Numerically Solving Maxwell's Equations. Implementation Issues for Magnetoquasistatics in KARDOS

Delia Teleaga and Jens Lang [1]

Abstract.

Having experienced a couple
of tricky issues during the implementation
of edge elements within the fully space-time adaptive
PDE solver KARDOS [6] to solve magnetoquasistatic problems
we found it useful to share our exciting learning process with
interested readers and beginners.

June 20, 2008

[1]This work was partly supported by the Deutsche Forschungsgemeinschaft (DFG) within the project "Space-time adaptive magnetic field computation" under the grants LA1372/3-1 and LA1372/3-2.

Contents

Chapter 1

Maxwell's Equations and Magnetoquasistatics

Electromagnetic phenomena are governed by Maxwell's equations. These equations relate the electric and magnetic field intensity vectors and the material properties of a medium. The full set of equations may be written as

$$\mathbf{curl}\boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t} \quad \text{(Faraday's law)} \tag{1.1}$$

$$\mathbf{curl}\boldsymbol{H} = \frac{\partial \boldsymbol{D}}{\partial t} + \boldsymbol{J} \quad \text{(Ampere's law)} \tag{1.2}$$

together with the material laws

$$\boldsymbol{B} = \mu \boldsymbol{H}, \quad \boldsymbol{J} = \sigma \boldsymbol{E} + \boldsymbol{J}_s, \quad \boldsymbol{D} = \epsilon \boldsymbol{E}\ , \tag{1.3}$$

and the continuity equation

$$\frac{\partial \rho}{\partial t} + \operatorname{div} \boldsymbol{J} = 0, \tag{1.4}$$

where

$\boldsymbol{B}$ – magnetic flux density
$\boldsymbol{H}$ – magnetic field intensity
$\boldsymbol{D}$ – displacement current density
$\boldsymbol{E}$ – electric field intensity
$\boldsymbol{J}$ – electric current density
$\boldsymbol{J}_s$ – applied current density
ρ – charge density
μ – medium's permeability
σ – medium's electric conductivity
ϵ – medium's permittivity

Important consequences are

$$\operatorname{div} \boldsymbol{D} = \rho \tag{1.5}$$
$$\operatorname{div} \boldsymbol{B} = 0 \tag{1.6}$$

Boundary conditions.
The equations above must be supplemented by appropriate boundary conditions. At the surface of a perfect electrical conductor (PEC), the electric and the magnetic fields must be such that

$$\boldsymbol{n} \times \boldsymbol{E} = \boldsymbol{0}, \quad \boldsymbol{n} \cdot \boldsymbol{H} = 0, \tag{1.7}$$

where $\boldsymbol{n}$ is the unit outward normal vector to the boundary, i.e., the PEC boundary condition sets the tangential component of the electric field and the normal component of the magnetic field to zero. Correspondingly, at the surface of a perfect magnetic conductor (PMC), the fields must be such that

$$\boldsymbol{n} \times \boldsymbol{H} = \boldsymbol{0}, \quad \boldsymbol{n} \cdot \boldsymbol{E} = 0. \tag{1.8}$$

If problems formulated in infinite domains are considered, these conditions must be supplemented by suitable radiation conditions.

Interface conditions.
If regions of the domain have different material properties, one should impose at an interface between sub-domains a and b the tangential jump conditions

$$\boldsymbol{n} \times (\boldsymbol{E}^a - \boldsymbol{E}^b) = \boldsymbol{0}, \quad \boldsymbol{n} \times (\boldsymbol{H}^a - \boldsymbol{H}^b) = \boldsymbol{0} \tag{1.9}$$

on the electric and magnetic fields and the normal jump conditions

$$\boldsymbol{n} \cdot (\epsilon^a \boldsymbol{E}^a - \epsilon^b \boldsymbol{E}^b) = 0, \quad \boldsymbol{n} \cdot (\mu^a \boldsymbol{H}^a - \mu^b \boldsymbol{H}^b) = 0 \tag{1.10}$$

on the flux densities.
Maxwell's equations describe the most intricate electromagnetic wave phenomena. Of course, the analysis of such fields is difficult and not always necessary. Wave phenomena occur on short time scales or at high frequencies that are often of no practical concern. If this is the case, the fields may be described by truncated versions of Maxwell's equations applied to relatively long time scales and low frequencies (quasistatics). The quasistatic laws are obtained from Maxwell's equations by neglecting either the magnetic induction or the electric displacement current. Here, we are interested in the latter, also known as eddy current model.

Magnetoquasistatics.
It is assumed that

$$\frac{\partial \boldsymbol{D}}{\partial t} = 0. \tag{1.11}$$

Using the *magnetic vector potential* $\boldsymbol{A}$ defined by

$$\boldsymbol{B} = \mathbf{curl} \boldsymbol{A} \ , \tag{1.12}$$

we obtain from (1.1) the electric field intensity $\boldsymbol{E}$ expressed as function of $\boldsymbol{A}$

$$\boldsymbol{E} = -\frac{\partial \boldsymbol{A}}{\partial t}. \tag{1.13}$$

Now from (1.1), (1.11), and (1.3) we end up with an equation for the magnetic vector potential $\boldsymbol{A}$:

$$\mathbf{curl}(\frac{1}{\mu} \mathbf{curl} \boldsymbol{A} \ ) + \sigma \frac{\partial \boldsymbol{A}}{\partial t} = \boldsymbol{J}_s, \tag{1.14}$$

where

$$\operatorname{div} \boldsymbol{J}_s = 0, \tag{1.15}$$

if we use the *Coulomb gauging* $\operatorname{div} \boldsymbol{A} = 0$.
In what follows we are interested in solving equation (1.14) for $\boldsymbol{A}$.

Chapter 2

Edge FEM for Maxwell's Equations

Besides finite difference methods and the method of moments, another attempt to solve Maxwell's equations is the classical Finite Element Method (FEM). In literature, as e.g. in [9], it is shown that when the usual node-based elements, obtained by interpolating the nodal values, are employed to represent vector electric or magnetic fields, then several serious problems are encountered. There are three main problems:

- the occurence of non-physical or so-called spurious solutions, which is generally attributed to lack of enforcement of the divergence condition,
- the inconvenience of imposing boundary conditions at material interfaces as well as at conducting surfaces,
- the difficulty in treating conducting and dielectric edges and corners due to field singularities associated with these structures.

Another approach, more suitable for solving electromagnetic problems, is the vector or edge FEM (EFEM). This uses vectorial basis function in the space $\boldsymbol{H}(\mathrm{curl})$. The properties of these functions are presented in Section 2.2. Firstly, a scalar weak formulation is derived from (1.14) and the main differences between a nodal and a vector FEM are underlined, because we have to correspondingly modify the fully adaptive space-time solver KARDOS.

2.1 Towards discrete equations for EFEM

Let Ω be a bounded domain in $\mathbb{R}^3$ with boundary Γ. The outer normal vector is denoted by $\boldsymbol{n}$. We introduce the following spaces:

$$\begin{aligned} \boldsymbol{H}(\mathrm{curl};\Omega) &= \{\boldsymbol{v} \in (L^2(\Omega))^3 : \mathbf{curl}\,\boldsymbol{v} \in (L^2(\Omega))^3\}, \\ \boldsymbol{H}_0(\mathrm{curl};\Omega) &= \{\boldsymbol{v} \in \boldsymbol{H}(\mathrm{curl};\Omega) : \boldsymbol{v} \times \boldsymbol{n} = \mathbf{0} \text{ on } \Gamma\}. \end{aligned}$$

We are looking for a solution $\boldsymbol{A}$ to (1.14) in the space $\boldsymbol{H}_0(\mathrm{curl};\Omega)$. A variational formulation of (1.14) is the following: find $\boldsymbol{A} \in \boldsymbol{V} = \boldsymbol{H}_0(\mathrm{curl};\Omega)$ such that

$$(\mu^{-1}\mathbf{curl}\,\boldsymbol{A}\,, \mathbf{curl}\,\boldsymbol{W})_\Omega + \sigma(\frac{\partial \boldsymbol{A}}{\partial t}, \boldsymbol{W})_\Omega = (\boldsymbol{J}_s, \boldsymbol{W})_\Omega, \forall \boldsymbol{W} \in \boldsymbol{V}. \tag{2.1}$$

Remarks:
1. There is often a debate in the literature about whether or not it is necessary to specify the divergence condition $\mathrm{div}\,\boldsymbol{A} = 0$ explicitly. Theoretically, one should always include a Lagrange multiplier in the variational formulation.
2. We have not yet considered a Lagrange multiplier in our weak formulation, because this would imply that we should use simultaneously edge and classical elements in KARDOS, since the Lagrange multiplier belongs to $H_0^1(\Omega)$.

The Galerkin FE approximation is usually obtained by replacing the continuous space $\boldsymbol{V}$ by conforming discrete subspaces, i.e., we are looking for a solution $\boldsymbol{A}_N \in \boldsymbol{V}_N$, where $\boldsymbol{V}_N$ is a finite-dimensional subspace of $\boldsymbol{V}$ such that

$$(\mu^{-1}\mathbf{curl}\,\boldsymbol{A}_N, \mathbf{curl}\,\boldsymbol{W}_N)_\Omega + \sigma(\frac{\partial \boldsymbol{A}_N}{\partial t}, \boldsymbol{W}_N)_\Omega = (\boldsymbol{J}_s, \boldsymbol{W}_N)_\Omega, \forall \boldsymbol{W}_N \in \boldsymbol{V}_N. \tag{2.2}$$

Let $\{\boldsymbol{\Phi}_i : i = 1, \ldots, N\}$ be a basis of $\boldsymbol{V}_N$. This implies that $\boldsymbol{A}_N \in \boldsymbol{V}_N$ can be written as:

$$\boldsymbol{A}_N(\boldsymbol{x}) = \sum_{i=1}^{N} u_i \boldsymbol{\Phi}_i(\boldsymbol{x}), \quad \forall\, \boldsymbol{x} \in \Omega. \tag{2.3}$$

One has to determine the coefficients u_i, called also *degrees of freedom*.

KARDOS uses classical FEM. If the unknown is a vector, as e.g. $\boldsymbol{A} = (A_1, A_2, A_3)^T$ in (1.14), then the variable *noOfEquations* should be set to 3 in *kardos.c* and we should have indeed three equations for finding the three components of the solution A_i, $i = 1, 2, 3$. But this is not the case when

using edge elements, i.e., when the basis functions are vector functions. After multiplying (1.14) with a vector basis function, integrating over Ω and using integration by parts, one obtains the scalar variational problem (2.2). Thus, although the basis functions and the solution are vectors, the degrees of freedom are scalars.

This implies the following modifications in KARDOS:

1. In *kardos.c*: set $noOfEquations = 1$.

2. In $max.c/InitMax()$: the variables $u0$, $ux0$, $sourceS$ and others similar to them get allocated memory with the help of function $GetUMemMaxw$, since they have now three components.

3. The choice and organization of the basis functions will be explained in Section 2.2.

4. The dofs u_i are not anymore defined as values of the solution computed at node points. E.g., the edge dofs are given by

$$u_i = \int_{e_i} \boldsymbol{A} \cdot \boldsymbol{\tau}_{e_i} ds, \tag{2.4}$$

   where $\boldsymbol{\tau}_{e_i}$ is defined in (2.16).
   This implies that the following functions have to be modified:

   - In *triangutil.c*: *ValueAtPartner(...)* and similar functions.
   - In *timeinteg.c*: *InitAndStartValues(...), SetInitValuesOnNodes(...)* and similar functions.
   - In *tsetup.c*: *TimePreSets(), SetNewValuesOnNodes(...), RefTetrahedronTime(...).*

   But more on dofs in Section 2.4.

Now we proceed as usually in a Galerkin approach towards obtaining a linear system. We choose the discrete test functions $\boldsymbol{W}_N$ to be the functions $\{\boldsymbol{\Phi}_i : i = 1, \ldots, N\}$. Inserting (2.3) into (2.2), we obtain (after discretizing in time) a linear system written in the form

$$(C - kM)\boldsymbol{u} = B, \tag{2.5}$$

where $\boldsymbol{u} = (u_1, \ldots, u_N)^T$ and C, M and B are

$$C_{ij} = \int_\Omega \mu^{-1} \mathbf{curl}\, \boldsymbol{\Phi}_i \cdot \mathbf{curl}\, \boldsymbol{\Phi}_j \, d\boldsymbol{x}, \quad M_{ij} = \textstyle\int_\Omega \boldsymbol{\Phi}_i \cdot \boldsymbol{\Phi}_j \, d\boldsymbol{x},$$
$$B_i = \int_\Omega \boldsymbol{J}_s \cdot \boldsymbol{\Phi}_i \, d\boldsymbol{x}. \tag{2.6}$$

These entries are computed in *assmax.c*, but only locally, i.e., for $\Omega =$ tetrahedron t. As we will see in Section 2.2, the basis functions are given only on a reference tetrahedron and they have to be transformed on every tetrahedron t in the mesh. For this we need the *covariant transformation* [13]. Let $\hat{K}$ denote a reference element and K its image through a map F_K,

$$F_K : \hat{K} \to K, \quad F_K(\boldsymbol{\xi}) = \boldsymbol{x}, \, \forall \, \boldsymbol{\xi} \in \hat{K}, \, \boldsymbol{x} \in K.$$

In [13] it is proved that a vector function $\hat{\boldsymbol{u}} \in \boldsymbol{H}(\mathrm{curl}; \hat{K})$ is transformed to a vector function $\boldsymbol{u} \in \boldsymbol{H}(\mathrm{curl}; K)$ by the so-called *covariant transformation*

$$\boldsymbol{u} = JF_K^{-T} \cdot \hat{\boldsymbol{u}}, \tag{2.7}$$

and then it yields (see also [13], pg. 77-78)

$$[\nabla \times \boldsymbol{u}] = JF_K^{-T} [\hat{\nabla} \times \hat{\boldsymbol{u}}] JF_K^{-1}, \tag{2.8}$$

where the Jacobian matrix JF_K and the curl matrix $[\nabla \times \boldsymbol{u}]$ are defined by

$$[JF_K]_{ij} = \frac{\partial (F_K)_i}{\partial \xi_j}, \quad [\nabla \times \boldsymbol{u}]_{ij} = \frac{\partial \boldsymbol{u}_i}{\partial x_j} - \frac{\partial \boldsymbol{u}_j}{\partial x_i}, \quad 1 \le i, j \le 3. \tag{2.9}$$

In the particular case of an affine map of the form

$$\boldsymbol{x} = F_K(\boldsymbol{\xi}) = B_K \boldsymbol{\xi} + b_K, \tag{2.10}$$

where B_K is a constant matrix given by (using KARDOS notations)

$$B_k = \begin{pmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{pmatrix}, \quad or \quad B_k = \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix},$$

we have $JF_K = B_K$ and thus $\boldsymbol{u} = B_K^{-T} \hat{\boldsymbol{u}}$. In 2D, one may verify that

$$\mathrm{curl}\, \boldsymbol{u} = \frac{1}{\det B_K} \hat{\mathrm{curl}} \hat{\boldsymbol{u}}. \tag{2.11}$$

In 3D, in [13], pg. 78, Corr. 3.58, it is proved that

$$\mathbf{curl}\,\boldsymbol{u} = \frac{1}{\det B_K} B_K \cdot \hat{\mathbf{curl}}\,\hat{\boldsymbol{u}}. \tag{2.12}$$

See the Appendix for definition of curl and **curl**. Thus, in 2D, the entries in the curl-curl matrix C and in the mass matrix M take the following form:

$$\begin{aligned}
c_{ij} &= \int_K \mu^{-1} \operatorname{curl} \boldsymbol{\Phi}_i \operatorname{curl} \boldsymbol{\Phi}_j \, dxdy \\
&= \frac{1}{\det B_K} \int_{\hat{K}} \hat{\mu}^{-1} \left(\frac{\partial \Phi_i^2}{\partial \xi} - \frac{\partial \Phi_i^1}{\partial \eta}\right)\left(\frac{\partial \Phi_j^2}{\partial \xi} - \frac{\partial \Phi_j^1}{\partial \eta}\right) d\xi d\eta, \\
m_{ij} &= \int_K k \boldsymbol{\Phi}_i \cdot \boldsymbol{\Phi}_j \, dxdy = \det B_K \int_{\hat{K}} \hat{k} (B_K^{-T} \hat{\boldsymbol{\Phi}}_i) \cdot (B_K^{-T} \hat{\boldsymbol{\Phi}}_j) \, d\xi d\eta \\
&= \frac{1}{\det B_K} \int_{\hat{K}} \hat{k} \, [(f_{22}\Phi_i^1 - f_{21}\Phi_i^2)(f_{22}\Phi_j^1 - f_{21}\Phi_j^2) + \\
&\qquad + (f_{12}\Phi_i^1 - f_{11}\Phi_i^2)(f_{12}\Phi_j^1 - f_{11}\Phi_j^2)] \, d\xi d\eta.
\end{aligned}$$

These formulas appear in $assmax.c$, in e.g. $CompOpCurlMaxw(...)$. In 3D, we have obtained similar, but rather lengthy formulas. The case when μ^{-1} is a 3×3 matrix is also considered.

2.2 $\boldsymbol{H}$(curl)-conforming basis functions

The following description of the popularisation of edge elements is taken from [12]:
In 1980, Nedelec [14] gave the details of a recipe for the construction of $\boldsymbol{H}$(curl) *and* $\boldsymbol{H}$(div) *conforming elements on tetrahedra and hexahedra. He also proposed the conditions which these elements should satisfy in order for them to be regarded as conforming. More precisely, Nedelec [14] shows that if domains* K *und* K' *share a common face* f *with normal* $\boldsymbol{n}$*, then a smooth vector field* $\boldsymbol{u}$ *on each domains belongs to* $\boldsymbol{H}(\text{curl}; K \cup K')$ *provided that* $\boldsymbol{n} \times \boldsymbol{u}$ *is the same on each side of the face* f*. Thus a conforming discretization of* $\boldsymbol{H}$(curl) *is characterized by continuity of tangential components across element interfaces.*
He also proposed [15] a second family of $\boldsymbol{H}$(curl) *conforming elements, with basis functions requiring double the number of unknowns. The new family produced a better convergence rate for the vector, but an identical convergence*

rate for the curl, when compared to the original family. Although Nedelec proposed recipes for the construction of $\boldsymbol{H}(\text{curl})$ *conforming elements, there are many choices which can be adopted for the basis functions and all have their respective advantages and disadvantages. For example, certain choices may lead to matrices which are ill-conditioned and difficult to solve, while others may be difficult to implement in computer programs. Subsequently, in the 1980s and the 1999s, many different sets of basis functions have been proposed.*

In [12] may be also found a summary of these developments. Here we will mention only a few of them, namely the ones who gained more applicability and attention.

- In 1997, Graglia et al. [8] proposed a set of interpolatory basis functions of type I, i.e., without gradients. The Nedelec's $0th$ order vector basis functions multiplied by a complete interpolatory polynomial of order p generate the vector basis functions of pth order.
- In 2000, Demkowicz et al. pioneered the application of hp-adaptive finite element techniques to electromagnetics. They allow for locally variable order elements of type I, under the so-called *minimum rule*, i.e., the degree of polynomial on an edge or on a face is restricted to be equal to the order of the lowest polynomial employed on the elements which contain the edge, or contain the face. In 2D, they describe in [16] a complete finite element package which allowed for the use of hybrid quadrilateral/triangular meshes with simultaneous h and p-refinement. In 3D, they describe in [17] also a complete finite element package, but only for hexahedral elements. Their approach allows for hanging nodes and for anisotropic refinement. They describe a data structure which allow such refinement, although they do not have yet a strategy for the correct selection of the combination of h and p-refinements. Curved elements are allowed through the definition of a reference element and a geometric mapping.
- In 2001, Ainsworth and Coyle [2] have proposed an alternative hierarchical basis of type II for triangles and quadrilaterals which has certain advantages over the basis proposed by Demkowicz. For example, numerical results for triangular and quadrilateral discretisations [2] show that the basis proposed by Ainsworth and Coyle has superior conditioning properties ([12]). Also in 3D, they proposed an alternative hierarchical basis for hexahedral elements, for which they also give theoretical estimates which bound the condition number of the mass and stiffness matrices for arbitrary order p, and a hierarchical basis on tetra-

hedral meshes [1]. Since KARDOS is based on tetrahedral meshes, we choose the basis described in [1] to work with and we will shortly give more details on it.

- The most recent *hp*-finite element package based on a new hierarchical tetrahedral element is developed by J. Schoeberl et al. [20]. The new shape functions provide not only the global complete sequence property, but also local complete sequence property for each edge, face, element-block. This local property allows an arbitrary and variable choice of the polynomial degree for each edge, face and element, without any minimal order conditions. Another advantage underlined in [20] is that the gradient shape functions are contained explicitely, such that they can be skipped when needed (e.g. magnetostatic bvp). Furthermore, the simple block-diagonal preconditioner gets efficient.

Now we focus on our choice of a hierarchical tetrahedral basis, namely the one given by Ainsworth and Coyle [1]. The vector-valued polynomial space of order p on the reference element $\hat{t}$ is given by

$$\hat{\boldsymbol{X}}_p^{curl} = (\mathcal{P}(\hat{t}))^3.$$

The set of hierarchical basis functions for $\hat{\boldsymbol{X}}_p^{curl}$ is given in Table 2.1 [1].

The reference tetrahedron $\hat{t}$ used by Ainsworth and Coyle is represented in Figure 2.1. Its edges are all of the same length, in contrast to the reference tetrahedron used up to now in KARDOS, see Figure 2.2. We may either transform the basis functions on our tetrahedron or change the reference element with the equilateral one. The latter choice implies modifications on the (i) integration rules, (ii) the transformation from a tetrahedron in the mesh to the reference tetrahedron, and (ii) thus (since the transformation map is not explicitly defined in KARDOS) the functions in *assmax.c, triangutil.c.* In the new 3D-version of KARDOS, this equilateral reference tetrahedron is introduced, together with the necessary modifications.

In KARDOS, the basis functions are organized as follows:

- for $p = 0$ in function *NedVecSShape(...)*,
- for $p = 1$ in function *NedVecLShape(...)*,
- for $p = 2$ in function *NedVecQShape(...)*,
- for $p = 3$ in function *NedVecCShape(...)*.

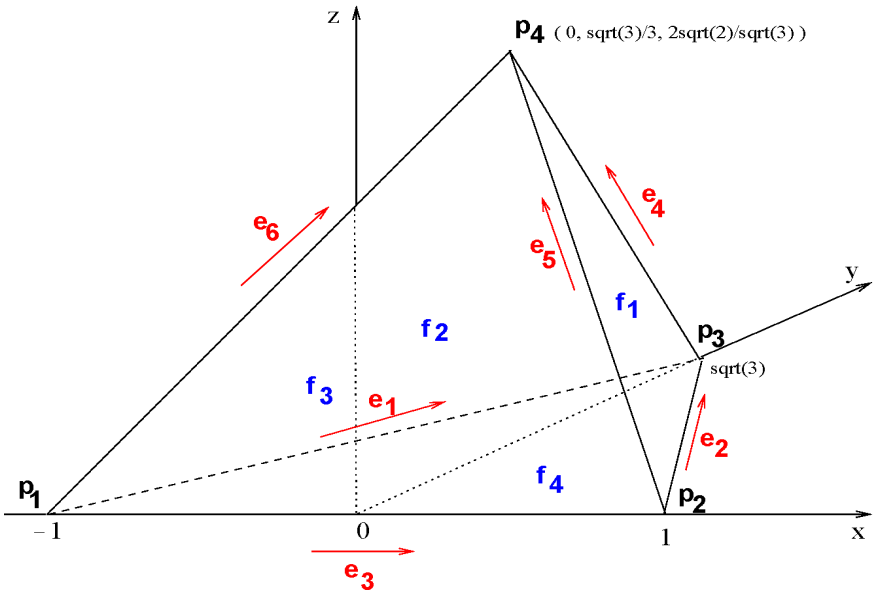

Figure 2.1: The equilateral reference tetrahedron [1] used in KARDOS for H(curl) discretizations

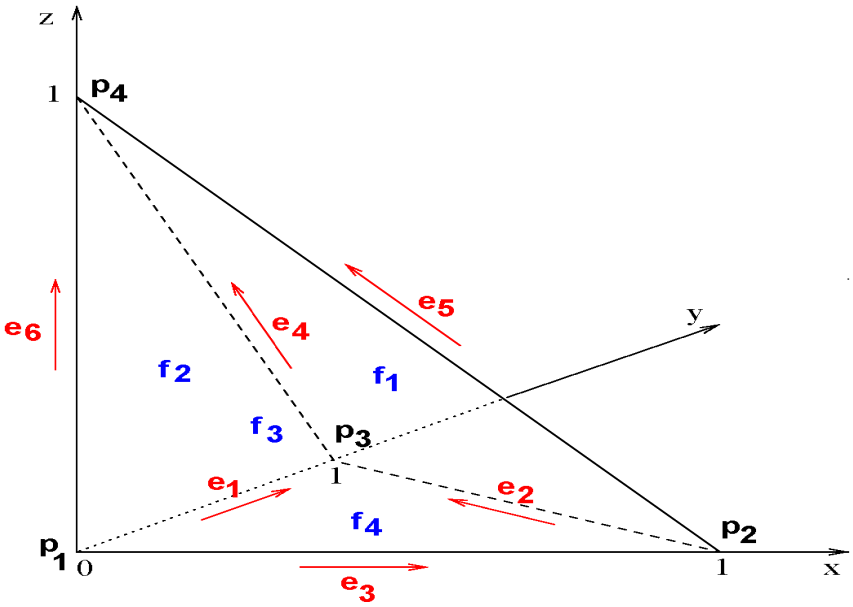

Figure 2.2: The reference tetrahedron used in KARDOS for Lagrange discretizations

In each function NedVec-Shape(...), the vector valued basis functions are

Space: $(\mathcal{P}(\hat{t}))^3$, $p \in \mathbb{N}$

Edge functions: $\boldsymbol{e} = [oi] \in \mathcal{E}(\hat{t})$

$$\hat{\boldsymbol{\phi}}_0^{\boldsymbol{e}} = \hat{\lambda}_i \hat{\boldsymbol{\nabla}} \hat{\lambda}_o - \hat{\lambda}_o \hat{\boldsymbol{\nabla}} \hat{\lambda}_i$$

$$\hat{\boldsymbol{\phi}}_1^{\boldsymbol{e}} = \hat{\lambda}_i \hat{\boldsymbol{\nabla}} \hat{\lambda}_o + \hat{\lambda}_o \hat{\boldsymbol{\nabla}} \hat{\lambda}_i$$

$$\hat{\boldsymbol{\phi}}_{l+1}^{\boldsymbol{e}} = \tfrac{2l+1}{l+1} L_l(\hat{\xi}_{oi}) \hat{\boldsymbol{\phi}}_1^{\boldsymbol{e}} - \tfrac{l}{l+1} L_{l-1}(\hat{\xi}_{oi}) \hat{\boldsymbol{\phi}}_0^{\boldsymbol{e}}, \quad 1 \le l \le p-1$$

Edge-based face functions: $\boldsymbol{f} \in \mathcal{F}(\hat{t})$

For each edge $\boldsymbol{e} \in \partial \boldsymbol{f}$:

$$\hat{\boldsymbol{\phi}}_{\boldsymbol{e},l}^{\boldsymbol{f}} = \beta_{\boldsymbol{e}} \psi_l(\hat{\xi}_{\boldsymbol{e}}) \hat{\boldsymbol{\nabla}} \hat{\lambda}_{\boldsymbol{f}\setminus\boldsymbol{e}}, \quad 0 \le l \le p-2,$$

where $\boldsymbol{f}\setminus\boldsymbol{e}$ denotes the vertex opposite edge $\boldsymbol{e}$ in face $\boldsymbol{f}$.

Face bubble functions: $\boldsymbol{f} = [oij] \in \mathcal{F}(\hat{t})$

$$\hat{\boldsymbol{\phi}}_{i,lm}^{\boldsymbol{f}} = \beta_{oij} \psi_l(\hat{\xi}_{oi}) \psi_m(\hat{\xi}_{oj}) \hat{\boldsymbol{\tau}}^{[oi]},$$

$$\hat{\boldsymbol{\phi}}_{j,lm}^{\boldsymbol{f}} = \beta_{oij} \psi_l(\hat{\xi}_{oi}) \psi_m(\hat{\xi}_{oj}) \hat{\boldsymbol{\tau}}^{[oj]}, \quad 0 \le l, m, l+m \le p-3$$

Face-based interior functions: $\boldsymbol{f} = [oij] \in \mathcal{F}(\hat{t})$

$$\hat{\boldsymbol{\phi}}_{\boldsymbol{f},lm}^{\hat{\boldsymbol{t}}} = \beta_{\boldsymbol{f}} \psi_l(\hat{\xi}_{oi}) \psi_m(\hat{\xi}_{oj}) \hat{\boldsymbol{\nabla}} \hat{\lambda}_{\boldsymbol{t}\setminus\boldsymbol{f}}, \quad 0 \le l, m, l+m \le p-3$$

where $\boldsymbol{t}\setminus\boldsymbol{f}$ denotes the vertex of $\hat{\boldsymbol{t}}$ opposite face $\boldsymbol{f}$.

Interior bubble functions: $\hat{\boldsymbol{t}} = [oijk]$

$$\hat{\boldsymbol{\phi}}_{d,lmn}^{\hat{\boldsymbol{t}}} = \beta_{\boldsymbol{t}} \psi_l(\hat{\xi}_{oi}) \psi_m(\hat{\xi}_{oj}) \psi_n(\hat{\xi}_{ok}) \hat{\boldsymbol{e}}_d,$$

$d \in \{1,2,3\}$, $0 \le l, m, n, l+m+n \le p-4$.

Table 2.1: Hierarchic basis functions for $\boldsymbol{H}$(curl)-conforming FE space of order p [1].

listed component by component, i.e., in 3D, the first three functions (cases 0, 1, 2) form the first vector-valued basis function, the next three form the second one and so on, and in the following order:

a) Firstly, the edge functions, edge by edge, i.e., firstly all edge functions associated with edge $\hat{e}_1$, then those with $\hat{e}_2$ and so on.

b) Then the face functions, face by face. These are of two types: (i) edge-based face functions, listed edge by edge wrt. a face, (ii) face bubble functions (only for $p \ge 3$).

c) Finally, the interior functions (only for $p \geq 3$). These are also of two types: (i) face-based interior functions, listed face by face wrt. a tetrahedron, (ii) interior bubble functions (only for $p \geq 4$).

Up to now we have only considered $p \leq 3$. The basis functions computation may be found in *ksrc/nedshape.c*. To get an idea how it looks like in KARDOS, we list below a few functions from *ksrc/nedshape.c*:

```
static double coordx[4]={-1.0, 1.0, 0.0, 0.0},
              coordy[4]={0.0, 0.0, SQRT3, SQRT3/3.},
              coordz[4]={0.0, 0.0, 0.0, 2.*SQRT2/SQRT3};

static int edges[6][2]={{1, 3},{2, 3},{1, 2},{3, 4},{2, 4},{1, 4}};
static int faces[4][3]={{2, 3, 4},{1, 3, 4},{1, 2, 4},{1, 2, 3}};

static void Lambda(real x, real y, real z, int no, real *f)
{
 switch (no)
 {
 case 1:
  *f=0.5*(1.0-x-y*sqrt(3.)/3.-z*sqrt(6.)/6.);
  break;
 case 2:
  *f=0.5*(1.0+x-y*sqrt(3.)/3.-z*sqrt(6.)/6.);
  break;
 case 3:
  *f=y*sqrt(3.)/3.-z*sqrt(6.)/12.;
  break;
 case 4:
  *f=z*sqrt(6.)/4.;
  break;
 }
 return;
}

static void GradLambda(int no, real *fx, real *fy, real *fz)
{
 switch (no)
 {
 case 1:
  *fx=-0.5;
  *fy=-0.5*sqrt(3.)/3.;
  *fz=-0.5*sqrt(6.)/6.;
  break;
 case 2:
  *fx=0.5;
  *fy=-0.5*sqrt(3.)/3.;
  *fz=-0.5*sqrt(6.)/6.;
```

```
  break;
 case 3:
  *fx=0.;
  *fy=sqrt(3.)/3.;
  *fz=-sqrt(6.)/12.;
  break;
 case 4:
  *fx=0.;
  *fy=0.;
  *fz=sqrt(6.)/4.;
  break;
 }
 return;
}

static void EdgeShapep0(real x, real y, real z, int p1, int p2,
                        real *f1, real *f2, real *f3)
{
 real l1, l2, l1x, l1y, l1z, l2x, l2y, l2z;

 Lambda(x, y, z, p1, &l1);
 GradLambda(p1, &l1x, &l1y, &l1z);
 Lambda(x, y, z, p2, &l2);
 GradLambda(p2, &l2x, &l2y, &l2z);

 *f1=(l2*l1x-l1*l2x);
 *f2=(l2*l1y-l1*l2y);
 *f3=(l2*l1z-l1*l2z);

 return;
}

static void JacEdgeShapep0(real x, real y, real z, int p1, int p2,
                           real *f1x, real *f1y, real *f1z,
                           real *f2x, real *f2y, real *f2z,
                           real *f3x, real *f3y, real *f3z)
{
 real l1x, l1y, l1z, l2x, l2y, l2z;

 GradLambda(p1, &l1x, &l1y, &l1z);
 GradLambda(p2, &l2x, &l2y, &l2z);

 *f1x=(l2x*l1x-l1x*l2x); *f1y=(l2y*l1x-l1y*l2x); *f1z=(l2z*l1x-l1z*l2x);
 *f2x=(l2x*l1y-l1x*l2y); *f2y=(l2y*l1y-l1y*l2y); *f2z=(l2z*l1y-l1z*l2y);
 *f3x=(l2x*l1z-l1x*l2z); *f3y=(l2y*l1z-l1y*l2z); *f3z=(l2z*l1z-l1z*l2z);

 return;
}
```

2.3 The problem of enforcing conformity

Up to now, in KARDOS (classical FEM) the so-called *sign conflict problem* was solved in *nodes.c, GetNodeAddresses(...)*, where the addresses of the nodes are called in the right order, i.e., if the numbering of an element does not coincide with that of the reference element, then the order is changed such that it coincides. But this is a consequence of the definition of Lagrange basis functions and of nodal degrees of freedom, and it does not apply to the edge FEM. In the classical FEM, a basis function and the corresponding degree of freedom are associated to a point in the element.

The *sign conflict problem* for edge elements is discussed, e.g., in [1], where a strategy based on considering two types of reference tetrahedra is presented. In [11], for the 2D case, a very simple solution to this problem is proposed, namely: assign a global direction to each edge in the mesh and multiply those pth order local edge degrees of freedom whose local orientation is different to the global one by the factor $(-1)^{p+1}$. The tangential component of the interior basis functions vanishes on the edges of the element and so they do not contribute to the continuity requirements. The factor $(-1)^{p+1}$ comes from the following property of the edge basis functions:

$$\boldsymbol{\tau}_e \cdot \boldsymbol{\Phi}_p^e|_e = L_p(\xi_e), \quad p \neq 1, \boldsymbol{\tau}_e \cdot \boldsymbol{\Phi}_1^e|_e = -L_1(\xi_e), \tag{2.13}$$

where ξ_e is a parametrization for the edge e, $\boldsymbol{\tau}_e$ is the unit tangent vector to e, $\boldsymbol{\Phi}_p^e|_e$ is the pth order edge basis function on the edge e, and L_p is the Legendre polynomial of degree p with the property

$$L_p(\xi_l) = (-1)^p L_p(\xi_r), \tag{2.14}$$

where ξ_l and ξ_r are equal and opposite parametrizations for the edge on the neighbouring elements.

We apply the same procedure to solve the sign conflict problem for edge dofs also in 3D. More precisely, we have implemented the following:

1. In *nodes.c/GetNodeAddresses(...)*:

```
for (k=j*noOfEdgElem; k<(j+1)*noOfEdgElem; k++)
 {
   l = (nodesState->startEdgeNodes)[k];
   adr[i++] = (t->e1)->vec+l;
 }
```

```
for (k=(j+1)*noOfEdgElem-1; k>=j*noOfEdgElem; k--)
 {
   l = (nodesState->startEdgeNodes)[k];
   adr[i++] = (t->e1)->vec+l;
 }
```

2. Each edge e has a global orientation, namely from the point $e \to p1$ to $e \to p2$.

3. The local orientation of an edge e in the tetrahedron t is computed w.r.t. the reference tetrahedron, which has a prescribed (fixed) edge orientation as illustrated in Fig. 2.1 or 2.2.
   This is set in *assmaxw.c/OpenMaxwAss(...):*

```
/* edge orientation: 1.0 or -1.0 (double)*/
    ((*lData)->orientEdges)[0] = Orient_Edge(t->p1, t->e1);
    ((*lData)->orientEdges)[1] = Orient_Edge(t->p2, t->e2);
    ((*lData)->orientEdges)[2] = Orient_Edge(t->p1, t->e3);
    ((*lData)->orientEdges)[3] = Orient_Edge(t->p3, t->e4);
    ((*lData)->orientEdges)[4] = Orient_Edge(t->p2, t->e5);
    ((*lData)->orientEdges)[5] = Orient_Edge(t->p1, t->e6);
```

4. The factor $orientation^{p+1}$, where $orientation = 1$ or -1, is computed in *triangutil.c/Sign(TD *t, int k, int maxp)* and the function *Sign(...)* is used in *assmaxw.c, triangutil.c* and *maxwell.c.*

For $p = 2$, there are three face dofs per face and one may associate each face dof with a vertex point from the corresponding face. Thus, in this case it is possible to solve the orientation problem by rotating the face dofs according to the ordering of the vertices in the corresponding face. This is done in *nodes.c*, GetNodeAddresses(...), analogously to the Lagrange case.

Remarks:
1. In this way, for $p \leq 2$ we could solve the sign conflict problem using only one reference tetrahedron, in contrast to [1] where two reference tetrahedra are proposed.

For completeness, we will give now the solution to the orientation problem proposed in [1]. Denote the set of vertices, edges, and faces of a single element $\boldsymbol{t}$ by $\mathcal{V}(\boldsymbol{t})$, $\mathcal{E}(\boldsymbol{t})$, and $\mathcal{F}(\boldsymbol{t})$, respectively, and the corresponding global sets by $\mathcal{V}$, $\mathcal{E}$, and $\mathcal{F}$. Then we have the following.

- Each edge $\boldsymbol{e} \in \mathcal{E}$ is described by a pair $[oi]$ of numbers for the global vertices located at the endpoints of the edge. The ordering of the numbers is determined by requiring

$$o < i.$$

The edge may be then assigned a unique parametrisation given by

$$\xi_{\boldsymbol{e}} : R^3 -> R \quad : \xi_{\boldsymbol{e}} = \xi_{oi} = \lambda_i - \lambda_o. \tag{2.15}$$

The parametrisation is intrinsic to the edge, depending solely on the global numbering of the endpoints of the edge. Furthermore, the tangent vector $\boldsymbol{\tau_e}$ to $\boldsymbol{e}$ is defined by

$$\boldsymbol{\tau_e} = \boldsymbol{p}_i - \boldsymbol{p}_o. \tag{2.16}$$

- A face $\boldsymbol{f} \in \mathcal{F}$ is described by a triple $[oij]$ formed from the global numbering of the vertices of the face. By applying a rotation of the local numbering of the vertices, it is possible to ensure that the vertex $\boldsymbol{v}_o$ (with the smallest global numbering) has local number 1, i.e., it is on the first position. Thus, we need to ensure that for each face

$$o < i \quad and \quad o < j.$$

The face is then assigned a unique parametrisation given by ξ_{oi}, ξ_{oj}.

- Likewise, a tetrahedron $\boldsymbol{t} = [oijk] \in \mathcal{T}$ is parametrised by ξ_{oi}, ξ_{oj} and ξ_{ok}, where it is assumed that

$$o < i, j < k.$$

The relation between i and j is either $i < j$ or $j < i$ and must be maintained during the orientation process. The proposed orientation process leads to the conclusion that any tetrahedron $\boldsymbol{t}$ can be oriented in one of only two ways (either with $i < j$ or with $j < i$), and this calls for the definition of *two reference elements*. This process consists in an appropriate reordering of the local numbering of the vertices:

1. First bring o (i.e. the smallest global vertex) on the first position by rotating the local numbering on a face (of the two) containing o and the local vertex 1 (i.e., the vertex which seats on the first position, instead of o). Of course, if the two are already aligned, then no action is needed.

2. Bring k (i.e. the largest global vertex) on the last position by rotating the local numbering on the face opposite local vertex 1 (i.e. the last 3 numbers).

3. Decide of which type is the tetrahedron. If $i < j$, then is of type I, and otherwise of type II. See Figure 2.3.

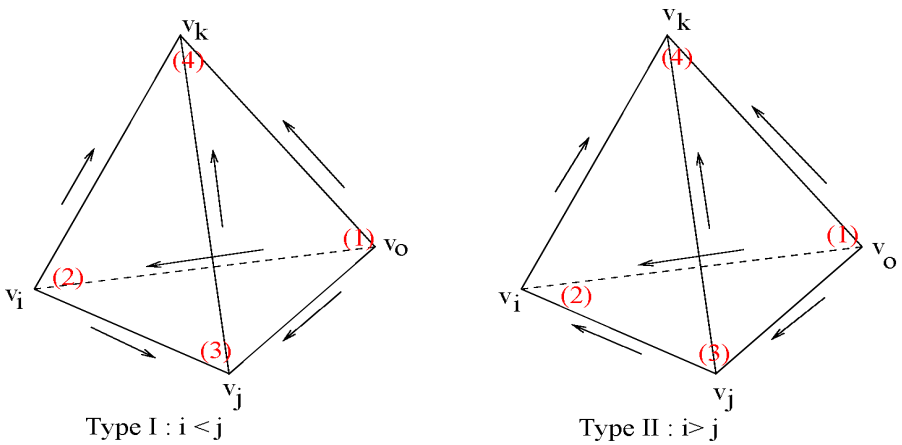

Figure 2.3: The two possible reference tetrahedra. The arrows indicate the global orientation of the edges. The tetrahedra differ only in the orientation of the edge connecting vertices v_i and v_j, which depends on the ordering of the global numbers i and j

Example:
As an example, consider a tetrahedron with global numbering [15 96 8 24]. The above procedure applied to this particular tetrahedron gives:
1. the triple [15 96 8] is cyclically permuted until the 8 is the first. This yields [8 15 96] and gives the new local to global numbering [8 15 96 24] for the tetrahedron.
2. rotate [15 96 24] until 96 becomes the last. This yields [24 15 96] and the new numbering for the tetrahedron is [8 24 15 96]. **3**. since $24 > 15$, the tetrahedron is classified as being of type II.

Remarks:
1. In practice, the reduction to a type I or II reference element is performed as a mesh post-processing step.
2. Once the tetrahedra have been classified as type I or II, there is no more the need to check orientations. This means that the $(-1)^{p+1}$ rule for

the edges is no longer needed. Also for the face orientation we don't have anymore to check, since very edge and every face of the appropriate reference configuration is identical with the intrinsic orientations of the edges and faces viewed in isolation.

2.4 Managing the degrees of freedom

Let us firstly remember the definition of a finite element, which is the basis of a FEM. A *finite element* is a triple $\{T, K, \Sigma\}$ where

- T is e.g. a tetrahedron.
- X is a polynomial space of element shape functions of dimemsion N.
- $\Sigma = \{\Psi_i : V \supseteq X \to R,\, i = 1, \ldots, N\}$ is a space of linear functionals (the *degrees of freedom*), dual to X, i.e.,

$$\Psi_i(\boldsymbol{\Phi}_j) = \delta_{ij}, \tag{2.17}$$

  where $\{\boldsymbol{\Phi}_j\}_{j=1}^N$ is a basis of X.

The corresponding interpolation operator $\Pi : V \to X$ is defined as

$$\Pi\, \boldsymbol{u} = \sum_{i=1}^{N} \Psi_i(\boldsymbol{u}) \boldsymbol{\Phi}_i. \tag{2.18}$$

As seen in Sect. 2.2, the basis functions are of three types: edge, face, and interior functions. Thus, the dofs are also of three types. In [14] the following expressions for the dofs are given:

- $6(p+1)$ edge dofs of the form

$$\Psi_e^{\varphi}(\boldsymbol{u}) := \int_e (\boldsymbol{u} \cdot \boldsymbol{\tau}) \varphi \, ds, \quad \forall \varphi \in P_p(e), \quad \forall \text{ edges } e \tag{2.19}$$

- $4(p-1)(p+1)$ face dofs of the form

$$\Psi_f^{\boldsymbol{q}}(\boldsymbol{u}) := \int_f (\boldsymbol{u} \cdot \boldsymbol{q}) \, d\gamma, \quad \forall \boldsymbol{q} \in \mathcal{D}_{p-1}(f), \quad \forall \text{ faces } f \tag{2.20}$$

- $(p-2)(p-1)(p+1)/2$ interior dofs of the form

$$\Psi_t^{\boldsymbol{q}}(\boldsymbol{u}) := \int_t (\boldsymbol{u} \cdot \boldsymbol{q}) \, d\boldsymbol{x}, \quad \forall \boldsymbol{q} \in \mathcal{D}_{p-2}(t), \tag{2.21}$$

where $\mathcal{D}_k = (P_{k-1})^3 \oplus \tilde{P}_{k-1} \cdot \boldsymbol{x}$, and $\tilde{P}_k$ is the space of homogeneous polynomials of total degree exactly k.

Remark:
1. The problem with the above expressions is that the dofs are not uniquely defined until the test polynomials φ and $\boldsymbol{q}$ in the above formulas are determined. They depend on the chosen basis functions, since we have to satisfy condition (2.17).

We shall try another approach. One may write, $\forall \tilde{\boldsymbol{u}} \in (P_3(\hat{t}))^3$, $\forall \boldsymbol{x} \in \hat{t}$:

$$\begin{aligned}\tilde{\boldsymbol{u}}(\boldsymbol{x}) &= \sum_{i=1}^{6}\sum_{k=0}^{3} u_{e_i}^k \boldsymbol{\Phi}_{e_i}^k(\boldsymbol{x}) + \\ &+ \sum_{j=1}^{4}\sum_{i=1}^{3}\sum_{k=0}^{1} u_{e_i,k}^{f_j} \boldsymbol{\Phi}_{e_i,k}^{f_j}(\boldsymbol{x}) + \sum_{j=1}^{4}\{u_{p_2,00}^{f_j}\boldsymbol{\Phi}_{p_2,00}^{f_j}(\boldsymbol{x}) + u_{p_3,00}^{f_j}\boldsymbol{\Phi}_{p_3,00}^{f_j}(\boldsymbol{x})\} + \\ &+ \sum_{j=1}^{4} u_{f_j,00}^{t}\boldsymbol{\Phi}_{f_j,00}^{t}(\boldsymbol{x}), \qquad (2.22)\end{aligned}$$

where $u_{e_i}^k$ denote the edge dofs, $u_{e_i,k}^{f_j}$, $u_{p,00}^{f_j}$ the face dofs and $u_{f_j,00}^t$ the interior dofs, and $\boldsymbol{\Phi}_{e_i}^k$, $\boldsymbol{\Phi}_{e_i,k}^{f_j}$, $\boldsymbol{\Phi}_{p,00}^{f_j}$, $\boldsymbol{\Phi}_{f_j,00}^t$ are the basis functions from Sect. 2.2. Now we have to solve the following problem: assuming that in (2.22) the function $\tilde{\boldsymbol{u}}(\boldsymbol{x})$ is given, find the dofs. This problem is addressed also in [12], pg. 18, in the following way:

Edge dofs:
Multiplying (2.22) with the tangent vector $\boldsymbol{\tau}_{e_j}$, where e_j is an edge of $\hat{t}$, and using the following properties of the basis functions

$$\begin{aligned}&\boldsymbol{\Phi}_{e_i}^k|_{e_j} \cdot \boldsymbol{\tau}_{e_j} = 0, \qquad \text{if } i \neq j \\ &\boldsymbol{\Phi}_{e_i,k}^{f_j}|_e \cdot \boldsymbol{\tau}_e = 0, \qquad \boldsymbol{\Phi}_{p,00}^{f_j}|_e \cdot \boldsymbol{\tau}_e = 0, \qquad \forall e \\ &\boldsymbol{\Phi}_{f_j,00}^t|_e \cdot \boldsymbol{\tau}_e = 0, \qquad \forall e\end{aligned}$$

(i.e. the tangential components of the face and interior functions vanish on all edges) one obtains

$$\tilde{\boldsymbol{u}} \cdot \boldsymbol{\tau}_{e_j} = \sum_{k=0}^{3} u_{e_j}^k \boldsymbol{\Phi}_{e_j}^k \cdot \boldsymbol{\tau}_{e_j} \text{ on edge } e_j$$

which implies

$$\left(\sum_{k=0}^{3} u_{e_j}^k \boldsymbol{\Phi}_{e_j}^k \cdot \boldsymbol{\tau}_{e_j},\, \boldsymbol{\Phi}_{e_j}^l \cdot \boldsymbol{\tau}_{e_j}\right)_{e_j} = \left(\tilde{\boldsymbol{u}} \cdot \boldsymbol{\tau}_{e_j},\, \boldsymbol{\Phi}_{e_j}^l \cdot \boldsymbol{\tau}_{e_j}\right)_{e_j}, \tag{2.23}$$

for e_j edge of $\hat{t}$ and $l = 0, ..., 3$. Thus, for each edge e_j of $\hat{t}$, it is obtained a system with four unknowns ($u_{e_j}^k$, $k = 0, ..., 3$) and four equations.

Remarks:
1. Using the property (2.13) and the orthogonality of Legendre polynomials

$$\int_{-1}^{1} L_k(\xi) L_l(\xi) d\xi = \frac{2}{2k+1}\delta_{kl}, \tag{2.24}$$

we observe that system (2.23) is diagonal, i.e., the edge dofs may be directly computed

$$u_{e_j}^k = \begin{cases} \frac{2k+1}{2} \int_{e_j} \tilde{\boldsymbol{u}} \cdot \boldsymbol{\tau}_{e_j} L_k(\xi_{e_j}) d\xi_{e_j}, & k = 0, 2, 3, \\ -\frac{3}{2} \int_{e_j} \tilde{\boldsymbol{u}} \cdot \boldsymbol{\tau}_{e_j} L_1(\xi_{e_j}) d\xi_{e_j}, & k = 1, \end{cases} \tag{2.25}$$

However, we solve system (2.23) to obtain the edge dofs, because this is a more general approach, i.e., could be applied also for another set of basis functions.

Face dofs:
For each face f of $\hat{t}$, let us consider the outer normal vector $\boldsymbol{n}_f$ and multiply (2.22) with it. On the face f we obtain

$$\begin{aligned} \tilde{\boldsymbol{u}} \times \boldsymbol{n}_f = \; & \textstyle\sum_{e_i \in \partial f} \sum_{k=0}^{3} u_{e_i}^k \boldsymbol{\Phi}_{e_i}^k \times \boldsymbol{n}_f + \sum_{e_i \in \partial f} \sum_{k=0}^{1} u_{e_i,k}^f \boldsymbol{\Phi}_{e_i,k}^f \times \boldsymbol{n}_f + \\ & + u_{p_2,00}^f \boldsymbol{\Phi}_{p_2,00}^f \times \boldsymbol{n}_f + u_{p_3,00}^f \boldsymbol{\Phi}_{p_3,00}^f \times \boldsymbol{n}_f, \end{aligned} \tag{2.26}$$

since the tangential components of interior functions vanish on all faces. Then one may write for each face f a system of six equations and six unknowns $u_{e_i,k}^f$, $e_i \in \partial f$, $i = 1, 2, 3$, $k = 0, 1$):

$$\begin{aligned} \sum_{e_i \in \partial f} \sum_{k=0}^{1} u_{e_i,k}^f \left(\boldsymbol{\Phi}_{e_i,k}^f \times \boldsymbol{n}_f,\, \boldsymbol{\Phi}_{e_j,k}^f \times \boldsymbol{n}_f\right)_f + \cancel{\sum_{p} u_{p,00}^f \left(\boldsymbol{\Phi}_{p,00}^f \times \boldsymbol{n}_f, \boldsymbol{\Phi}_{e_j,k}^f \times \boldsymbol{n}_f\right)_f} \; = \\ = \left(\left(\tilde{\boldsymbol{u}} - \sum_{e_i \in \partial f} \sum_{k=0}^{3} u_{e_i}^k \boldsymbol{\Phi}_{e_i}^k\right) \times \boldsymbol{n}_f,\, \boldsymbol{\Phi}_{e_j,k}^f \times \boldsymbol{n}_f\right)_f, \end{aligned} \tag{2.27}$$

since one may verify that the second term in the left hand side vanishes.
To obtain the face bubble coefficients, we multiply (2.26) with $\mathbf{\Phi}^f_{p,00} \times \boldsymbol{n}_f$, $p = p_2$ or p_3, and integrate over f, obtaining, for each face f, a system with 2 equations and 2 unknowns ($u^f_{p,00}$):

$$\sum_{p\in\{p_2,p_3\}} u^f_{p,00}\Big(\mathbf{\Phi}^f_{p,00}\times\boldsymbol{n}_f,\ \mathbf{\Phi}^f_{q,00}\times\boldsymbol{n}_f\Big)_f =$$

$$= \left(\left(\tilde{\boldsymbol{u}} - \sum_{e_i\in\partial f}\sum_{k=0}^{3} u^k_{e_i}\mathbf{\Phi}^k_{e_i} - \sum_{e_i\in\partial f}\sum_{k=0}^{1} u^f_{e_i,k}\mathbf{\Phi}^f_{e_i,k}\right)\times\boldsymbol{n}_f,\ \mathbf{\Phi}^f_{q,00}\times\boldsymbol{n}_f\right)_f, \quad (2.28)$$

Interior dofs:
Solve the following system with four equations and four unknowns $u^t_{f_j,00}$:

$$\sum_{j=1}^{4} u^t_{f_j,00}\left(\mathbf{\Phi}^t_{f_j,00}\cdot\boldsymbol{n}_{f_i},\ \mathbf{\Phi}^t_{f_i,00}\cdot\boldsymbol{n}_{f_i}\right)_t =$$

$$= \left(\left(\tilde{\boldsymbol{u}} - \sum_{e,k} u^k_e\mathbf{\Phi}^k_e - \sum_{j,e,k}^{f_j}\mathbf{\Phi}^{f_j}_{e,k} - \sum_{j,p} u^{f_j}_{p,00}\mathbf{\Phi}^{f_j}_{p,00}\right)\cdot\boldsymbol{n}_{f_i},\ \mathbf{\Phi}^t_{f_i,00}\cdot\boldsymbol{n}_{f_i}\right)_t, \quad (2.29)$$

for $i = 1, ..., 4$.

Thus, in *timeintegmaxw.c*, instead of the functions *ValueAtPartner(...), ValueAtTetra(...)* and similar to them from *triangutil.c*, we have introduced the following functions

- *EdgeValuesMaxw(TD *t, int e, whichTet tet, int index, real *edgedofs)*, which for each edge of a tetrahedron assembles and solves a system (2.23) using the function *DirectGauss(locA, noE, locB, edgedofs)*.
- *FaceEdgeValuesMaxw(TD *t, int f, whichTet tet, int index, int indexEdge, real *faceedgedofs)*, which for each face of a tetrahedron assembles and solves a system (2.27) using also the *DirectGauss* function.
- *FaceBubbleValuesMaxw(TD *t, int f, whichTet tet, int index, int indexEdgeFace, real *facebubbledofs)*, which for each face of a tetrahedron assembles and solves a system (2.28) using also the *DirectGauss* function.
- *InteriorValuesMaxw(TD *t, whichTet tet, int index, int indexEdgeFaceBubble, real *intdofs)*, which for each tetrahedron assembles and solves a system (2.29) using also the *DirectGauss* function.

Remarks:
1. The parameter *tet* of type *which Tet* shows from which mesh the solution is computed. This way, e.g., the function *EdgeValuesMaxw(...)* plays the role of the function *EdgeValuesAtPartnerMaxw(...)*, *EdgeValuesAtTetraMaxw(...)* or *InitAndStartValuesMaxw(...)*.

In *timeintegmaxw.c*, function *SetInitValuesOnNodes(TD *td)* is modified as follows:

```
int SetInitValuesOnNodesMaxw(TD *td)
 {
   int i, j, p, e, f=0, fi, ke, ef, kef;
   int no, noP, noE, noT, noTE, noTB, noTEperEdge, noTd,
       noOfEqu, nodesOfTetVar, noOfStages;
   int varUt = theTimePrbStruct->varUt;
   char *adr[400];
   REAL x, y, z;

   noP = nodesState->noOfPointNodes;
   noE = nodesState->noOfEdgeNodes;
   noT = nodesState->noOfTriangleNodes;
   noTE = 3*(noE-2);
   noTB = noT-noTE;
   noTEperEdge = noTE/3;
   noTd = nodesState->noOfTetrahedronNodes;
   noOfEqu = nodesState->noOfEquations;
   noOfStages = actTimeIntegtor->noOfStages;
   nodesOfTetVar = ( 4*(noP+noT) + 6*noE + noTd )/noOfEqu;
   no = GetNodeAddresses(td,adr,400);

  if (eVecInit==nil) if (!GetVecsMaxw()) return false;

  i=0;
  for (e=0;e<6;e++)
   {
    EdgeValuesMaxw(td, e, fromInit, rInit, eVecInit);
    if (varUt)
 EdgeValuesMaxw(td,e,fromInitUt,rInitUt,eVecInitUt);

     switch (timeState->stepTypePar)
  {
   case START_STEP:
     for (p=0;p<noOfStages;p++)
     for (j=0; j<noE; j++) eMatStage[p][j] = 0.0;
     break;
   case LATER_STEP:
     for (p=0;p<noOfStages;p++)
      EdgeValuesMaxw(td, e, fromPartner,
```

```
                        rStage[p], eMatStage[p]);
   break;
}
   for (ke=0; ke<noE; ke++)
{
     RV(adr[i],rInit) = eVecInit[ke];
 if (varUt) RV(adr[i],rInitUt) = eVecInitUt[ke];
     for (p=0;p<noOfStages;p++)
  RV(adr[i],rStage[p]) = eMatStage[p][ke];
 i++;
}
  }

if (noTE>0)
 for(f=0; f<4; f++)
  {
    FaceEdgeValuesMaxw(td, f, fromInit, rInit,
                        rInit, feVecInit);
    if (varUt)
    FaceEdgeValuesMaxw(td, f, fromInitUt, rInitUt,
                        rInitUt, feVecInitUt);

    switch (timeState->stepTypePar)
     {
      case START_STEP:
    for (p=0;p<noOfStages;p++)
     for (j=0; j<noTE; j++)
      feMatStage[p][j] = 0.0;
    break;
      case LATER_STEP:
    for (p=0;p<noOfStages;p++)
     FaceEdgeValuesMaxw(td, f, fromPartner,
                             rStage[p], rStage[p],
                             feMatStage[p]);
    break;
     }

    for (ef=0; ef<3; ef++)
     for (kef=0; kef<noTEperEdge; kef++)
  {
    RV(adr[i],rInit) = feVecInit[noTEperEdge*ef+kef];
    if (varUt)
     RV(adr[i],rInitUt) = feVecInitUt[noTEperEdge*ef+kef];
    for (p=0;p<noOfStages;p++)
     RV(adr[i],rStage[p]) = feMatStage[p][noTEperEdge*ef+kef];
    i++;
  }

    if (noTB>0)
```

```
     {
   FaceBubbleValuesMaxw(td, f, fromInit, rInit, rInit,
                           fbVecInit);
   if (varUt)
        FaceBubbleValuesMaxw(td, f, fromInitUt, rInitUt,
                            rInitUt, fbVecInitUt);

switch (timeState->stepTypePar)
 {
   case START_STEP:
     for (p=0;p<noOfStages;p++)
      for (j=0; j<noTB; j++)
   fbMatStage[p][j] = 0.0;
     break;
   case LATER_STEP:
     for (p=0;p<noOfStages;p++)
      FaceBubbleValuesMaxw(td, f, fromPartner, rStage[p],
                               rStage[p], fbMatStage[p]);
     break;
 }

for (fi=0; fi<noTB; fi++)
 {
   RV(adr[i],rInit) = fbVecInit[fi];
   if (varUt)
     RV(adr[i],rInitUt) = fbVecInitUt[fi];
   for (p=0;p<noOfStages;p++)
     RV(adr[i],rStage[p]) = fbMatStage[p][fi];
   i++;
 }
   }
 }

if (noTd>0)
  {
    InteriorValuesMaxw(td, fromInit, rInit, rInit,
                       intVecInit);
    if (varUt)
      InteriorValuesMaxw(td, fromInitUt, rInitUt,
                         rInitUt, intVecInitUt);

    switch (timeState->stepTypePar)
 {
   case START_STEP:
     for (p=0;p<noOfStages;p++)
      for (j=0; j<noTd; j++)
       intMatStage[p][j] = 0.0;
     break;
   case LATER_STEP:
```

```
      for (p=0;p<noOfStages;p++)
       InteriorValuesMaxw(td, fromPartner, rStage[p],
                                rStage[p], intMatStage[p]);
      break;
  }

     for (f=0; f<noTd; f++)
  {
    RV(adr[i],rInit) = intVecInit[f];
    if (varUt)
     RV(adr[i],rInitUt) = intVecInitUt[f];
    for (p=0;p<noOfStages;p++)
     RV(adr[i],rStage[p]) = intMatStage[p][f];
    i++;
  }
    }

 return true;
}
```

So we compute successively the dofs. Firstly the edge dofs, then using the already computed edge dofs, the face dofs and finally the interior dofs.
The functions *SetStartValuesOnNodesMaxw(TD *td), RefTetrahedronTimeMaxw(TD *td)* are similarly modified.

2.5 Numerical integration

To evaluate integrals over edges, faces, and tetrahedra we need quadrature formulas. Special care has to be taken when solution data have to be interpolated from a fine to a coarse mesh as in space-adaptive approaches. Then naive Gauss quadrature is not sufficient. Let us consider the following example: we want to integrate over an edge e (coarse mesh), which is the union of two similar edges e_1 and e_2 (fine mesh), see Figure 2.4. Suppose the integrant is linear over each subedge, that is, piecewise linear over e. Then using e.g. a two-point Gauss quadrature for an integration over e, we loose accuracy of the integration. A remedy here is to choose integration points x_1 and x_2 in the midpoints of e_1 and e_2. In this way one obtains the midpoint rule for e. Clearly, the situation becomes more complicated for triangles and tetrahedra, which may be refined in different ways and thus for each refinement type (which is not a-priori known, it has to be checked for each element) we would need different quadrature formula.

Remarks
1. Another idea would be to divide, if possible, the integral over an ele-

ment (edge, triangle or tetrahedron) into integrals over sub-elements to apply Gauss quadrature formula in each of them. This approach is not yet implemented in KARDOS.

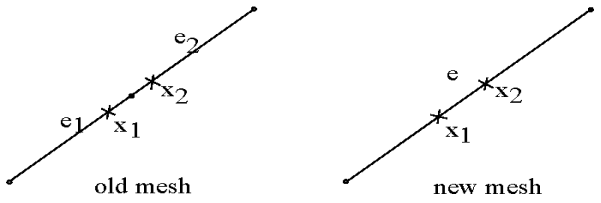

Figure 2.4: Example of an edge which is coarsened

We have implemented the following routines for lowest order edge elements.

- In *assemble3.h* new fields are introduced using the structure *integData*

```
struct integData
{
   ...
   real **edgeIPX, **edgeIPY, **edgeIPZ, **edgeIW;
   real ..., ***edgeVals;
   int  ..., noOfEdgeIP;
};
typedef struct integData integData;
```

- In *shape.c*, quadratures over edges are introduced.
- Inn *shape.c*, function *CompShapeVals(...)* is accordingly modified by adding:

```
for (i=0; i<noSF; i++)
    for(k=0; k<6; k++)
       for (j=0; j<(iData->noOfEdgeIP); j++)
         (*ShapeF)( (iData->edgeIPX)[k][j],
                    (iData->edgeIPY)[k][j],
                    (iData->edgeIPZ)[k][j],
                    i,
                    &((iData->edgeVals)[i][k][j]),
                    &dum1,&dum2,&dum3,&dum4,&dum5,
                    &dum6,&dum7,&dum8,&dum9
                   );
```

- In *nedshape.c*, function *static int UpdateIDataMaxw(integData *iData, int iFormula, int iLFormula, int iEFormula, int SF, void (*ShapeF)(...), int symP)* is introduced, similar to the function *UpdateIData(...)* from *shape.c*.

- In *timeintegmaxw.c*, in function *EdgeValuesMaxw(...)* special quadratures are chosen when the partner is refined:

```
if ( ((tet == fromPartner)||
      (tet == fromInit)||
      (tet == fromInitUt)) &&
(tPartner->firstSon != nil) )
  {
    edgeIPX    = refEdgeIntegData->edgeIPX;
    edgeIPY    = refEdgeIntegData->edgeIPY;
    edgeIPZ    = refEdgeIntegData->edgeIPZ;
    edgeIW     = refEdgeIntegData->edgeIW;
    edgeVals   = refEdgeIntegData->edgeVals;
    noOfEdgeIP = refEdgeIntegData->noOfEdgeIP;
  }
```

Chapter 3

Eigenvalues Computation

Here we consider the eigenvalue problem described in [12], pg. 50. The solution of Maxwell's cavity problem involves the computation of the nonzero resonant frequencies $\omega^2 \backslash \{0\}$, and the associated electric field $\boldsymbol{E}$ defined by

$$\begin{aligned} \mathbf{curl}\mu^{-1}\mathbf{curl}\boldsymbol{E} - \omega^2\epsilon\boldsymbol{E} &= 0, \quad in \quad \Omega & (3.1)\\ \boldsymbol{n} \times \boldsymbol{E} &= 0, \quad on \quad \partial\Omega. & (3.2) \end{aligned}$$

The variational problem, without including a Lagrange multiplier, is: find $\boldsymbol{E} \in \boldsymbol{H}_0(\text{curl})$ and $\omega^2 \in R\backslash\{0\}$ such that

$$(\mu^{-1}\mathbf{curl}\boldsymbol{E}\ , \mathbf{curl}\boldsymbol{W})_\Omega - \omega^2(\epsilon\boldsymbol{E}\ , \boldsymbol{W})_\Omega = 0, \quad \boldsymbol{W} \in \boldsymbol{H}_0(\text{curl}). \tag{3.3}$$

We first computed in KARDOS the curl-curl matrix C and the mass matrix M. Using MATLAB's function *eigs* for sparse matrices we got the generalized eigenvalues from

$$C \cdot V = M \cdot V \cdot D,$$

where D is a diagonal matrix containing the eigenvalues, and V is a full matrix which columns are the corresponding eigenvectors.
We consider this problem in the 2D and 3D case, as in [12].

3.1 The two-dimensional case

3.1.1 The L-shape domain

Firstly, the computation of the eigenvalues for the $(-1,-1) \times (1,1)$ L-shape domain with perfect electrical conductor (PEC) boundaries is considered. At a PEC boundary the tangential component of the electric field and the normal component of the magnetic field are zero.

Two sequences of five geometric meshes are constructed, the first sequence is generated by uniform refinements, while the second one is constructed with refinement towards the reentrant corner. On each mesh, the polynomial order was uniformly increased from $p = 0$ to $p = 3$.
For each discretization, the first three eigenvalues were computed and the errors in the computed eigenvalues are determined using the available benchmark values

$$\omega_1^2 = 1.47562182408, \quad \omega_2^2 = 3.53403136678, \quad \omega_3^2 = 9.86960440109.$$

The obtained results are shown in Figure 3.1 and 3.2.

3.1.2 The square domain

Now we consider the case when the cavity Ω is the unit cube $[0, 1] \times [0, 1]$. In [2] also a square cavity is considered. The eigenvalues are known to be

$$\lambda^2 = \pi^2(m^2 + n^2), \quad \text{with } m, n = 0, 1, ... \text{ s.t. } m^2 + n^2 \neq 0.$$

We consider again a sequence of 5 five graded meshes and on each mesh we compute the first 15 eigenvalues for $p = 0, 1, 2, 3$. In Fig. 3.3, the first 15 eigenvalues computed with $p = 0, 1, 2, 3$ are shown, for the 4th mesh in the sequence. Relative errors for the third eigenvalues are shown in Fig. 3.4.

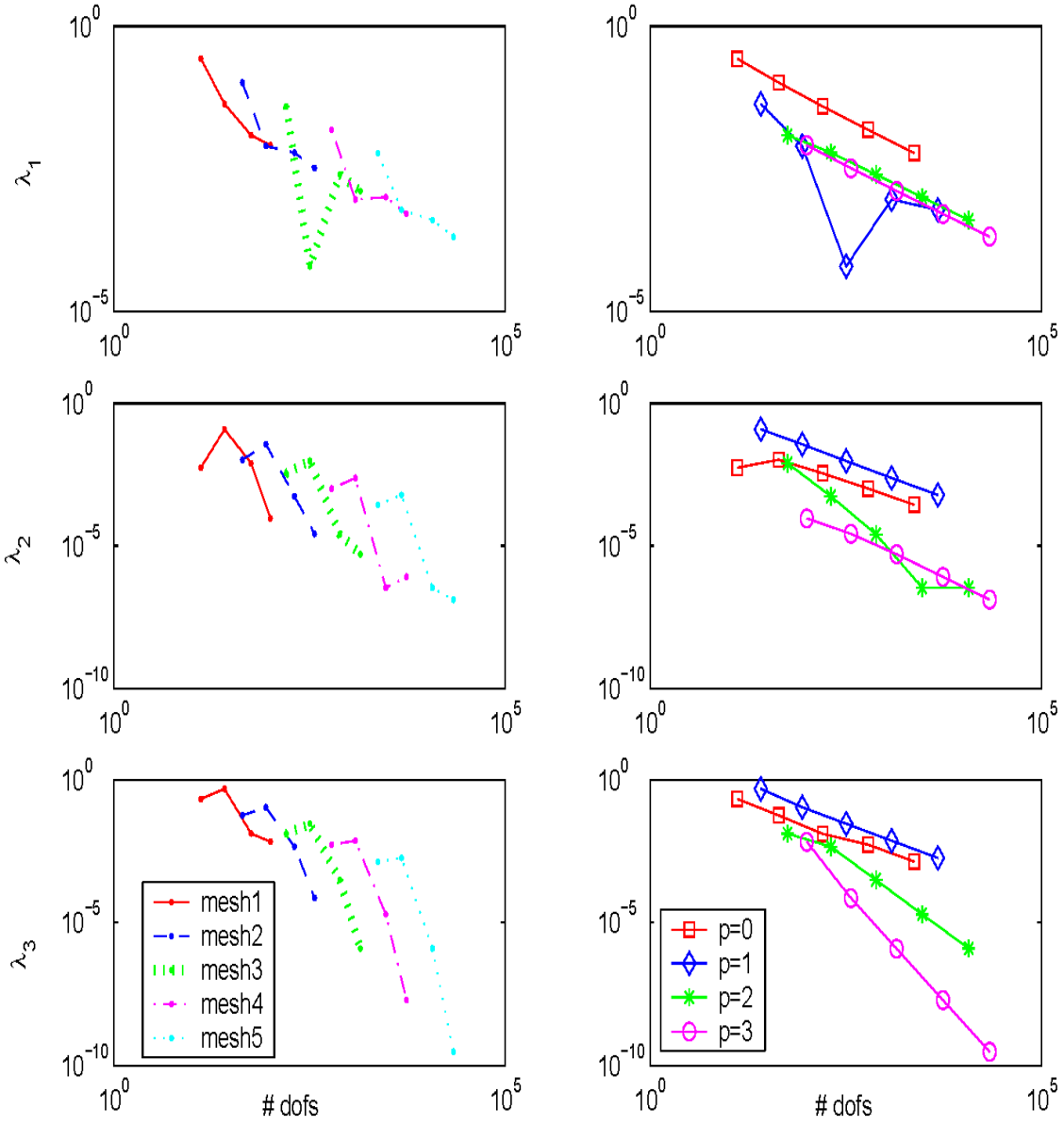

Figure 3.1: Relative errors for the first three eigenvalues, left with increasing p-refinement on the first sequence of five graded meshes generated by uniform refinements, and right with uniform h-refinement, for $p = 0, 1, 2, 3$

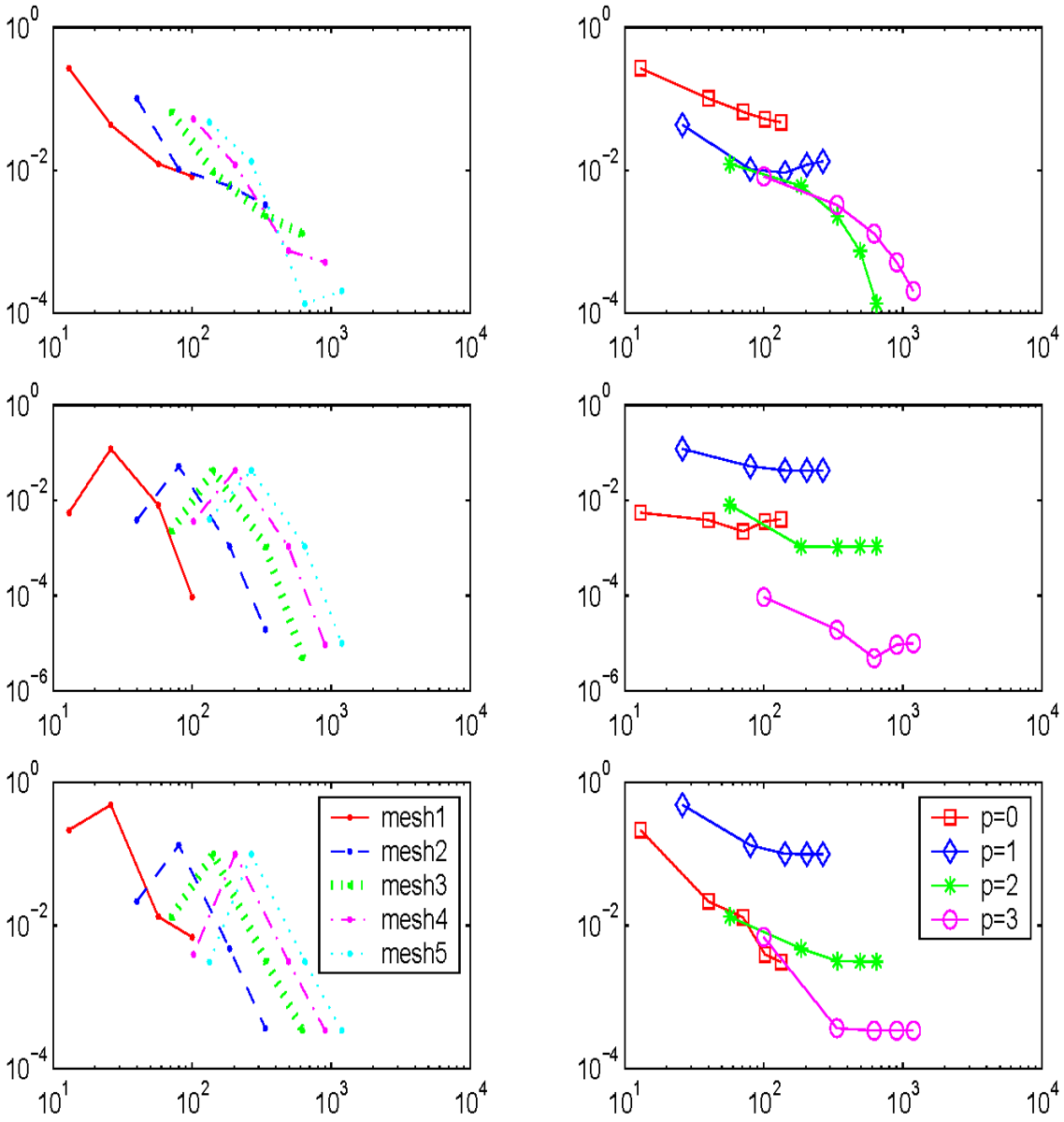

Figure 3.2: Results displayed as in Fig. 3.1, but for the sequence of five graded meshes constructed with refinement towards the reentrant corner

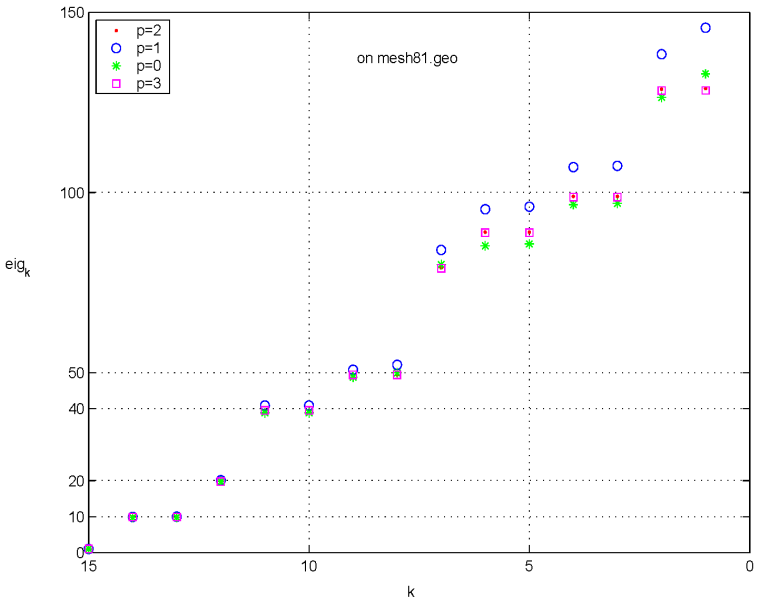

Figure 3.3: The first 15 eigenvalues for the square cavity, with $p = 0, 1, 2, 3$, on a fixed mesh

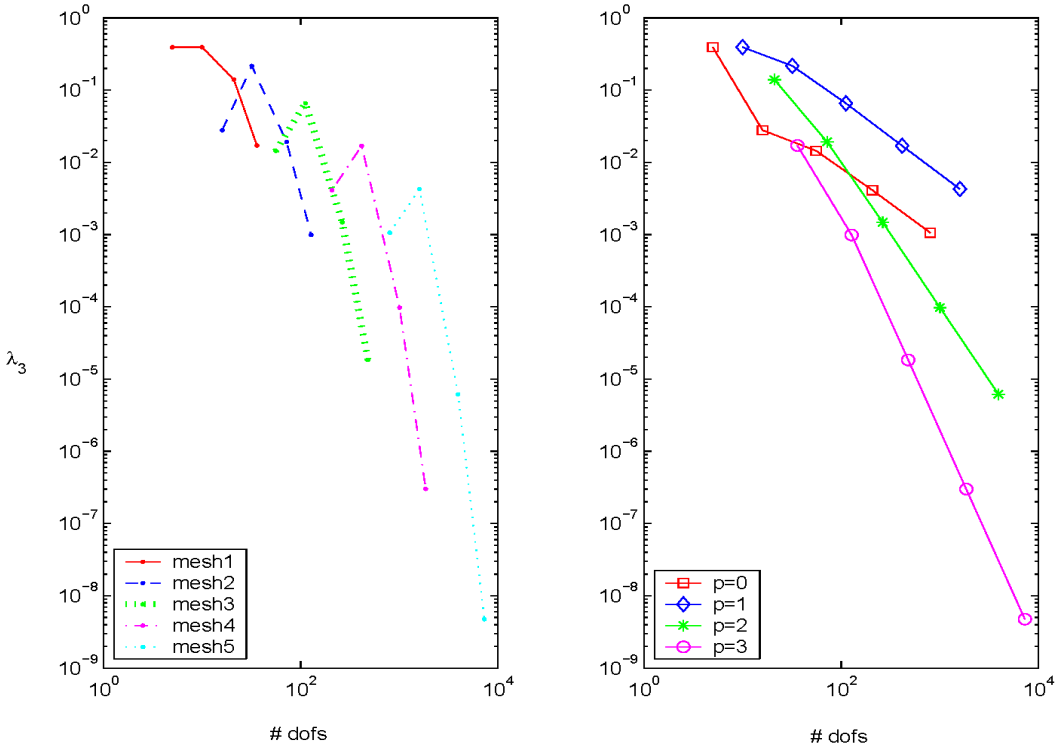

Figure 3.4: Convergence studies for the third eigenvalue of the 2D square cavity

3.2 The three-dimensional case

We consider again the case when the cavity Ω is the unit cube $[0,1]^3$, but now in 3D. The eigenvalues are known to be

$$\lambda^2 = \pi^2(l^2+m^2+n^2), \quad \text{with } l,m,n = 0,1,... \text{ s.t. } lm+ln+mn > 0.$$

When $lmn > 0$, there are two identical eigenvalues associated with linearly independent eigenfunctions. This problem is solved also in [4].
For this case, we considered three meshes: *cubeD125.geo, cubeD729.geo* and *cubeD4913.geo*.
In Fig. 3.5 the first eigenvalues are shown, obtained with $p = 0$ on these three meshes. For $p = 1, 2$, see Fig. 3.6 and Fig. 3.7, respectively.

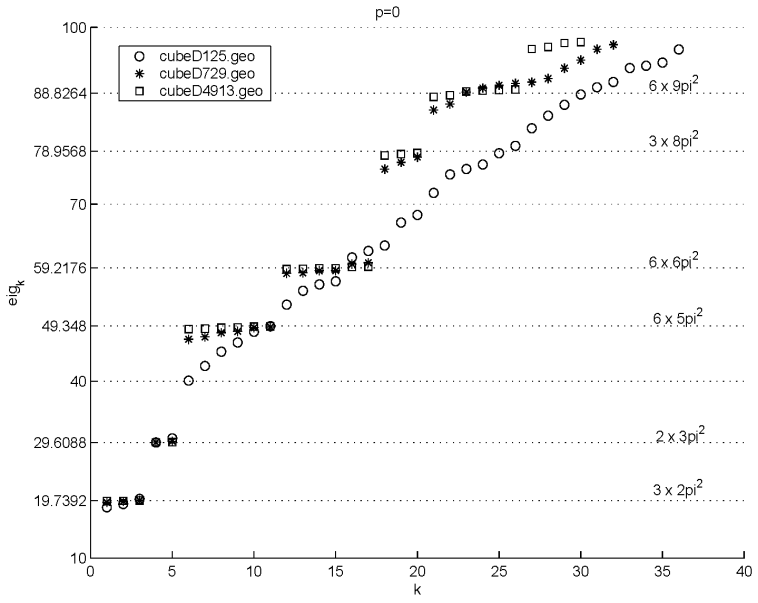

Figure 3.5: The first eigenvalues for the cube cavity, with $p = 0$, on three different meshes. The values above the solid lines indicate the exact values and the multiplicity of the eigenvalues

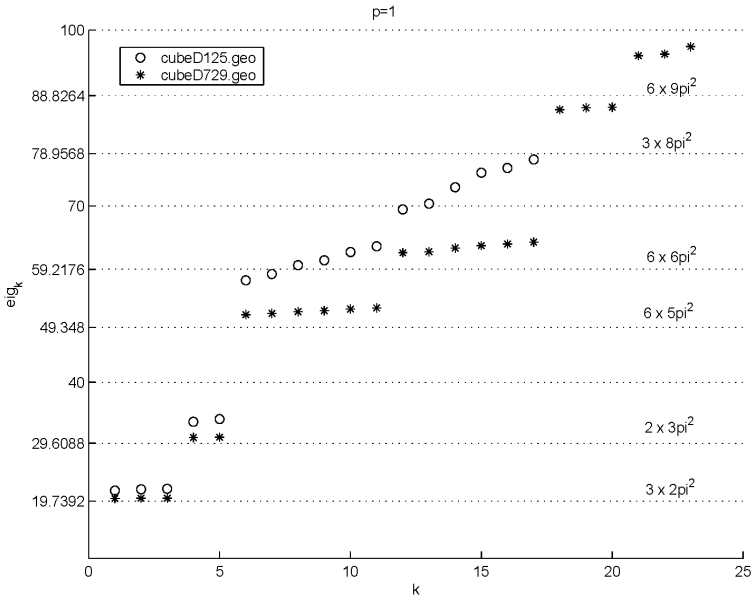

Figure 3.6: The first eigenvalues for the cube cavity, with $p = 1$, on two different meshes

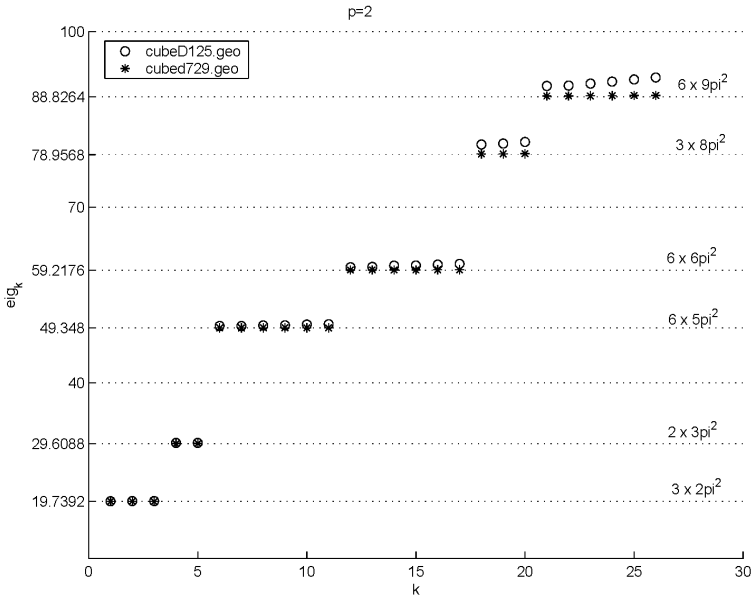

Figure 3.7: The first eigenvalues for the cube cavity, with $p = 2$, on two different meshes

Chapter 4

Magnetoquasistatic problems

4.1 A magnetostatic boundary value problem

We consider the magnetostatic boundary value problem (as in [20])

$$\mathbf{curl}\mu^{-1}\mathbf{curl}\boldsymbol{A} = \boldsymbol{j}, \tag{4.1}$$

where $\boldsymbol{j}$ is the given current density, and $\boldsymbol{A}$ is the vector potential for the magnetic flux $\boldsymbol{B} = \mathbf{curl}\boldsymbol{A}$. For gauging, as in [20], we add a small (6 orders of magnitudes smaller than μ^{-1}) zero-order term to the operator, i.e., we solve the problem (1.10) with

$$\sigma = 10^{-6}\mu^{-1}.$$

The magnetic field induced by a cylindrical coil is computed. We choose

$$\mu = 4.0 * \pi * 10^{-7}, \quad \sigma = 0.8.$$

Up to now the current density $\boldsymbol{j}$ is explicitly given by

$$\boldsymbol{j}(x,y,z) = \begin{cases} (-\frac{y}{\sqrt{(x^2+y^2}}, \frac{x}{\sqrt{(x^2+y^2}}, 0) * 1000, & \text{in coil,} \\ 0, & \text{elsewhere.} \end{cases} \tag{4.2}$$

Remarks:
1. $\boldsymbol{j}$ must be given or computed s.t. $\operatorname{div}\boldsymbol{j} = 0$.

In *maxwelltime.ksk* we set:

```
seliterate pcg
seldirect ma28
setpardirect dropfac 1.0e-8   licnfac 20   lirnfac 20
```

```
selprecond ilu
setpariter itertol 1.0e-4 itermaxsteps 1000 krylovdim 20
seldisc wfem
seltimeinteg ros1
solveinform iterinfo 1 estiinfo 0 refinfo 0 solveinfo 1 timeinfo 1
setpartime maxsteps 1 timetol 1.0e+30 spacetol 1.0e+0
setpartime maxtimestep 1.0e+30 direct 0 maxreductions 1
setscaling atol 1.0e-2 rtol 1.0
timestepping timestep 1.0e+30 tend 1.0e+30
```

Results are shown im Fig. 4.1-4.3.

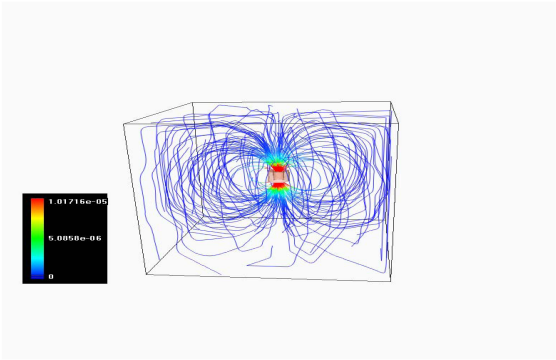

Figure 4.1: The magnetic field induced by a cylindrical coil, $p = 0$.

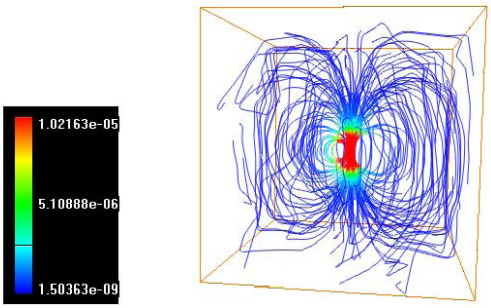

Figure 4.2: The magnetic field induced by a cylindrical coil, $p = 1$.

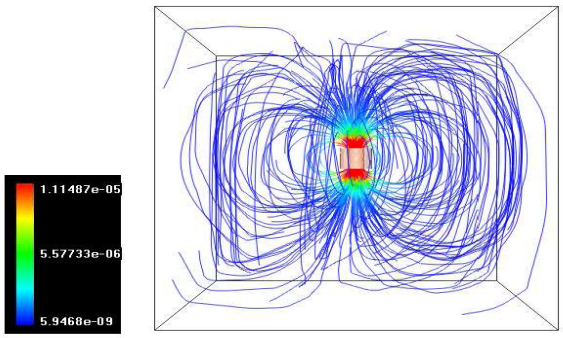

Figure 4.3: The magnetic field induced by a cylindrical coil, $p = 2$.

4.2 TEAM 7 problem: asymmetrical conductor with hole

We consider the TEAM 7 benchmark problem [7]. The problem consists of a rectangular aluminium plate, $\sigma = 3.526\,10^7\,S/m$, with eccentric rectangular cutout placed under an eccentrically positioned coil, Fig. 4.4. A sinusoidal current of $2742\,A$ and $50\,Hz$ flows through the coil and induces a current in the plate. An analytic model is applied to ensure $\operatorname{div} J_s = 0$. As computational domain we use a cube with $1m$ edge length. Homogeneous boundary conditions, $\boldsymbol{n} \times \boldsymbol{A} = 0$, are taken.

To check our implementation, we have compared our results with experimental data. Fig. 4.5 reveals good agreement.

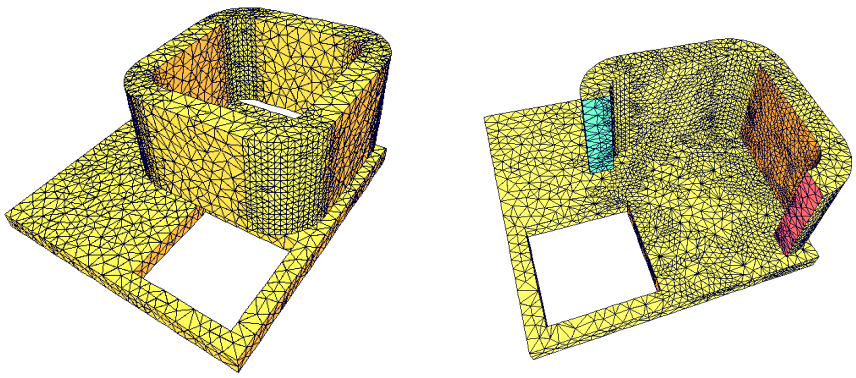

Figure 4.4: Coarse (left) and selected fine (right) tetrahedral meshes. The surrounding mesh for the air region is not shown.

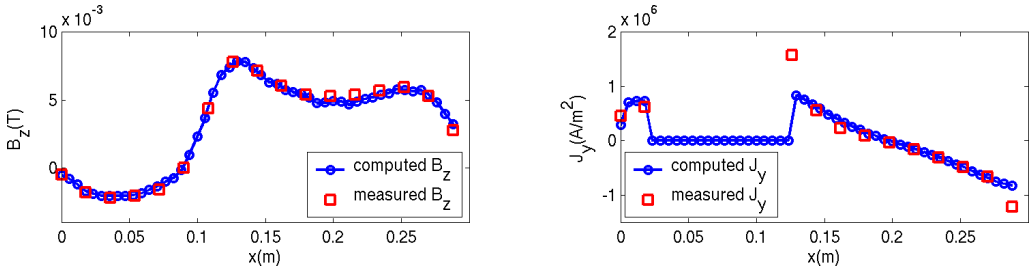

Figure 4.5: Comparison of computed and measured real part of $\boldsymbol{B}_z$ (left) and of $\boldsymbol{J}_{E,y}$ (right), see [7] for reference values.

Appendix A

Notations:

$$\boldsymbol{\nabla} u = \quad \mathbf{grad}\, u = \quad (\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial u}{\partial z}) \tag{A.1}$$

$$\nabla \cdot \boldsymbol{u} = \quad \operatorname{div} \boldsymbol{u} = \quad \frac{\partial u_1}{\partial x} + \frac{\partial u_2}{\partial y} + \frac{\partial u_3}{\partial z} \tag{A.2}$$

$$\nabla \times \boldsymbol{u} = \quad \mathbf{curl}\, \boldsymbol{u} = \quad \begin{pmatrix} \frac{\partial u_3}{\partial y} - \frac{\partial u_2}{\partial z} \\ \frac{\partial u_1}{\partial z} - \frac{\partial u_3}{\partial x} \\ \frac{\partial u_2}{\partial x} - \frac{\partial u_1}{\partial y} \end{pmatrix} \tag{A.3}$$

If $\boldsymbol{u} = (u_1, u_2)^T$, then

$$\operatorname{curl}\, \boldsymbol{u} = \frac{\partial u_2}{\partial x} - \frac{\partial u_1}{\partial y}. \tag{A.4}$$

Vector identities:

$$\mathbf{curl}\, \mathbf{grad}\, u = \mathbf{0} \tag{A.5}$$

$$\operatorname{div} \mathbf{curl}\, \boldsymbol{u} = 0. \tag{A.6}$$

Integral theorems:

$$\int_V \nabla \cdot \boldsymbol{u}\, dV = \int_S \boldsymbol{u} \cdot \boldsymbol{n}\, da \tag{A.7}$$

$$\int_V \nabla \times \boldsymbol{u}\, dV = \int_S \boldsymbol{n} \times \boldsymbol{u}\, da \tag{A.8}$$

$$\int_S (\nabla \times \boldsymbol{u}) \cdot \boldsymbol{n}\, da = \int_C \boldsymbol{u} \cdot d\boldsymbol{x} \tag{A.9}$$

$$\int_S (\boldsymbol{n} \times \nabla u)\, da = \int_C u\, d\boldsymbol{x} \tag{A.10}$$

Bibliography

[1] M. AINSWORTH, J. COYLE, *Hierarchic Finite Element Basis on Unstructured Tetrahedral Meshes*, Int. J. Num. Meth. Eng., 58(14), pp. 2103-2130, 2003.

[2] M. AINSWORTH, J. COYLE, *Hierarchic hp-edge element families for Maxwell's Equations on hybrid quadrilateral/triangular meshes*, Comput. Methods Appl.Mech. Eng., 190, pp. 6709-6733, 2001.

[3] M. CLEMENS, J. LANG, D. TELEAGA, G. WIMMER, *Adaptivity in Space and Time for Magnetoquasistatics*, submitted to JCM Special Issue on Adaptive and Multilevel in Electromagnetics, 2007.

[4] J. COYLE, P.D. LEDGER, *Evidence of Exponential Convergence in the Computation of Maxwell Eigenvalues*, Computer Methods in Applied Mechanics and Engineering, 194, pp. 587-604, 2005.

[5] M. COSTABEL, M. DAUGE, *Computation of resonance frequences for Maxwell equations in non smooth domains*, Lecture notes in computational science and engineering Vol. 31, Springer, 2003.

[6] B. ERDMANN, J. LANG, R. ROITZSCH, *KARDOS User's Guide*, Technical Report ZR-02-42, ZIB, 2002, see also http://www.zib.de/Numerik/software/kardos.

[7] K. FUJIWARA, T. NAKATA, *Results for benchmark problem 7 (asymmetrical conductor with a hole)*, COMPEL, vol. 9, nr. 3, 137-154, 1990.

[8] R.D. GRAGLIA, D.R. WILTON, A.F. PETERSON, *Higher Order Interpolatory Vector Bases for Computational Electromagnetics*, IEEE Trans. Antennas and Propagat., 45, 1997.

[9] J. JIN, *The FEM in Electromagnetics*, Wiley & Sons, New York, 1993.

[10] J. LANG, D. TELEAGA, *Towards a Fully Space-Time Adaptive FEM for Magnetoquasistatics*, IEEE Trans. Magn., 44, pp. 1238-1241, 2008.

[11] P.D. LEDGER, *An hp-Adaptive Finite Element Procedure for Electromagnetic Scattering Problems*, PhD Thesis, Dept. Civil Engineering, Univ. of Wales, Swansea, 2002.

[12] P.D. LEDGER, K.MORGAN, *The Application of the hp-Finite Element Method to Electromagnetic problems*, Archives of Comput. Meth. in Engrg., 12, pp. 235-302, 2005.

[13] P. MONK, *Finite Element Methods for Maxwell's Equations*, Oxford Univ. Press, 2003.

[14] J.C. NEDELEC, *Mixed finite elements in* R^3, Numerische Mathematik, 35, pp. 315-341, 1980.

[15] J.C. NEDELEC, *A new family of mixed finite elements in* R^3, Numerische Mathematik, 50, pp. 57-81, 1986.

[16] W. RACHOWICZ, L. DEMKOWICZ, *An hp-adaptive finite element method for electromagnetics. Part1: Data structure and constrained approximation*, Comput. Methods Appl. Mech. Eng. 187, pp. 307-335, 2000.

[17] W. RACHOWICZ, L. DEMKOWICZ, *A three-dimensional hp-adaptive finite element package for electromagnetics*, Int. J. Num. Meth. Engin., 53, pp. 147-180, 2002.

[18] Z. REN, *Influence of the RHS on the convergence behaviour of the Curl-Curl equation*, IEEE Trans. Magn. 32, no.3, 1996.

[19] J. SCHOEBERL, *Numerical Methods for Maxwell Equations*, Skript SS2005.

[20] J. SCHOEBERL, S. ZAGLMAYR, *High Order Nedelec Elements with local complete sequence properties*, International Journal for Computation and Mathematics in Electrical and Electronic Engineering (COMPEL), 24, pp. 374-384, 2005.