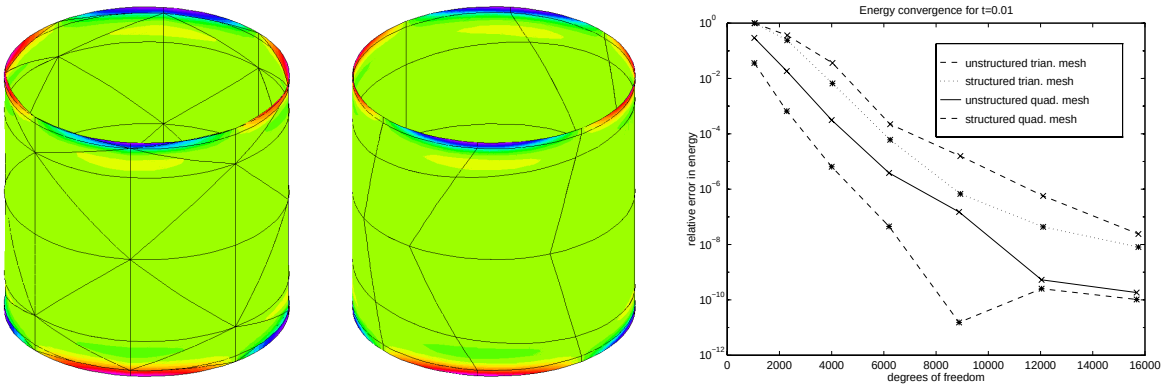


Figure 18: Shell solution on a structured/unstructured mesh and convergence rates.

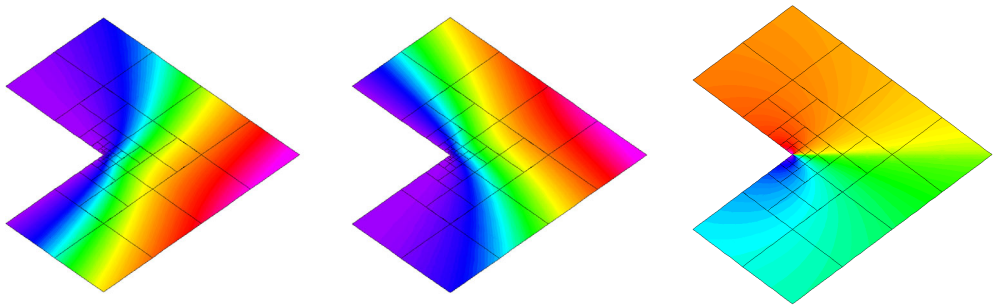


Figure 19: Solution on a geometrically refined mesh.

8 Numerical Examples

We confine our numerical examples to two significant problems of engineering interest, namely the shell problem and the Stokes problem. Here we only report briefly on the numerical results and refer to our previous work [15, 25] for additional details.

In [15] we analyzed in detail the numerical solution of shell problems. We used the Naghdi shell model, a second order elliptic system for three displacement- and two rotation fields. In our theoretical investigation we derived an exponential convergence result for the $hp-$FEM. This result could only be obtained because the $hp-$FEM is capable of resolving the boundary layers that are in the shell problem. Figure 18 shows the hp-approximation of a rotation on a structured and an unstructured $hp-$FE mesh. The corresponding convergence rates for triangular and quadrilateral meshes are also displayed. These convergence rates clearly indicate that the $hp-$FEM converges for a shell thickness $t = 0.01$. Also it is evident that the $hp-$FEM is robust, since the results are similar for structured and unstructured meshes.

Our second example is Example 3 above, the Stokes problem, a mixed problem. In Figure 19 we show the solution to the Stokes problem. We see that the pressure has a singularity at the reentrant corner and we clearly see the geometric refinement towards this reentrant corner. The corresponding convergence rates are displayed in Figure 20 and the exponential convergence of our $hp-$FEM is obvious. For further theoretical investigations, we refer to [25, 27].

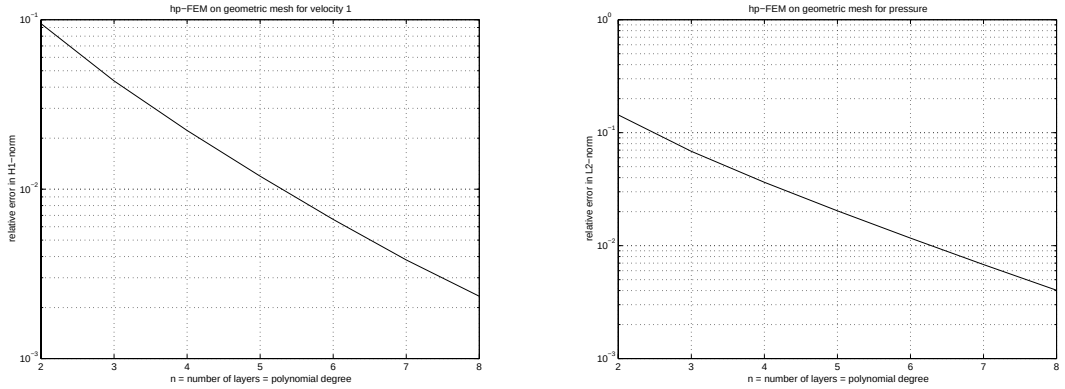


Figure 20: Convergence rates for velocity and pressure.

9 Summary and future work

A new framework for scientific computations for two dimensional problems is described in this work. The described $hp-$FE code relies on a flexible implementation which allows a general class of hp finite elements. Among them are the classical discontinuous and continuous finite elements and their vector valued counterparts as well as the non standard $H(div)$ and $H(curl)$ conforming finite elements.

Practically, all families of hp-elements can be realized within this package and general systems of equations can be discretized, allowing in particular multiple field problems. The implemented anisotropic mesh refinement algorithms provide a basis for the extraordinary numerical performance, which is evident from the numerical examples presented in the previous section.

In our future work we will also address issues that are related to the optimization of the code, i.e. solution techniques and in particular fast assembling algorithms that reduce the global system size and allow for a significant speed up of the linear system solver. In this context, we will also investigate on domain decomposition techniques and parallelization in the spirit of [22]. Another natural extension of the presented $hp-$framework is to address nonlinear problems and also to extend to fully three dimensional problems.

In our forthcoming work we will present additional evidence to support the superior performance of this $hp-$FE framework.

References

[1] I. Babuška and M. Suri. The p and hp Versions of the Finite Element Methods, Basic Principles and Properties. *SIAM review* **36** (1994) 578-632.

[2] P. G. Ciarlet. *The Finite Element Methods for Elliptic Problems*. North-Holland 1987.

[3] L. Demkowicz, J.T. Oden, W. Rachowicz and O. Hardy. Toward a Universal hp Adaptive Finite Element Strategy. Part 1: Constrained Approximation and Data Structure. *Computer Methods in Applied Mechanics and Engineering*, 77 (1989), 79-112.

[4] J.T. Oden, L. Demkowicz, W. Rachowicz and T. A. Westermann. Toward a Universal hp Adaptive Finite Element Strategy. Part 2: A Posteriori Error Estimation. *Computer Methods in Applied Mechanics and Engineering*, 77 (1989), 113-180.

[5] D. Braess. *Finite Elements*. Cambridge University Press, 1997.

[6] F. Brezzi and M. Fortin. *Mixed and hybrid Finite Element Methods*. Springer Series in Comp. Mathematics vol. 15 , Springer Verlag New York (1991).

[7] L. Demkowicz, A. Bajer and K. Banas. Geometric Modeling Package. *TICAM Report 92-06*, 1992.

[8] L. Demkowicz, W. Rachowicz, K. Banas and J. Kucwaj. 2-D *hp* Adaptive Package (2D*hp*AP). *Technical Report*, Section of Applied Mathematics, Technical University of Cracow, Poland, 1992.

[9] L. Demkowicz and J. T. Oden. 'New Developments in Applications of *hp*-Adaptive BE/FE Methods to Elastic Scattering. in *The mathematics of finite elements and applications*, ed. by J. R. Whiteman, Wiley, 1994.

[10] L. Demkowicz, A. Karafiat and J. T. Oden. Solution of Elastic Scattering Problems in Linear Acoustics using *hp* Boundary Element Method. *Computer Methods in Applied Mechanics and Engineering*, vol. 101, pp. 251-282, 1992.

[11] L. Demkowicz and L. Vardapetyan. Modeling of Electromagnetic Absorption/Scattering Problems using *hp*-Adaptive Finite Elements. *TICAM Report*, 1997, to appear in *Computer Methods in Applied Mechanics and Engineering*.

[12] K. Gerdes. Solution of the 3D Laplace and Helmholtz Equation in Exterior Domains of Arbitrary Shape using HP-Finite-Infinite Elements. Doctoral Dissertation, The University of Texas at Austin, 1996.

[13] K. Gerdes and L. Demkowicz. Solutions of 3D-Laplace and Helmholtz Equations in Exterior Domains using hp Infinite Elements. *Comput. Methods Appl. Mech. Engrg.* 137 (1996) 239-273.

[14] K. Gerdes and F. Ihlenburg. On the Pollution Effect in FE Solutions of the 3D-Helmholtz Equation. In print in *Comput. Methods Appl. Mech. Engrg.*

[15] K. Gerdes, A. M. Matache and C. Schwab. On Membrane Locking in *hp* FEM for a Cylindrical Shell. In print in *Zeitschrift für Angewandte Mathematik und Mechanik*.

[16] V. Girault and P.-A. Raviart. *Finite Element Methods for Navier-Stokes Equations*. Springer-Verlag, 1986.

[17] B. Guo and I. Babuška. The $hp-$sVversion FEM I: The Basic Approximation Results; and Part II: General Results and Applications. *Comp. Mech.* vol. 1 (1986) 21-41 and 203-226.

[18] F. Kikuchi. Mixed and Penalty Formulations for Finite Element Analysis of an Eigenvalue Problem in Electromagnetism. *Computer Methods in Applied Mechanics and Engineering* vol. 64 (1987), 509-521.

[19] P. Monk. An Analysis of Nédélec's Method for the Spatial Discretization of Maxwell's Equations. *J. Comp. Appl. Math.* vol. 47 (1993) 101-121.

[20] J. C. Nedelec. Mixed Finite Elements in $\mathbb{R}^3$. *Numerische Mathematik*, vol. 35 (1980) 315-341.

[21] J. C. Nedelec. A New Family of Mixed Finite Elements in $\mathbb{R}^3$. *Numerische Mathematik*, vol. 50 (1986) 57-81.

[22] J. T. Oden, A. Patra and Y. Feng. Parallel Domain Decomposition Solver for Adaptive *hp* Finite Element Methods. *SIAM J. Numer. Anal.* vol. 34 (1997) 2090-2118.

[23] W. Rachowicz, "An Anisotropic *h*-Type Mesh-Refinement Strategy," *Computer Methods in Applied Mechanics and Engineering*, **109** (1993) 169-181

[24] W. Rachowicz, "An -*hp* Finite Element Method on One-Irregular Meshes, Error Estimation and Mesh Refinement Strategy. Doctoral Dissertation, The University of Texas at Austin, 1989.

[25] D. Schötzau, C. Schwab and R. Stenberg. Mixed *hp*-FEM on Anisotropic Meshes II: Hanging Nodes and Tensor Products of Boundary Layer Meshes. *SAM Report 97-14*, submitted to *Numerische Mathematik*.

[26] C. Schwab and M. Suri. The *p* and *hp* Versions of the Finite Element Method for Problems with Boundary Layers. *Math. Comp.* vol. 65 (1996) 1403-1429.

[27] C. Schwab and M. Suri. Mixed *hp*-FEM for Incompressible Fluid Flow. In preparation.

[28] Burkhard Sündermann. Lebesgue Constants in Lagrangian Interpolation at the Fekete points. Ergebnisberichte der Lehrstühle Mathematik III und VIII (Angewandte Mathematik) 44, Universität Dortmund, 1980.

[29] R. Stenberg and M. Suri. Mixed *hp* Finite Element Methods for Problems in Elasticity and Stokes Flow. *Num. Math.* **72** (1996), 367-389.

[30] R. Stenberg and M. Suri. An *hp*-Error Analysis of MITC Plate Elements. *SIAM J. Numer. Anal.* vol. 34, 544-568.

[31] B. Szabo and I. Babuska. *Finite Element Analysis*. Wiley 1991.