Computing 69, 239–263 (2002)
Digital Object Identifier (DOI) 10.1007/s00607-002-1459-8

Computing
Printed in Austria

Matlab Implementation of the Finite Element Method in Elasticity

J. Alberty, Kiel, **C. Carstensen**, Vienna, **S. A. Funken**, Kiel and **R. Klose**, Kiel

Received June 18, 2001; revised February 25, 2002
Published online: December 5, 2002

Abstract

A short Matlab implementation for P_1 and Q_1 finite elements (FE) is provided for the numerical solution of 2d and 3d problems in linear elasticity with mixed boundary conditions. Any adaptation from the simple model examples provided to more complex problems can easily be performed with the given documentation. Numerical examples with postprocessing and error estimation via an averaged stress field illustrate the new Matlab tool and its flexibility.

AMS Subject Classification: 65N30, 65R20, 73C50.

Keywords: Finite element method, elasticity, Matlab.

1. Introduction

Unlike complex black-box commercial computer codes, this paper provides a simple and short open-box Matlab implementation of P_1 and Q_1 finite elements (FE) for the numerical solutions of linear elasticity problems. Instead of covering all kinds of possible problems in one code, the proposed tool aims to be plain, easy to understand, and to modify. Therefore, only simple model examples are included to be adapted to whatever is needed in the spirit of [1]. We present an error estimator which illustrates the accuracy of the numerical calculation.

A different approach is realized by the company Comsol with the finite element package Femlab [7]. It is a tool for the solution of partial differential equations of many physical phenomena, including structural mechanics. A special feature is the comfortable mesh generator and the possibility to combine different physical problems. The code is written for the most part in Matlab and can be modified, except for the mesh generator and the solver for nonlinear problems. The basis functions in the finite element method can be chosen to be linear, quadratic, cubic or of fourth order. The solver can use adaptivity as well.

The plan of our paper is as follows. The model problem, the Navier-Lamé equations, with general boundary conditions is described in Sect. 2; the weak formulation and discretisation in Sect. 3. The heart of this contribution is the data representation of the triangulation, the Dirichlet and Neumann boundary in

Sect. 4 together with the discrete space. The main steps are the assembly procedure of the stiffness matrix in Sect. 5, the right-hand side in Sect. 6, and the incorporation of the Dirichlet boundary conditions in Sect. 7. A post-processing routine to preview the numerical solution is given in Sect. 8; in Sect. 9 an implementation of an a posteriori error estimator is presented [3], [4], [5]. Applications illustrate the usage of the new tools in Sects. 10, 11, 12, and 13 and serve as examples for writing the user-specified Matlab routines `f.m`, `g.m`, and `u_d.m`. The main program is given partly in these sections and in its total in the appendices.

The programs are written for Matlab 6 but adaptation for earlier versions is possible. For a finite element calculation it is necessary to run the main program with (user-specified files) `coordinates.dat`, `elements.dat`, `dirichlet.dat`, and `neumann.dat` if necessary, as well as the sub-routines `f.m`, `g.m`, and `u_d.m`. The graphical representation is performed with the function `show.m` and the error is estimated in `aposteriori.m`. All files with the code and the data for all examples can be downloaded from *www.math.tuwien.ac.at/~carsten/*.

2. Model Problem

The proposed Matlab program employs the finite element method to calculate a numerical solution $\mathbf{U}$ which approximates the solution $\mathbf{u}$ to the following d-dimensional Navier-Lamé problem (P). Let $\Omega \subset \mathbb{R}^d$ be a bounded Lipschitz domain with polygonal boundary Γ. On some closed subset Γ_D of the boundary with positive length, we assume Dirichlet conditions while we have Neumann boundary conditions on the (possible empty) part Γ_N. The d components of the displacement $\mathbf{u}$ need not satisfy either Dirichlet- or Neumann conditions, i.e., Γ_D and Γ_N may overlap. Given $\mathbf{f} \in L^2(\Omega)$, $\mathbf{M} \in L^\infty(\Gamma_D)^{d\times d}$, $\mathbf{w} \in H^1(\Omega)$, and $\mathbf{g} \in L^2(\Gamma_N)$ as well as the positive parameters λ and μ, seek $\mathbf{u} \in H^1(\Omega)$ with

$$(\lambda+\mu)(\nabla \operatorname{div} \mathbf{u})^T + \mu\Delta\mathbf{u} = -\mathbf{f} \qquad \text{in } \Omega, \tag{1}$$

$$(\lambda \operatorname{tr}(\epsilon(\mathbf{u}))I + 2\mu\epsilon(\mathbf{u}))\cdot \mathbf{n} = \mathbf{g} \qquad \text{on } \Gamma_N, \tag{2}$$

$$\mathbf{M}\cdot\mathbf{u} = \mathbf{w} \qquad \text{on } \Gamma_D, \tag{3}$$

where $\epsilon(\mathbf{u}) = (\nabla\mathbf{u} + (\nabla\mathbf{u})^T)/2$. The matrix $\mathbf{M}$ contains in row number $j = 1,\ldots,d$ the entries $m_{j1},\ldots,m_{jd}$ (explained in Sect. 7) to model gliding geometric boundary conditions. According to Korn's inequality and the Lax-Milgram Lemma [2, 6], there exists a weak solution to (1)–(3) for reasonable boundary conditions in (3), i.e., there exists $\mathbf{u} \in H^1(\Omega)$ that satisfies (3) and, for all $\mathbf{v} \in H^1_D(\Omega) := \{\mathbf{v} \in H^1(\Omega) : \mathbf{M}\mathbf{v} = 0 \text{ on } \Gamma_D\}$,

$$\int_\Omega \epsilon(\mathbf{v}) : \mathbb{C}\epsilon(\mathbf{u})\, dx = \int_\Omega \mathbf{f}\cdot\mathbf{v}\, dx + \int_{\Gamma_N} \mathbf{g}\cdot\mathbf{v}\, ds. \tag{4}$$

($\mathbb{C}$ is the fourth-order isotropic material tensor corresponding to (1).) The weak form (4) will be relevant in the sequel and so boundary conditions are treated correctly while (2) can be wrong if only some components of $\mathbf{u}$ on $\Gamma_N \cap \Gamma_D$ are fixed and so (2) holds only for some projections of both sides.

3. Finite Element Discretisation

For the implementation, problem (4) is discretised using the standard Galerkin method. The space $H^1(\Omega)$ is replaced by the finite dimensional subspace $\mathscr{S}$. If $\mathbf{w}_h$ is the projection of $\mathbf{w}$ onto $\mathscr{S}$, the discretised problem (P_S) reads: Seek $\mathbf{u}_h \in \mathscr{S}$ such that $\mathbf{M}\mathbf{u}_h = \mathbf{w}_h$ on Γ_D and, for all $\mathbf{v}_h \in \mathscr{S}$ with $\mathbf{M}\mathbf{v}_h = 0$ on Γ_D,

$$\int_\Omega \epsilon(\mathbf{v}_h) : \mathbb{C}\epsilon(\mathbf{u}_h)\, dx = \int_\Omega \mathbf{f} \cdot \mathbf{v}_h\, dx + \int_{\Gamma_N} \mathbf{g} \cdot \mathbf{v}_h\, ds. \tag{5}$$

Let $\mathscr{T}$ denote a triangulation of Ω specified in Sect. 4 and let $\mathscr{N}$ be the set of all nodes in $\mathscr{T}$; set $(\boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_{dN}) = (\varphi_1\mathbf{e}_1, \varphi_1\mathbf{e}_2, \ldots, \varphi_1\mathbf{e}_d, \ldots, \varphi_N\mathbf{e}_1, \varphi_N\mathbf{e}_2, \ldots, \varphi_N\mathbf{e}_d)$ be the nodal basis of the finite dimensional space $\mathscr{S}$, where N is the number of nodes of the mesh, d the dimension of the displacement vector, and φ_z is the scalar hat function of node z in the triangulation $\mathscr{T}$, i.e., $\varphi_z(z) = 1$ and $\varphi_z(y) = 0$ for all $y \in \mathscr{N}$ with $y \neq z$. Up to geometric boundary conditions (involved in Sect. 7), (5) reads

$$\int_\Omega \epsilon(\boldsymbol{\eta}_k) : \mathbb{C}\epsilon(\mathbf{u}_h) dx = \int_\Omega \mathbf{f} \cdot \boldsymbol{\eta}_k\, dx + \int_{\Gamma_N} \mathbf{g} \cdot \boldsymbol{\eta}_k\, ds \qquad (k = 1, \ldots, dN). \tag{6}$$

For the discrete displacement vector we have $\mathbf{u}_h = \sum_{\ell=1}^{dN} U_\ell \boldsymbol{\eta}_\ell$ and thus (6) yields the linear system of equations

$$\sum_{\ell=1}^{dN} \left(\int_\Omega \epsilon(\boldsymbol{\eta}_k) : \mathbb{C}\epsilon(\boldsymbol{\eta}_\ell) dx \right) U_\ell = \int_\Omega \mathbf{f} \cdot \boldsymbol{\eta}_k\, dx + \int_{\Gamma_N} \mathbf{g} \cdot \boldsymbol{\eta}_k\, ds \qquad (k = 1, \ldots, dN). \tag{7}$$

The coefficient matrix (global stiffness matrix) $A = (A_{k\ell}) \in \mathbb{R}^{dN \times dN}$ and the right-hand side $\mathbf{b} = (b_k) \in \mathbb{R}^{dN}$ are defined as

$$A_{k\ell} = \int_\Omega \epsilon(\boldsymbol{\eta}_k) : \mathbb{C}\epsilon(\boldsymbol{\eta}_\ell) dx \quad \text{and} \quad b_k = \int_\Omega \mathbf{f} \cdot \boldsymbol{\eta}_k\, dx + \int_{\Gamma_N} \mathbf{g} \cdot \boldsymbol{\eta}_k\, ds. \tag{8}$$

The coefficient matrix is sparse, symmetric and positive semidefinite. The condition $\mathbf{M}\mathbf{u}_h = \mathbf{w}_h$ on Γ_D and $\mathbf{M}\mathbf{v}_h = 0$ on Γ_D will be incorporated in Sect. 7 via Lagrange multipliers.

4. Data Representation of the Triangulation

Supposing the domain Ω has a polygonal boundary Γ, we can cover $\bar{\Omega}$ by a regular triangulation $\mathscr{T}$, i.e., $\bar{\Omega} = \bigcup_{T\in\mathscr{T}} T$, where the elements of $\mathscr{T}$ are triangles and/or quadrilaterals for $d = 2$ and tetrahedrons for $d = 3$. Regular triangulation in the sense of Ciarlet [6] means that the nodes $\mathscr{N}$ of the mesh lie on the vertices of the elements, the elements of the triangulation do not overlap, no node lies on an edge of an element, and each edge $E \subset \Gamma$ of an element $T \in \mathscr{T}$ belongs completely to $\bar{\Gamma}_N$, completely to $\bar{\Gamma}_D$, or completely to both.

Matlab supports reading data from files given in ASCII format by `*.dat` files. Fig. 1 shows the mesh of a two dimensional example and the corresponding file `coordinates.dat`, which contains the coordinates of each node. The two real numbers per row are the x- and y-coordinates of each node.

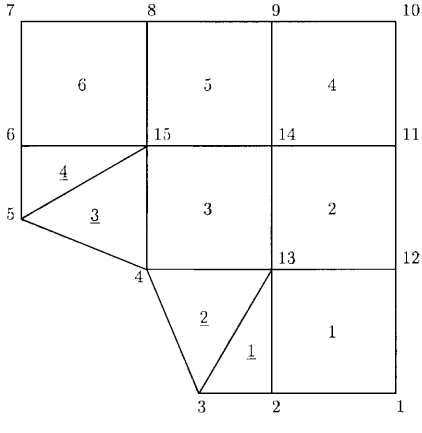
The files `elements3.dat` and `elements4.dat` contain for each triangular and quadrilateral element the node numbers of the vertices, numbered anti-clockwise. See, for instance, the triangular element marked $\underline{1}$ in Fig. 1 which is given by the node numbers $(13, 3, 2)$ and *not* $(13, 2, 3)$.

The space $\mathscr{S}$ is chosen globally continuous and linear on each triangle ($d = 2$) or tetrahedron ($d = 3$) and bilinear on each quadrilateral element.

With this space $\mathscr{S}$ and its corresponding nodal basis, the integrals in (8) can be calculated as a sum over all elements and a sum over all edges on Γ_N, i.e., for $j, k = 1, \ldots, dN$,

$$A_{jk} = \sum_{T\in\mathscr{T}} \int_T \epsilon(\boldsymbol{\eta}_j) : \mathbb{C}\epsilon(\boldsymbol{\eta}_k) dx, \tag{9}$$

$$b_j = \sum_{T\in\mathscr{T}} \int_T \mathbf{f}\cdot\boldsymbol{\eta}_j dx + \sum_{E\subset\Gamma_N} \int_E \mathbf{g}\cdot\boldsymbol{\eta}_j ds. \tag{10}$$



coordinates.dat

3.0	0.0
2.0	0.0
1.41	0.0
1.0	1.0
0.0	1.41
0.0	2.0
0.0	3.0
1.0	3.0
2.0	3.0
3.0	3.0
3.0	2.0
3.0	1.0
2.0	1.0
2.0	2.0
1.0	2.0

Fig. 1. Plot of triangulation and corresponding data file `coordinates.dat`

elements3.dat

13	3	2
4	3	13
15	5	4
5	15	6

elements4.dat

2	1	12	13
13	12	11	14
4	13	14	15
14	11	10	9
15	14	9	8
6	15	8	7

Fig. 2. Data files `elements3.dat` and `elements4.dat`

The files `neumann.dat` and `dirichlet.dat` contain the two node numbers which bound the corresponding edge on the boundary. In this way, the sum over $E \subset \Gamma_N$ results in a loop over all entries in `neumann.dat`. The right-hand side **f** and **g** is evaluated in Sect. 6 and the Dirichlet conditions from `u_d.m` are discussed in Sect. 7.

5. Assembling the Stiffness Matrix

The local stiffness matrix is determined by the coordinates of the vertices of the corresponding element and is calculated in the function `stima3` for triangular elements, `stima4` for quadrilateral elements, and `stima` for tetrahedral elements.

dirichlet.dat

3	2
2	1
7	6
6	5

neumann.dat

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

Fig. 3. Data files `dirichlet.dat` and `neumann.dat`

Let k_1 through k_K be the numbers of the nodes of an element T. For a d-dimensional problem there are then dK basis functions with support on T, namely

$$\boldsymbol{\eta}_{\pi(T,1)} = \varphi_{k_1}\mathbf{e}_1, \ldots, \ \boldsymbol{\eta}_{\pi(T,d)} = \varphi_{k_1}\mathbf{e}_d,$$

$$\ldots$$

$$\boldsymbol{\eta}_{\pi(T,dK-(d-1))} = \varphi_{k_K}\mathbf{e}_1, \ldots, \ \boldsymbol{\eta}_{\pi(T,dK)} = \varphi_{k_K}\mathbf{e}_d,$$

where $\mathbf{e}_j$ is the j-th unit vector. The function φ_{k_j} is the local scalar hat function of node k_j. The function π maps the indices $1, \ldots, dK$ of the local numeration with respect to T to the global numeration. The entries of the local stiffness matrix of an element T read

$$STIMA(T)_{k\ell} = \int_T \epsilon(\boldsymbol{\eta}_{\pi(T,k)}) : \mathbb{C}\epsilon(\boldsymbol{\eta}_{\pi(T,\ell)})dx \qquad (k, \ell = 1, \ldots, dK) \tag{11}$$

and are assembled to the global stiffness matrix A. This means, the entry A_{pq} is the sum of $STIMA(T)_{k\ell}$ over all elements T with $p = \pi(T, k)$ and $q = \pi(T, \ell)$, which is performed by the following Matlab commands for the two-dimensional case for triangles and quadrilaterals.

```
for j = 1:size(elements3,1)
  I = 2*elements3(j,[1,1,2,2,3,3]) -[1,0,1,0,1,0];
  A(I,I) = A(I,I) +stima3(coordinates(elements3(j,:),:),lambda,mu);
end
for j = 1:size(elements4,1)
  I = 2*elements4(j,[1,1,2,2,3,3,4,4]) -[1,0,1,0,1,0,1,0];
  A(I,I) = A(I,I) +stima4(coordinates(elements4(j,:),:),lambda,mu);
end
```

The local stiffness matrix for each element T requires an individual discussion for triangular, tetrahedral, and quadrilateral elements.

5.1. Triangular Elements

We consider triangular elements with the vertices (x_j, y_j, z_j) for $j = 1, 2, 3$. The Voigt representation $\gamma : H^1(\Omega) \to L^2(\Omega)$ of the linear Green strain tensor reads

$$\gamma(\mathbf{u}) = \begin{pmatrix} \partial u_1/\partial x \\ \partial u_2/\partial y \\ \partial u_2/\partial x + \partial u_1/\partial y \end{pmatrix} = \begin{pmatrix} \epsilon_{11}(\mathbf{u}) \\ \epsilon_{22}(\mathbf{u}) \\ 2\epsilon_{12}(\mathbf{u}) \end{pmatrix}. \tag{12}$$

For $\sigma = \mathbb{C}\epsilon(\mathbf{u})$ we have the relation

$$\begin{pmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{pmatrix} = \begin{pmatrix} \lambda+2\mu & \lambda & 0 \\ \lambda & \lambda+2\mu & 0 \\ 0 & 0 & \mu \end{pmatrix} \begin{pmatrix} \epsilon_{11}(\mathbf{u}) \\ \epsilon_{22}(\mathbf{u}) \\ 2\epsilon_{12}(\mathbf{u}) \end{pmatrix} =: C\gamma(\mathbf{u}). \tag{13}$$

and thus obtain

$$\epsilon(\mathbf{v}) : \mathbb{C}\epsilon(\mathbf{u}) = \sigma_{11}\epsilon_{11} + \sigma_{22}\epsilon_{22} + \underbrace{\sigma_{12}\epsilon_{12} + \sigma_{21}\epsilon_{21}}_{=2\sigma_{12}\epsilon_{12}} = \gamma^T(\mathbf{v})C\gamma(\mathbf{u}). \tag{14}$$

From (11) and relation (14) and the fact that $\boldsymbol{\eta}_{\pi(T,k)}$ etc. are affine on T it follows that

$$STIMA(T)_{k\ell} = \int_T \gamma(\boldsymbol{\eta}_{\pi(T,k)})^T C\gamma(\boldsymbol{\eta}_{\pi(T,\ell)})dx = |T|\gamma(\boldsymbol{\eta}_{\pi(T,k)})^T C\gamma(\boldsymbol{\eta}_{\pi(T,\ell)}). \tag{15}$$

Laborious but elementary calculations verify

$$\gamma\left(\sum_{j=1}^{6} u_j \boldsymbol{\eta}_{\pi(T,j)}\right)|_T = \begin{pmatrix} \varphi_{k_1,x} & 0 & \varphi_{k_2,x} & 0 & \varphi_{k_3,x} & 0 \\ 0 & \varphi_{k_1,y} & 0 & \varphi_{k_2,y} & 0 & \varphi_{k_2,y} \\ \varphi_{k_1,y} & \varphi_{k_1,x} & \varphi_{k_2,y} & \varphi_{k_2,x} & \varphi_{k_3,y} & \varphi_{k_2,x} \end{pmatrix} \begin{pmatrix} u_1 \\ \vdots \\ u_6 \end{pmatrix} =: R \begin{pmatrix} u_1 \\ \vdots \\ u_6 \end{pmatrix} \tag{16}$$

and hence expression (15) can be written simultaneously for all indices as

$$STIMA(T) = |T|R^T CR. \tag{17}$$

The following function is a Matlab implementation of (17):

```
function stima3=stima3(vertices,lambda,mu)
PhiGrad = [1,1,1;vertices']\[zeros(1,2);eye(2)];
R = zeros(3,6);
R([1,3],[1,3,5]) = PhiGrad';
R([3,2],[2,4,6]) = PhiGrad';
C = mu*[2,0,0;0,2,0;0,0,1] +lambda*[1,1,0;1,1,0;0,0,0];
stima3 = det([1,1,1;vertices'])/2*R'*C*R;
```

This function makes use of the fact that the gradients of the scalar basis functions can be calculated as

$$\begin{pmatrix} \nabla\varphi_{k_1} \\ \nabla\varphi_{k_2} \\ \nabla\varphi_{k_3} \end{pmatrix} = \frac{1}{2|T|} \begin{pmatrix} y_2-y_3 & x_3-x_2 \\ y_3-y_1 & x_1-x_3 \\ y_1-y_2 & x_2-x_1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix}^{-1} \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{18}$$

5.2. Tetrahedral Elements

The Voigt representation $\gamma : H^1(\Omega)^3 \to L^2(\Omega)^6$ of the linear Green strain tensor reads

$$\gamma(\mathbf{u}) = \begin{pmatrix} \partial u_1/\partial x \\ \partial u_2/\partial y \\ \partial u_3/\partial z \\ \partial u_1/y + \partial u_2/x \\ \partial u_1/z + \partial u_3/x \\ \partial u_2/z + \partial u_3/y \end{pmatrix} = \begin{pmatrix} \epsilon_{11}(\mathbf{u}) \\ \epsilon_{22}(\mathbf{u}) \\ \epsilon_{33}(\mathbf{u}) \\ 2\epsilon_{12}(\mathbf{u}) \\ 2\epsilon_{13}(\mathbf{u}) \\ 2\epsilon_{23}(\mathbf{u}) \end{pmatrix}. \tag{19}$$

Using the relation

$$\begin{pmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{12} \\ \sigma_{13} \\ \sigma_{23} \end{pmatrix} = \begin{pmatrix} \lambda+2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda+2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda+2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{pmatrix} \begin{pmatrix} \epsilon_{11}(\mathbf{u}) \\ \epsilon_{22}(\mathbf{u}) \\ \epsilon_{33}(\mathbf{u}) \\ 2\epsilon_{12}(\mathbf{u}) \\ 2\epsilon_{13}(\mathbf{u}) \\ 2\epsilon_{23}(\mathbf{u}) \end{pmatrix} =: C\gamma(\mathbf{u}), \tag{20}$$

we obtain $\epsilon(\mathbf{v}) : \mathbb{C}\epsilon(\mathbf{u}) = \gamma^T(\mathbf{v})C\gamma(\mathbf{u})$. Since again $\boldsymbol{\eta}_{\pi(T,k)}$ etc. are affine on T, (15) also holds for tetrahedral elements, but this time with C as defined in 12. Finally, we have

$$\gamma\left(\sum_{j=1}^{12} u_j \boldsymbol{\eta}_{\pi(T,j)}\right)|_T = \begin{pmatrix} \varphi_{k_1,x} & 0 & 0 & \varphi_{k_2,x} & 0 & 0 & \varphi_{k_3,x} & 0 & 0 & \varphi_{k_4,x} & 0 & 0 \\ 0 & \varphi_{k_1,y} & 0 & 0 & \varphi_{k_2,y} & 0 & 0 & \varphi_{k_3,y} & 0 & 0 & \varphi_{k_4,y} & 0 \\ 0 & 0 & \varphi_{k_1,z} & 0 & 0 & \varphi_{k_2,z} & 0 & 0 & \varphi_{k_3,z} & 0 & 0 & \varphi_{k_4,z} \\ \varphi_{k_1,y} & \varphi_{k_1,x} & 0 & \varphi_{k_2,y} & \varphi_{k_2,x} & 0 & \varphi_{k_3,y} & \varphi_{k_3,x} & 0 & \varphi_{k_4,y} & \varphi_{k_4,x} & 0 \\ \varphi_{k_1,z} & 0 & \varphi_{k_1,x} & \varphi_{k_2,z} & 0 & \varphi_{k_2,x} & \varphi_{k_3,z} & 0 & \varphi_{k_3,x} & \varphi_{k_4,z} & 0 & \varphi_{k_4,x} \\ 0 & \varphi_{k_1,z} & \varphi_{k_1,y} & 0 & \varphi_{k_2,z} & \varphi_{k_2,y} & 0 & \varphi_{k_3,z} & \varphi_{k_3,y} & 0 & \varphi_{k_4,z} & \varphi_{k_4,y} \end{pmatrix} \times \begin{pmatrix} u_1 \\ \vdots \\ u_{12} \end{pmatrix} \tag{21}$$

and so we obtain (17), where R is the coefficient matrix on the right-hand side of (21).

The Matlab realisation of (17) for tetrahedral elements utilizes

$$\begin{pmatrix} \nabla\varphi_{k_1} \\ \nabla\varphi_{k_2} \\ \nabla\varphi_{k_3} \\ \nabla\varphi_{k_4} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{pmatrix}^{-1} \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{22}$$

and reads as follows.

```
function stima = stima(vertices,lambda,mu)
PhiGrad = [1,1,1,1;vertices']\[zeros(1,3);eye(3)];
R = zeros(6,12);
R([1,4,5],1:3:10) = PhiGrad';
R([4,2,6],2:3:11) = PhiGrad';
R([5,6,3],3:3:12) = PhiGrad';
C([1:3],[1:3]) = lambda*ones(3,3)+2*mu*eye(3);
C([4:6],[4:6]) = mu*eye(3);
stima = det([1,1,1,1;vertices'])/6*R'*C*R;
```

5.3. Quadrilateral Elements

Finally, we consider quadrilateral elements with vertices (x_1, y_1) through (x_4, y_4). Since T is a parallelogram, there exists an affine mapping

$$\Phi(\xi,\eta) = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_2 - x_1 & x_4 - x_1 \\ y_2 - y_1 & y_4 - y_1 \end{pmatrix} \begin{pmatrix} \xi \\ \eta \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \tag{23}$$

which maps $[0,1]^2$ onto T. The shape functions on $T_{ref} = [0,1]^2$ are defined as

$$\begin{aligned} \tilde{\boldsymbol{\eta}}_1(\xi,\eta) &= \begin{pmatrix} \tilde{\varphi}_1(\xi,\eta) \\ 0 \end{pmatrix}, \quad \tilde{\boldsymbol{\eta}}_2(\xi,\eta) = \begin{pmatrix} 0 \\ \tilde{\varphi}_1(\xi,\eta) \end{pmatrix}, \\ &\vdots \\ \tilde{\boldsymbol{\eta}}_7(\xi,\eta) &= \begin{pmatrix} \tilde{\varphi}_4(\xi,\eta) \\ 0 \end{pmatrix}, \quad \tilde{\boldsymbol{\eta}}_8(\xi,\eta) = \begin{pmatrix} 0 \\ \tilde{\varphi}_4(\xi,\eta) \end{pmatrix} \end{aligned} \tag{24}$$

with

$$\begin{aligned} \tilde{\varphi}_1(\xi,\eta) &= (1-\xi)(1-\eta), & \tilde{\varphi}_2(\xi,\eta) &= \xi(1-\eta), \\ \tilde{\varphi}_3(\xi,\eta) &= \xi\eta, & \tilde{\varphi}_4(\xi,\eta) &= (1-\xi)\eta. \end{aligned} \tag{25}$$

The entries of the local stiffness matrix for the element T are obtained by transforming the integral in (11) onto the reference element T_{ref}. We use the following abbreviation for the transpose of the gradient of the transformation.

$$F = \begin{pmatrix} \frac{\partial \Phi_1^{-1}(x,y)}{\partial x} & \frac{\partial \Phi_1^{-1}(x,y)}{\partial y} \\ \frac{\partial \Phi_2^{-1}(x,y)}{\partial x} & \frac{\partial \Phi_2^{-1}(x,y)}{\partial y} \end{pmatrix}. \tag{26}$$

We notice that for the expression $\epsilon(\boldsymbol{\eta}_{\pi(T,k)}) : \mathbb{C}\epsilon(\boldsymbol{\eta}_{\pi(t,\ell)})$ there are four different possibilities depending on the zero-components of the basis functions. In the first case we have to transform an expression of the form $\epsilon\begin{pmatrix}\varphi_{\pi(T,i)}(x,y)\\0\end{pmatrix} : \mathbb{C}\epsilon\begin{pmatrix}\varphi_{\pi(T,j)}(x,y)\\0\end{pmatrix}$ to one depending on the variables ξ, η, which results in

$$(\nabla\tilde{\varphi}_i)\, F\begin{pmatrix}\lambda+2\mu & 0\\ 0 & \mu\end{pmatrix}F^T\, \nabla\tilde{\varphi}_j{}^T. \tag{27}$$

In the second case we have to transform $\epsilon\begin{pmatrix}0\\\varphi_{\pi(T,i)}(x,y)\end{pmatrix} : \mathbb{C}\epsilon\begin{pmatrix}0\\\varphi_{\pi(T,j)}(x,y)\end{pmatrix}$ and obtain

$$(\nabla\tilde{\varphi}_i)\, F\begin{pmatrix}\mu & 0\\ 0 & \lambda+2\mu\end{pmatrix}F^T\, \nabla\tilde{\varphi}_j{}^T. \tag{28}$$

In the third case we transform the expression $\epsilon\begin{pmatrix}0\\\varphi_{\pi(T,i)}(x,y)\end{pmatrix} : \mathbb{C}\epsilon\begin{pmatrix}\varphi_{\pi(T,j)}(x,y)\\0\end{pmatrix}$ and obtain

$$(\nabla\tilde{\varphi}_i)\, F\begin{pmatrix}0 & \mu\\ \lambda & 0\end{pmatrix}F^T\, \nabla\tilde{\varphi}_j{}^T. \tag{29}$$

In the fourth case we have to transform $\epsilon\begin{pmatrix}\varphi_{\pi(T,i)}(x,y)\\0\end{pmatrix} : \mathbb{C}\epsilon\begin{pmatrix}0\\\varphi_{\pi(T,j)}(x,y)\end{pmatrix}$ and obtain

$$(\nabla\tilde{\varphi}_i)\, F\begin{pmatrix}0 & \lambda\\ \mu & 0\end{pmatrix}F^T\, \nabla\tilde{\varphi}_j{}^T. \tag{30}$$

Hence transforming the integration in (11) leads to four cases of the form

$$\int_{T_{ref}} E : \big((\nabla\tilde{\varphi}_i)^T\nabla\tilde{\varphi}_j\big)dx, \tag{31}$$

where E differs for each case and is the product of the three inner matrices in each of (27)–(30) and i and j depend on the corresponding shape functions $\tilde{\boldsymbol{\eta}}_k$ and $\tilde{\boldsymbol{\eta}}_\ell$ as in (24). Note that E is constant on T_{ref}. Hence, if we define the four matrices

$$R_{11} = \left(\int_0^1\int_0^1 \frac{\partial\tilde{\varphi}_i(\xi,\eta)}{\partial\xi}\frac{\partial\tilde{\varphi}_j(\xi,\eta)}{\partial\xi}d\xi d\eta\right)_{i,j=1,\ldots,4} = \frac{1}{6}\begin{pmatrix}2 & -2 & -1 & 1\\ -2 & 2 & 1 & -1\\ -1 & 1 & 2 & -2\\ 1 & -1 & -2 & 2\end{pmatrix}, \tag{32}$$

$$R_{22} = \left(\int_0^1\int_0^1 \frac{\partial\tilde{\varphi}_i(\xi,\eta)}{\partial\eta}\frac{\partial\tilde{\varphi}_j(\xi,\eta)}{\partial\eta}d\xi d\eta\right)_{i,j=1,\ldots,4} = \frac{1}{6}\begin{pmatrix}2 & 1 & -1 & -2\\ 1 & 2 & -2 & -1\\ -1 & -2 & 2 & 1\\ -2 & -1 & 1 & 2\end{pmatrix}, \tag{33}$$

$$R_{12} = R_{21}^T = \left(\int_0^1 \int_0^1 \frac{\partial \tilde{\varphi}_i(\xi,\eta)}{\partial \xi} \frac{\partial \tilde{\varphi}_j(\xi,\eta)}{\partial \eta} d\xi d\eta \right)_{i,j=1,\ldots,4} = \frac{1}{4} \begin{pmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 \end{pmatrix}, \tag{34}$$

the following Matlab routine calculates the local stiffness matrix for quadrilateral elements:

```
function stima4 = stima4(vertices,lambda,mu)
R_11 = [2 -2 -1  1; -2  2  1 -1; -1  1 2 -2;  1 -1 -2  2]/6;
R_12 = [1  1 -1 -1; -1 -1  1  1; -1 -1 1  1;  1  1 -1 -1]/4;
R_22 = [2  1 -1 -2;  1  2 -2 -1; -1 -2 2  1; -2 -1  1  2]/6;
F = inv([vertices(2,:)-vertices(1,:); vertices(4,:)-vertices(1,:)]);
L = [lambda+2*mu,lambda,mu];
stima4 = zeros(8,8);
E = F'*[L(1),0;0,L(3)]*F;
stima4(1:2:8,1:2:8) = E(1,1)*R_11 +E(1,2)*R_12 +E(2,1)*R_12' +E(2,2)*R_22;
E = F'*[L(3),0;0,L(1)]*F;
stima4(2:2:8,2:2:8) = E(1,1)*R_11 +E(1,2)*R_12 +E(2,1)*R_12' +E(2,2)*R_22;
E = F'*[0,L(3);L(2),0]*F;
stima4(2:2:8,1:2:8) = E(1,1)*R_11 +E(1,2)*R_12 +E(2,1)*R_12' +E(2,2)*R_22;
stima4(1:2:8,2:2:8) = stima4(2:2:8,1:2:8)';
stima4 = stima4/det(F);
```

6. Assembling the Right-hand Side

The right-hand side contains the work of the volume forces **f** and the surface forces **g**. In this section, the assembling of the right-hand will be studied for the two-dimensional (as the three-dimensional case is analogous).

The value of **f** are provided by *f.m* and evaluated in the centre of gravity (x_S, y_S) of T, to approximate the integral $\int_T \mathbf{f} \cdot \boldsymbol{\eta}_k dx$ in (10),

$$\int_T \mathbf{f} \cdot \boldsymbol{\eta}_k dx \approx \frac{1}{k_T} \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} f_j(x_S, y_S) \quad \text{with} \quad j := \text{mod}(k-1,2)+1, \tag{35}$$

where $k_T = 6$ if T is a triangle and $k_T = 4$ if T is a parallelogram. The following Matlab commands calculate the **f**-related part of (10).

```
% Volume forces
for j = 1:size(elements3,1)
  I = 2*elements3(j,[1,1,2,2,3,3]) -[1,0,1,0,1,0];
  fs = f(sum(coordinates(elements3(j,:),:))/3)';
  b(I) = b(I) +det([1,1,1;coordinates(elements3(j,:),:)'])*[fs;fs;fs]/6;
end
for j = 1:size(elements4,1)
  I = 2*elements4(j,[1,1,2,2,3,3,4,4]) -[1,0,1,0,1,0,1,0];
  fs = f(sum(coordinates(elements4(j,:),:))/4)';
  b(I) = b(I) +det([1,1,1;coordinates(elements4(j,1:3),:)'])*[fs;fs;fs;fs]/4;
end
```

The integral $\int_E \mathbf{g} \cdot \boldsymbol{\eta}_k ds$ in (10) that involves the Neumann conditions is approximated with the value of $\mathbf{g}$ in the centre (x_M, y_M) of the edge E with length $|E|$,

$$\int_E \mathbf{g} \cdot \boldsymbol{\eta}_k ds \approx \frac{1}{2}|E| g_j(x_M, y_M) \quad \text{with} \quad j := \mathrm{mod}(k-1, 2) + 1. \tag{36}$$

The user-defined function g specifies the values of $\mathbf{g}$ in its first component; its second argument is the outward normal vector. The following Matlab commands calculate the $\mathbf{g}$-related part of (10).

```
% Neumann conditions
if ~isempty(neumann)
  n = (coordinates(neumann(:,2),:) -coordinates(neumann(:,1),:))*[0,-1;1,0];
  for j = 1:size(neumann,1);
    I = 2*neumann(j,[1,1,2,2]) -[1,0,1,0];
    gm = g(sum(coordinates(neumann(j,:),:))/2, n(j,:)/norm(n(j,:)))';
    b(I) = b(I) +norm(n(j,:))*[gm;gm]/2;
  end
end
```

7. Incorporating Dirichlet Conditions

Gliding boundary conditions, such as those along symmetry axes, require a particular treatment as the displacement is fixed merely in one specified direction and possibly free in others. A general approach to this type of condition reads

$$\begin{pmatrix} \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_d \end{pmatrix} \mathbf{u} = \begin{pmatrix} w_1 \\ \vdots \\ w_d \end{pmatrix} \qquad \text{on } \Gamma, \tag{37}$$

where $\mathbf{m}_j : \Gamma \to \mathbb{R}^d$ and $w_j : \Gamma \to \mathbb{R}$. With the normal vector $\mathbf{n}$ along Γ and the canonical basis $\mathbf{e}_j$, $j = 1, \ldots, d$, in $\mathbb{R}^d$, gliding conditions, classical Dirichlet conditions and the lack of Dirichlet-type conditions on certain parts of Γ can all be included by

$$\begin{pmatrix} \mathbf{n} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{pmatrix} \mathbf{u} = \mathbf{0} \quad \text{or} \quad \begin{pmatrix} \mathbf{e}_1^T \\ \vdots \\ \mathbf{e}_d^T \end{pmatrix} \mathbf{u} = \mathbf{u_D} \quad \text{or} \quad \begin{pmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \end{pmatrix} \mathbf{u} = \mathbf{0}. \tag{38}$$

The data $m(x)$ and $W(x)$ in (37) are provided by the user in a user-defined Matlab program `u_d.m` through the command line $[m(x), W(x)] := u_D(x)$; examples of which are given in Sects. 10, 11, 12, and 13. For the discrete problem, this leads to the linear system

$$B\mathbf{U} = \mathbf{w}, \tag{39}$$

where each row of $B \in \mathbb{R}^{n \times dN}$ contains the value of some $\mathbf{m}_j \in \mathbb{R}^{dN}$ at a node on the boundary Γ_D and $\mathbf{w} \in \mathbb{R}^n$ contains the corresponding values w_j at this node; N is the number of nodes. It is assumed that $\operatorname{rang} B = n$. In particular, all $\mathbf{m}_j = \mathbf{0}$ are not included in B.

Coupling this set of conditions through Lagrange parameters with (7) leads to the extended system

$$\begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{U} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ \mathbf{w} \end{pmatrix}. \tag{40}$$

The following Matlab instructions compute the matrix B, the vector $\mathbf{w}$ for the two-dimensional case, and form the extended matrix and right-hand side:

```
% Dirichlet conditions
DirichletNodes = unique(dirichlet);
[W,M] = u_d(coordinates(DirichletNodes,:));
B = sparse(size(W,1),2*size(coordinates,1));
for k = 0:1
  for j = 0:1
    B(1+j:2:size(M,1),2*DirichletNodes-1+k) = diag(M(1+j:2:size(M,1),1+k));
  end
end
mask = find(sum(abs(B)'));
A = [A, B(mask,:)'; B(mask,:), sparse(length(mask),length(mask))];
b = [b;W(mask,:)];
```

8. Computing and Displaying the Numerical Solution

The rows of (40) form a system of equations with a symmetric, indefinite coefficient matrix $\hat{A} := \begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix}$ and right-hand side $\hat{\mathbf{b}} := (\mathbf{b}, \mathbf{w})^T$. It is solved by the binary Matlab operator \ and reads

```
x = A \ b;
```

Matlab makes use of the properties of a sparse, symmetric matrix for solving the system of equations efficiently. The solution vector is $\mathbf{x} = (\mathbf{U}, \boldsymbol{\lambda})^T$, where

U occupies the first dN components. For the two dimensional case U is extracted by

```
u = x(1 : 2 * size(coordinates,1));
```

Our two-dimensional problem models the plain strain condition. It that case, the complete stress tensor $\sigma \in \mathbb{R}^{3\times 3}_{sym}$ has the form:

$$\sigma = \begin{pmatrix} \sigma_{11} & \sigma_{12} & 0 \\ \sigma_{12} & \sigma_{22} & 0 \\ 0 & 0 & \sigma_{33} \end{pmatrix},$$

with $\sigma_{33} = \frac{\lambda}{2(\mu+\lambda)}(\sigma_{11}+\sigma_{22})$. It then follows for $|\mathrm{dev}\sigma|^2$, where $\mathrm{dev}A := A - \frac{\mathrm{tr}A}{3}\mathrm{Id}_{3\times 3}$ and $|A| := \left(\sum_{i,j=1}^{n} A_{ij}^2\right)^{1/2}$, the Frobenius norm, that

$$|\mathrm{dev}\,\sigma|^2 = \left(\frac{\mu^2}{6(\mu+\lambda)^2} + \frac{1}{2}\right)(\sigma_{11}+\sigma_{22})^2 + 2(\sigma_{12}^2 - \sigma_{11}\sigma_{22}).$$

The graphical representation shows the deformed mesh with a magnification of the displacement. The main interest might be on the elastic shear energy density $|\mathrm{dev}\sigma|^2/(4\mu)$ and so we provide a tool to attach grey tones (or a colour bar) to the displayed meshes. In the two-dimensional case we use the following Matlab routine:

```
function show(elements3,elements4,coordinates,AvS,u,lambda,mu)
AvC=(mu/(24*(mu+lambda)^2)+1/(8*mu))*(AvS(:,1)+...
     AvS(:,4)).^2+1/(2*mu)*(AvS(:,2).^2-AvS(:,1).*AvS(:,4));
factor=20;
colormap(1-gray)
trisurf(elements3,factor*u(1:2:size(u,1))+coordinates(:,1), ...
    factor*u(2:2:size(u,1))+coordinates(:,2), ...
    zeros(size(coordinates,1),1), AvC, 'facecolor','interp');
hold on
trisurf(elements4,factor*u(1:2:size(u,1))+coordinates(:,1), ...
    factor*u(2:2:size(u,1))+coordinates(:,2), ...
    zeros(size(coordinates,1),1), AvC, 'facecolor','interp');
view(0,90)
hold off
colorbar('vert')
```

The function `show` uses the variable `AvS`, which stores the nodal-values of σ_h^*; where σ_h is the stress calculated by the finite element method and σ_h^* is a smoother approximation. For triangular and tetrahedral elements, σ_h is constant on each element. In the case of quadrilateral elements, we use the stress in the centre of gravity of each element; σ_h^* is defined in every node of the mesh as the mean value of the stresses on the corresponding patch. The matrix `AvS` is calculated in the function `avmatrix` below.

The Matlab routine `trisurf(elements,x,y,z,AvC,'facecolor','interp')` is used to draw triangulations for equal types of elements. Every row of the matrix

`elements` determines one polygon where the x-, y-, and z-coordinate of each corner of this polygon is given by the corresponding entry in `x`, `y`, and `z`. (In 2d-problems, we use for the z-coordinate the same value (zero) in all mesh points.) The values together with the options "`facecolor`", "`interp`" determine the grey tone of the area determined by `AvC` and linearly interpolated.

The following Matlab function calculates the stress tensors σ_h, σ_h^* and the strain tensors $\mathbb{C}^{-1}\sigma_h$, $\mathbb{C}^{-1}\sigma_h^*$ for triangular and quadrilateral elements. The output of this routine is needed for the graphical representation of the solution in the function `show` and for the a posteriori error calculation in the function `aposteriori`. The variables `AvS` and `AvE` denote the stress and strain tensors, averaged over a patch. The variables `Sigma3`, `Sigma4`, `Eps3`, and `Eps4` denote the stress and strain tensors on triangular and quadrilateral elements.

```
function [AvE,Eps3,Eps4,AvS,Sigma3,Sigma4] = ...
                    avmatrix(coordinates,elements3,elements4,u,lambda,mu);
Eps3 = zeros(size(elements3,1),4);
Sigma3 = zeros(size(elements3,1),4);
Eps4 = zeros(size(elements4,1),4);
Sigma4 = zeros(size(elements4,1),4);
AreaOmega = zeros(size(coordinates,1),1);
AvS = zeros(size(coordinates,1),4);
AvE = zeros(size(coordinates,1),4);
for j = 1:size(elements3,1)
  area3 = det([1,1,1;coordinates(elements3(j,:),:)'])/2;
  AreaOmega(elements3(j,:)) = AreaOmega(elements3(j,:)) +area3;
  PhiGrad = [1,1,1;coordinates(elements3(j,:),:)']\[zeros(1,2);eye(2)];
  U_Grad = u([1;1]*2*elements3(j,:)-[1;0]*[1,1,1])*PhiGrad;
  Eps3(j,:) = reshape((U_Grad+U_Grad')/2,1,4);
  Sigma3(j,:) = reshape(lambda*trace(U_Grad)*eye(2) +2*mu*(U_Grad+U_Grad')/2,1,4);
  AvE(elements3(j,:),:) = AvE(elements3(j,:),:) +area3*[1;1;1]*Eps3(j,:);
  AvS(elements3(j,:),:) = AvS(elements3(j,:),:) +area3*[1;1;1]*Sigma3(j,:);
end;
for j = 1:size(elements4,1)
  area4 = det([1,1,1;coordinates(elements4(j,1:3),:)']);
  AreaOmega(elements4(j,:),:) = AreaOmega(elements4(j,:)) +area4;
  Vertices = coordinates(elements4(j,:),:);
  U_Grad = inv([Vertices(2,:)-Vertices(1,:);Vertices(4,:)-...
        Vertices(1,:)])*[-1,1,1,-1;-1,-1,1,1]*...
      [u(2*elements4(j,:)-1),u(2*elements4(j,:))]/2;
  Eps4(j,:) = reshape((U_Grad+U_Grad')/2,1,4);
  Sigma4(j,:) = reshape(lambda*trace(U_Grad)*eye(2) +2*mu*(U_Grad+U_Grad')/2,1,4);
  AvE(elements4(j,:),:) = AvE(elements4(j,:),:) +area4*[1;1;1;1]*Eps4(j,:);
  AvS(elements4(j,:),:) = AvS(elements4(j,:),:) +area4*[1;1;1;1]*Sigma4(j,:);
end
AvE = AvE./(AreaOmega*[1,1,1,1]);
AvS = AvS./(AreaOmega*[1,1,1,1]);
```

9. A Posteriori Error Estimator

The averaged stress field σ_h^* of Sect. 8 allows a simple a posteriori error estimation by comparing it to the discrete (discontinuous) stress σ_h. Indeed, $\eta_T :=$

$\|\sigma_h^* - \sigma_h\|_{L^2(T)}$ may serve as a local error indicator in automatic mesh-refining adaptive algorithms [3], [4]. A modified version of this error estimator, which involves a nodal interpolation of the true applied surface load g, was recently rigourously shown to be reliable in [5]. The error estimator $\eta = (\sum_{T\in\mathscr{T}} \eta_T^2)^{1/2}$ might be interpreted in practical computations as a hint on the possible discretisation error. In the examples below, the relative quantity

$$\eta = \frac{\sqrt{\sum_{T\in\mathscr{T}} \eta_T^2}}{\sqrt{\sum_{T\in\mathscr{T}} \int_T \sigma_h^* : \mathbb{C}^{-1}\sigma_h^* dx}} \quad \text{with} \quad \eta_T^2 = \int_T (\sigma_h - \sigma_h^*) : \mathbb{C}^{-1}(\sigma_h - \sigma_h^*) dx, \quad (41)$$

serves as such an error guess; the robust error estimator from [5] is denoted by $\hat{\eta}$ and given in brackets following the value of (41).

The stress tensors σ_h, σ_h^* and the strain tensors $\mathbb{C}^{-1}\sigma_h$, $\mathbb{C}^{-1}\sigma_h^*$ for triangular and quadrilateral elements are calculated for example in the function `avmatrix` given in Sect. 8.

The following Matlab routine calculates the estimated error for triangular and/or quadrilateral elements:

```
function estimate=aposteriori(coordinates,elements3,elements4, ...
                                AvE,Eps3,Eps4,AvS,Sigma3,Sigma4,u,lambda,mu);
eta3 = zeros(size(elements3,1),1);
eta4 = zeros(size(elements4,1),1);
e3 = zeros(size(elements3,1),1);
e4 = zeros(size(elements4,1),1);
for j=1:size(elements3,1)
  Area3=det([1,1,1;coordinates(elements3(j,:),:)'])/2;
  for i=1:4
    eta3(j) = eta3(j) + Area3 * (Sigma3(j,i)*Eps3(j,i) ...
        + AvS(elements3(j,:),i)'*[2,1,1;1,2,1;1,1,2]*AvE(elements3(j,:),i)/12 ...
        - AvS(elements3(j,:),i)'*[1;1;1]*Eps3(j,i)/3 ...
        - AvE(elements3(j,:),i)'*[1;1;1]*Sigma3(j,i)/3);
    e3(j) = e3(j) + Area3 * ...
        AvS(elements3(j,:),i)'*[2,1,1;1,2,1;1,1,2]*AvE(elements3(j,:),i)/12;
  end
end
for j=1:size(elements4,1)
  Area4=det([1,1,1;coordinates(elements4(j,1:3),:)']);
  for i=1:4
    eta4(j) = eta4(j) + Area4 * (Sigma4(j,i)*Eps4(j,i) ...
        + AvS(elements4(j,:),i)'*[4,2,1,2;2,4,2,1;1,2,4,2;2,1,2,4]* ...
        AvE(elements4(j,:),i)/36 ...
        - AvS(elements4(j,:),i)'*[1;1;1;1]*Eps4(j,i)/4 ...
        - AvE(elements4(j,:),i)'*[1;1;1;1]*Sigma4(j,i)/4);
    e4(j) = e4(j) + Area4 * ...
        AvS(elements4(j,:),i)'*[4,2,1,2;2,4,2,1;1,2,4,2;2,1,2,4]* ...
        AvE(elements4(j,:),i)/36;
  end
end
estimate = sqrt(sum(eta3)+sum(eta4))/ sqrt(sum(e3)+sum(e4));
```

10. L-Shape Example

The L-shape Ω described by the polygon $(-1,-1)$, $(0,-2)$, $(2,0)$, $(0,2)$, $(-1,1)$, $(0,0)$ serves as a first test example where the exact solution u is known in polar coordinates,

$$\begin{aligned} u_r(r,\theta) &= \frac{1}{2\mu} r^{\alpha}\big[-(\alpha+1)\cos\big((\alpha+1)\theta\big) + \big(C_2-(\alpha+1)\big)C_1\cos\big((\alpha-1)\theta\big)\big], \\ u_\theta(r,\theta) &= \frac{1}{2\mu} r^{\alpha}\big[(\alpha+1)\sin\big((\alpha+1)\theta\big) + (C_2+\alpha-1)C_1\sin\big((\alpha-1)\theta\big)\big]. \end{aligned} \tag{42}$$

The critical exponent $\alpha \approx 0.544483737$ is the solution of the equation $\alpha\sin(2\omega)+\sin(2\omega\alpha)=0$ with $\omega = 3\pi/4$ and $C_1 = -(\cos\big((\alpha+1)\omega\big))/(\cos\big((\alpha-1)\omega\big))$, $C_2 = (2(\lambda+2\mu))/(\lambda+\mu)$. The function (42) solves the Lamé equation (1) with $f=0$ and the boundary condition (3) on $\Gamma = \Gamma_D$ (with w defined through u). This example is a common benchmark problem for linear elasticity because it models a re-entrant corner with a typical singularity in the stress at $(0,0)$. For the elasticity module $E = 100000$ and the shear module $v = 0.3$ of bronze the Lamé constants are $\lambda = Ev/((1+v)(1-2v))$ and $\mu = E/(2(1+v))$.

The Matlab code for the functions `u_d.m`, `u_value.m` and `f.m` is given below. It should be noted that in this particular example, the material parameters E and v determine the boundary condition and thus appear in `u_value.m`.

```
function [W,M] = u_d(x)
M = zeros(2*size(x,1),2);
W = zeros(2*size(x,1),1);
M(1:2:end,1) = 1;
M(2:2:end,2) = 1;
value = u_value(x);
W(1:2:end,1) = value(:,1);
W(2:2:end,1) = value(:,2);
```

```
function volforce = f(x);
volforce = zeros(size(x,1),2);
```

Files `u_d.m` and `f.m` for L-Shape Example

```
function value = u_value(x)
[phi,r] = cart2pol(x(:,1),x(:,2));
E = 1e6; nu = 0.3; lambda = E*nu/((1+nu)*(1-2*nu)); mu = E/(2*(1+nu));
alph = .544483737; omega = 3*pi/4;
C_1 = -cos((alph+1)*omega)/cos((alph-1)*omega);
C_2 = 2*(lambda+2*mu)/(lambda+mu);
ut = 1/(2*mu) *r.^alph .*((alph+1)*sin((alph+1)*phi) ...
                                  +(C_2+alph-1)*C_1*sin((alph-1)*phi));
ur = 1/(2*mu) *r.^alph .*(-(alph+1)*cos((alph+1)*phi) ...
                                  +(C_2-alph-1)*C_1*cos((alph-1)*phi));
value = [ur.*cos(phi) -ut.*sin(phi), ur.*sin(phi) +ut.*cos(phi)];
```

File `u_value.m` for L-Shape Example

The function show of Sect. 8 yields the deformed mesh with a displacement multiplied by a factor of 3000 based on P_1 Finite Elements and Q_1 Finite Elements for 450 degrees of freedom of Figs. 4 and 5. The grey tones visualise $|\mathrm{dev}\sigma_h^*|^2/(4\mu)$ as described in Sect. 8 from the averaged stress field σ_h^*. The singularity at $(0,0)$ causes different maximal values in the visualized stress of Figs. 4 and 5 on coarse meshes (for finer meshes the stress at that corner increases).

The averaging a posteriori error estimate η of Sect. 9 (the reliable estimator $\hat{\eta}$ of [5]) indicates a relative error $\eta = 15\%$ ($\hat{\eta} = 15\%$) for Q_1 finite elements and $\eta = 26\%$ ($\hat{\eta} = 26\%$) for P_1 finite elements; η is not a reliable estimate but a reasonable error guess (while $\hat{\eta}$ is a reliable error estimator which merely lacks an unknown universal constant).

11. Cook's Membrane Problem

A two-dimensional tapered panel $\Omega = \mathrm{conv}\{(0,0),(48,44),(48,60),(0,44)\}$ of Plexiglass (with $E = 2900$, $v = 0.4$), named Cook's membrane problem, serves as a second numerical example for a bending dominated elastic response. The panel is clamped at one end ($x = 0$) and subjected to a shearing load $g = (0,1)$ on the opposite end ($x = 48$) with vanishing volume force f. The Matlab routines for the functions u_d, f, and g are given in the table below:

```
function [W,M] = u_d(x)
M= zeros(2*size(x,1),2);
W= zeros(2*size(x,1),1);
M(1:2:end,1)= 1;
M(2:2:end,2)= 1;
```

```
function volforce = f(x);
volforce= zeros(size(x,1),2);
```

```
function sforce = g(x,n)
sforce= zeros(size(x,1),2);
sforce(find(n(:,1)==1),2)= 1;
```

Files u_d.m, f.m, and g.m for Cook's membrane problem

Figure 6 shows the deformed mesh (578 degrees of freedom) displayed by the Matlab function show with a magnifying factor 20 for the displacements. The numerical solution shows peak stresses at the corner $(0,44)$ as it is expected from the literature.

The a posteriori error estimate of Sect. 9 indicates a relative error $\eta = 16\%$ (while the reliable estimator of [5] results in $\hat{\eta} = 17\%$).

12. Membrane with a Hole

A two-dimensional benchmark problem with the elastic body with a hole $\Omega = (-3,3)^2 \setminus B(0,\sqrt{2})$ ($E = 2900$ and $v = 0.4$) is stretched at the top ($y = 3$) and at the bottom ($y = -3$) with a surface load $g = n$, where n denotes the outer normal to $\partial\Omega$; the rest of the boundary is traction free [9]. As the problem is symmetric only a quarter of Ω was discretized. Section 7 discussed the resulting boundary conditions on the initial symmetry axes described by $(-1,0)\ \mathbf{u} = 0$ on

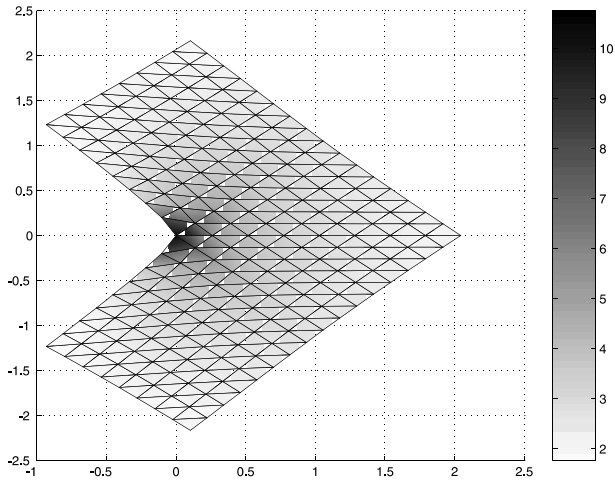


Fig. 4. Deformed mesh for P_1 FE on L-Shape

$\{0\} \times [\sqrt{2}, 3]$ and by $(0, -1)\ \mathbf{u} = 0$ on $[\sqrt{2}, 3] \times \{0\}$ which are included in the following table:

```
function [W,M] = u_d(x)
M= zeros(2*size(x,1),2);
W= zeros(2*size(x,1),1);
% symmetry condition on the x-axis
temp= find(x(:,1)>0 & x(:,2)==0);
M(2*temp-1,2)= 1;
% symmetry condition on the y-axis
temp= find(x(:,2)>0 & x(:,1)==0);
M(2*temp-1,1)= 1;
```

```
function volforce = f(x);
volforce= zeros(size(x,1),2);
```

```
function sforce = g(x,n)
sforce= zeros(size(x,1),2);
sforce(find(n(:,2)==1),2)= 1;
```

Files `u_d.m`, `f.m`, and `g.m` for membrane with hole

The Matlab program yield the deformed mesh (1122 degrees of freedom) shown in Fig. 7 with displacements magnified by a factor of 20. The example demonstrates how the combination of triangular and quadrilateral finite elements can be combined to describe regions with non-parallel boundaries without globally preferring a certain mesh orientation.

The relative error is $\eta = 8\%$ according to the a posteriori error estimate of Sect. 9 and $\hat{\eta} = 9\%$ according to the reliable estimator of [5].

13. 3D-Example : Iron Piece of Hardware

As a 3-dimensional example we solve the Navier-Lamé equation on a region that models an iron piece of hardware with a more complex geometry than the pre-

vious examples. The material is fixed at the two holes with centres $O_1 = \{17, -1.5, 21\}$ and $O_2 = \{48, -1.5, 21\}$ so that at their inner edges there is a homogeneous Dirichlet boundary. The hole with centre $O_3 = \{20, -77, 11.5\}$ is lifted in z-direction by 0.1 units described by a Dirichlet boundary condition with $u_D = \{0, 0, 0.1\}$ there; the rest of the boundary is traction free. The describing Matlab routines for the boundary conditions and the volume force are given in the following table:

```
function [W,M] = u_d(x)
M= zeros(3*size(x,1),3);
W= zeros(3*size(x,1),1);
M(1:3:end,1) = 1;
M(2:3:end,2) = 1;
M(3:3:end,3) = 1;
W(3*find(x(:,2)<-50)) = 0.1;
```

```
function volforce = f(x);
volforce= zeros(size(x,1),3);
```

```
function sforce = g(x,n)
sforce= zeros(size(x,1),3);
```

Files `u_d.m`, `f.m`, and `g.m` for the 3d iron piece of hardware

The deformed mesh for 6512 degrees of freedom is presented in Fig. 8, where the displacement is magnified by a factor of 100. Again the shades of grey visualise the square mean of the eigenvalues of the stress as calculated and displayed by functions `show` and `avmatrix` similar to those presented in Sect. 8 and 9.

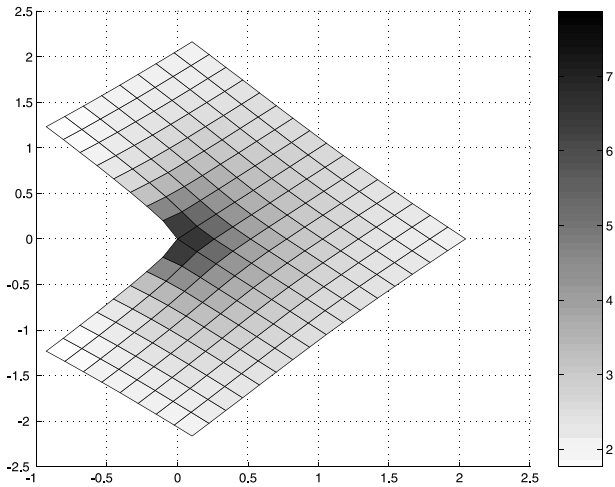


Fig. 5. Deformed mesh for Q_1 FE on L-Shape

Appendix A

Main Program for Two-dimensional Problems

What follows is the complete listing of the main program as it was used for Example 11 and Example 12. It differs in line 1 in Example 10 owing to different material constants E and v.

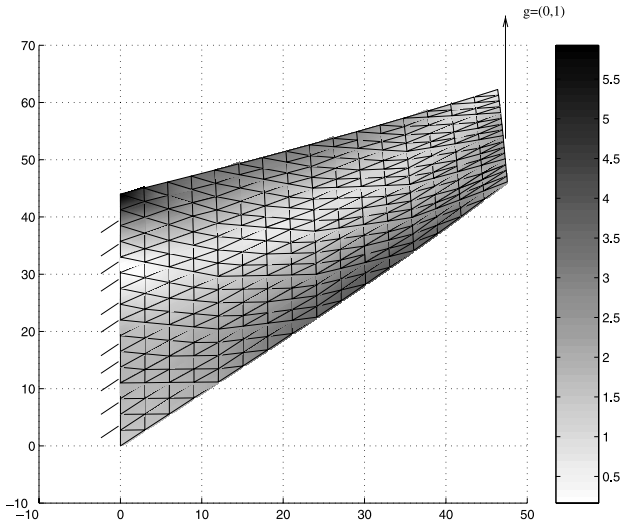


Fig. 6. Deformed mesh for Cook's membrane

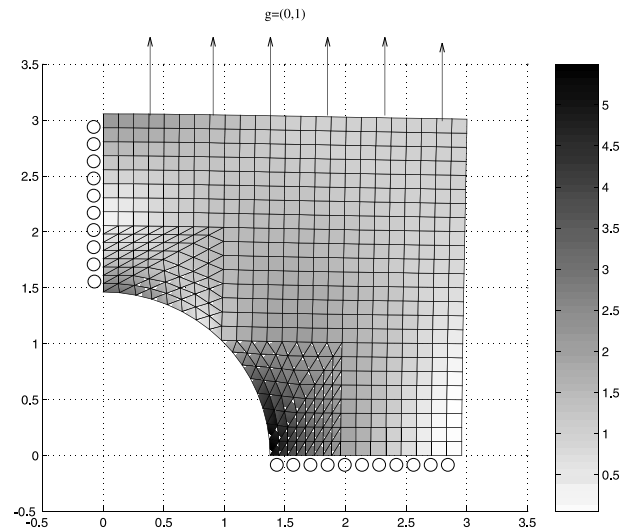


Fig. 7. Deformed mesh for membrane with hole

```
%FEM_LAME2D  two-dimensional finite element method for the Lame-Problem
% Initialisation
 1 E = 2900; nu = 0.4;
 2 mu = E/(2*(1+nu)); lambda = E*nu/((1+nu)*(1-2*nu));
 3 load coordinates.dat;
 4 eval('load elements3.dat;','elements3 = [];');
 5 eval('load elements4.dat;','elements4 = [];');
 6 eval('load neumann.dat;','neumann = [];');
 7 load dirichlet.dat;
 8 A = sparse(2*size(coordinates,1),2*size(coordinates,1));
 9 b = zeros(2*size(coordinates,1),1);
% Assembly
10 for j = 1:size(elements3,1)
11   I = 2*elements3(j,[1 1 2 2 3 3]) -[1 0 1 0 1 0];
12   A(I,I) = A(I,I) +stima3(coordinates(elements3(j,:),:),lambda,mu);
13 end
14 for j = 1:size(elements4,1)
15   I = 2*elements4(j,[1 1 2 2 3 3 4 4]) -[1 0 1 0 1 0 1 0];
16   A(I,I) = A(I,I) +stima4(coordinates(elements4(j,:),:),lambda,mu);
17 end
% Volume forces
18 for j = 1:size(elements3,1)
19   I = 2*elements3(j,[1 1 2 2 3 3]) -[1 0 1 0 1 0];
20   fs = f(sum(coordinates(elements3(j,:),:))/3)';
21   b(I) = b(I) +det([1,1,1;coordinates(elements3(j,:),:)'])*[fs;fs;fs]/6;
22 end
23 for j = 1:size(elements4,1)
24   I = 2*elements4(j,[1 1 2 2 3 3 4 4]) -[1 0 1 0 1 0 1 0];
25   fs = f(sum(coordinates(elements4(j,:),:))/4)';
26   b(I) = b(I) +det([1,1,1;coordinates(elements4(j,1:3),:)'])*[fs;fs;fs;fs]/4;
27 end
% Neumann conditions
28 if ~isempty(neumann)
29   n = (coordinates(neumann(:,2),:) -coordinates(neumann(:,1),:))*[0,-1;1,0];
30   for j = 1:size(neumann,1);
31     I = 2*neumann(j,[1,1,2,2]) -[1,0,1,0];
32     gm = g(sum(coordinates(neumann(j,:),:))/2, n(j,:)/norm(n(j,:)))';
33     b(I) = b(I) +norm(n(j,:))*[gm;gm]/2;
34   end
35 end
% Dirichlet conditions
36 DirichletNodes = unique(dirichlet);
37 [W,M] = u_d(coordinates(DirichletNodes,:));
38 B = sparse(size(W,1),2*size(coordinates,1));
39 for k = 0:1
40   for l = 0:1
41     B(1+l:2:size(M,1),2*DirichletNodes-1+k) = diag(M(1+l:2:size(M,1),1+k));
42   end
43 end
```

```
mask = find(sum(abs(B)'));
A = [A, B(mask,:)'; B(mask,:), sparse(length(mask),length(mask))];
b = [b;W(mask,:)];
% Calculating the solution
x = A \ b;
u = x(1:2*size(coordinates,1));
% Representation of the solution
[AvE,Eps3,Eps4,AvS,Sigma3,Sigma4] = ...
    avmatrix(coordinates,elements3,elements4,u,lambda,mu);
show(elements3,elements4,coordinates,AvS,u,lambda,mu);
estimate = aposteriori(coordinates,elements3,elements4,AvE,Eps3,Eps4, ...
    AvS,Sigma3,Sigma4,u,lambda,mu)
```

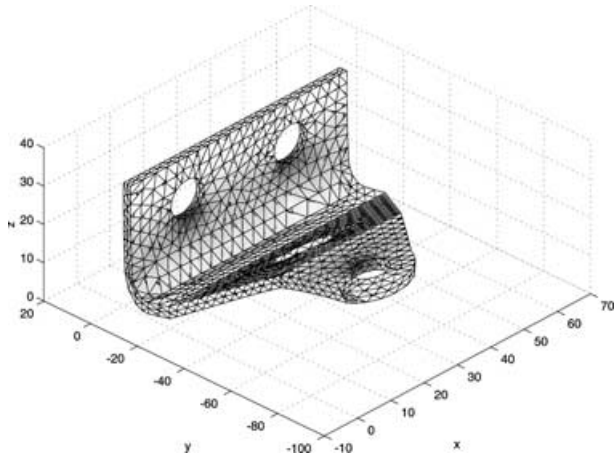


Fig. 8. Deformed mesh for the 3d iron piece of hardware

Appendix B

Main Program for Three-dimensional Problems

What follows is the complete listing of the main program as it was used for Example 13.

```
%FEM_LAME3D  three-dimensional finite element method for the Lame-Problem
% Initialisation
 1 E=100000;nu=0.3;mu=E/(2*(1+nu));lambda=E*nu/((1+nu)*(1-2*nu));
 2 load coordinates.dat;
 3 load elements.dat;
 4 eval('load neumann.dat;','neumann = [];');
 5 load dirichlet.dat;
 6 A = sparse(3*size(coordinates,1),3*size(coordinates,1));
 7 b = zeros(3*size(coordinates,1),1);
% Assembly
 8 for j = 1:size(elements,1)
 9   I = 3*elements(j,[1 1 1 2 2 2 3 3 3 4 4 4])-[2 1 0 2 1 0 2 1 0 2 1 0];
10   A(I,I) = A(I,I) +stima(coordinates(elements(j,:),:),lambda,mu);
11 end
% Volume forces
12 for j = 1:size(elements,1)
13   I = 3*elements(j,[1 1 1 2 2 2 3 3 3 4 4 4])-[2 1 0 2 1 0 2 1 0 2 1 0];
14   fs = f(sum(coordinates(elements(j,:),:))/4)';
15   b(I) = b(I)+det([1,1,1,1;coordinates(elements(j,:),:)'])*[fs;fs;fs;fs]/24;
16 end
% Neumann conditions
17 if ~isempty(neumann)
18   for j = 1:size(neumann,1)
19     n = cross( coordinates(neumann(j,2),:)-coordinates(neumann(j,1),:), ...
20         coordinates(neumann(j,3),:)-coordinates(neumann(j,1),:));
21     I = 3*neumann(j,[1 1 1 2 2 2 3 3 3])-[2 1 0 2 1 0 2 1 0];
22     gm = g(sum(coordinates(neumann(j,:),:))/3,n/norm(n))';
23     b(I) = b(I) +norm(n)*[gm;gm;gm]/6;
24   end
25 end
% Dirichlet conditions
26 dirichletnodes = unique(dirichlet);
27 [W,M] = u_d(coordinates(dirichletnodes,:));
28 B = sparse(size(W,1),3*size(coordinates,1));
29 for k = 0:2
30   for l = 0:2
31     B(1+l:3:size(M,1),3*dirichletnodes-2+k) = diag(M(1+l:3:size(M,1),1+k));
32   end
33 end
34 mask = find(sum(abs(B)'));
35 A = [A,B(mask,:)';B(mask,:),sparse(length(mask),length(mask))];
36 b = [b;W(mask,:)];
% Calculation of the solution
37 x = A\b;
38 u = x(1:3*size(coordinates,1));
% Graphical representation
39 show(elements,dirichlet,neumann,coordinates,u,lambda,mu);
```

Acknowledgments

It is our pleasure to thank Axel Hecht and Darius Zarrabi for discussions and other contributions on the propagation of this report. The fourth author (R. K.) thankfully acknowledges the partial support by the German research foundation at the Graduiertenkolleg 357 "Effiziente Algorithmen und Mehrskalenmethoden".

References

[1] Alberty, J., Carstensen, C., Funken, S. A.: Remarks around 50 lines of Matlab: finite element implementation Numerical Algorithms *20*, 117–137 (1999).
[2] Brenner, S. C., Scott, L. R.: The mathematical theory of finite element methods. Texts in Applied Mathematics *15*. New York: Springer 1994.
[3] Carstensen, C., Bartels, S.: Each averaging technique yields reliable a posteriori error control in FEM on unstructured grids, Part I: Low order conforming, nonconforming, and mixed FEM. Math. Comp. *71*, 945–969 (2002).
[4] Carstensen, C., Funken, S. A.: Averaging technique for FE-A posteriori error control in elasticity, Part I: Conforming FEM. Comput. Methods Appl. Mech. Engrg. *190*, 2483–2498 (2001).
[5] Carstensen, C., Funken, S. A.: Averaging technique for FE-A posteriori error control in elasticity. Part II: λ-independent estimates. Comput. Methods Appl. Mech. Engrg. *190*, 4663–4675 (2001).
[6] Ciarlet, P. G.: The finite element method for elliptic problems. Amsterdam: North-Holland 1978.
[7] COMSOL Group: FEMLAB: Multiphysics in Matlab. (`http://www.femlab.com`)
[8] Schwarz, H. R.: Methode der Finiten Elemente. Stutgaart: Teubner 1991.
[9] Wriggers, P. et al.: Benchmark perforated tension strip. Communication in talk at ENUMATH Conference in Heidelberg, 1997. (See also http://www.ibnm.uni-hannover.de/Forschung/Paketantrag/Benchmarks/benchmark.html)

Jochen Alberty
Mathematisches Seminar, Bereich II,
Christian-Albrechts-Universität zu Kiel
Ludewig-Meyn-Str. 4 24098 Kiel
Germany
e-mail: ja@numerik.uni-kiel.de

Carsten Carstensen
Institute for Applied Mathematics and
Numerical Analysis,
Vienna University of Technology
Wiedner Hauptstraße 8-10/115
A-1040 Vienna
Austria
e-mail: carsten.carstensen@tuwien.ac.at

Stefan A. Funken
Mathematisches Seminar, Bereich II,
Christian-Albrechts-Universität zu Kiel
Ludewig-Meyn-Str. 4
24098 Kiel
Germany
e-mail: saf@numerik.uni-kiel.de

Roland Klose
Mathematisches Seminar, Bereich II,
Christian-Albrechts-Universität zu
Kiel Ludewig-Meyn-Str. 4
24098 Kiel
Germany
e-mail: rkl@numerik.uni-kiel.de