Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Computational and Applied Mathematics 192 (2006) 219–250
JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS
www.elsevier.com/locate/cam

# P2Q2Iso2D = 2D Isoparametric FEM in Matlab☆

S. Bartels, C. Carstensen*, A. Hecht

*Humboldt-Universität zu Berlin, Unter den Linden 6, D-10099 Berlin, Germany.*

Received 20 September 2004; received in revised form 25 April 2005

**Abstract**

A short Matlab implementation realizes a flexible isoparametric finite element method up to quadratic order for the approximation of elliptic problems in two-dimensional domains with curved boundaries. Triangles and quadrilaterals equipped with varying quadrature rules allow for mesh refinement. Numerical examples for the Laplace equation with mixed boundary conditions indicate the flexibility of isoparametric finite elements.


*Keywords:* Isoparametric finite elements; Implementation; Visualization

## 1. Introduction

Various software packages are available for the numerical approximation of elliptic boundary value problems by finite elements on grids consisting of triangles or parallelograms. Such methods are well understood and advanced techniques such as geometric grids, *hp*-methods, or adaptive mesh-refinement are well established [5,6,8]. In some applications one aims to approximate problems on rather general domains with a few degrees of freedom. Therefore, the approximation of nonpolygonal domains is an important issue. Isoparametric finite elements can recover domains with piecewise quadratic boundary exactly and are therefore a good tool to approximate elliptic problems on domains with piecewise smooth boundary. We present a short Matlab implementation of this finite element method for the Laplace equation in two dimensions which can easily be modified to more general, even nonlinear, elliptic boundary value

☆ Supported by the DFG Research Center MATHEON "Mathematics for key technologies" in Berlin.
* Corresponding author. Tel.: +49 30 2093 5489; fax: +49 30 2093 5444.
*E-mail addresses:* sba@math.hu-berlin.de (S. Bartels), cc@math.hu-berlin.de (C. Carstensen), hecht@math.hu-berlin.de (A. Hecht).

0377-0427/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2005.04.032

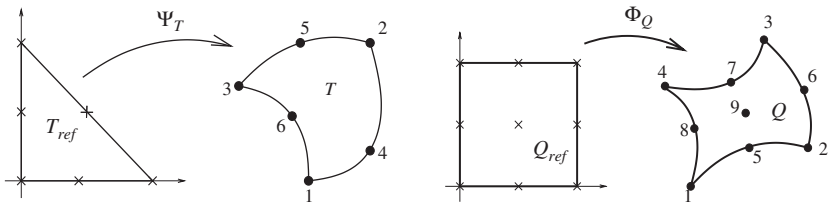


Fig. 1. Diffeomorphisms on the reference triangle $T_{\mathrm{ref}}$ onto a curved triangle $T$ (left) and on the reference square $Q_{\mathrm{ref}}$ onto a curved quadrilateral $Q$ (right).

problems. We refer to [4,10] for an introduction to isoparametric finite element methods, to [3,9] for the related blending function technique, and describe our program in the spirit of [1,2,7].

Wisdom from many practical computer experiments tells that quadratic finite elements are hard to beat (e.g. by *hp*-, adaptive, or other finite element schemes). Therefore, as a proposed method of choice, the employed data structure allows for the simultaneous usage of lowest order finite elements on triangles and parallelograms, of piecewise quadratic elements, and of curvilinear elements to resolve a piecewise quadratic boundary. The key concept is the definition of at most quadratic degree polynomial diffeomorphisms on a reference triangle or a reference square depicted in Fig. 1.

The diffeomorphisms are defined by specifying vertices of an element, optional nodes on the edges of an element, and optional nodes in the interior of elements with four vertices. Only two data files are needed to define lowest order elements, quadratic elements, and curvilinear elements with three or four vertices. Then, the isoparametric basis functions are given as

$$\varphi_j \circ \Psi_T \quad \text{or} \quad \psi_j \circ \Phi_Q$$

for a standard $P_2$ or $Q_2$ shape function on the reference element. This paper provides details on the implementation and quadrature rules for the stiffness matrices and right-hand sides.

The rest of the paper is organized as follows. We describe the model problem, the Laplace equation with mixed boundary conditions in two space dimensions, its weak formulation, and a general Galerkin scheme in Section 2. Section 3 defines admissible decompositions of Lipschitz domains that are the basis for the definition of the approximation scheme. Then, in Section 4 we present a procedure to compute the stiffness matrix and to incorporate volume forces as well as Neumann and Dirichlet boundary conditions. The numerical results of our Matlab tool applied to a stationary flow problem, the simulation of a semiconductor, and a problem from linear elasticity on a part of a disk with a corner singularity are shown in Section 5. Section 6 discusses the numerical realization of various quadrature rules. Finally, in Appendices A–C we present the entire Matlab code which consists of less than 400 lines using only standard Matlab commands for elementary matrix and list manipulations, comment on the realization of right-hand sides, and give a Matlab routine that displays the numerical solutions without artifacts. The software is downloadable at `http://www.math.hu-berlin.de/~cc/` under the item "Software".

## 2. Model problem and Galerkin approximation

Given a bounded Lipschitz domain $\Omega \subseteq \mathbb{R}^2$, a closed subset $\Gamma_D \subseteq \partial\Omega$ with positive length, and functions $f \in L^2(\Omega)$, $u_D \in H^1(\Omega)$, and $g \in L^2(\Gamma_N)$ for $\Gamma_N := \partial\Omega \backslash \Gamma_D$, the model problem under

consideration reads: Find $u \in H^1(\Omega)$ such that

$$-\Delta u = f \quad \text{in } \Omega, \quad u = u_D \quad \text{on } \Gamma_D, \quad \partial u/\partial n = g \quad \text{on } \Gamma_N. \tag{2.1}$$

We incorporate inhomogeneous Dirichlet conditions through a decomposition $v = u - u_D \in H^1_D(\Omega)$, where $H^1_D(\Omega) = \{w \in H^1(\Omega) : w|_{\Gamma_D} = 0\}$. Then, the weak formulation of (2.1) reads: Find $v \in H^1_D(\Omega)$ such that, for all $w \in H^1_D(\Omega)$, there holds

$$\int_\Omega \nabla v \cdot \nabla w \,\mathrm{d}x = \int_\Omega f w \,\mathrm{d}x + \int_{\Gamma_N} g w \,\mathrm{d}s - \int_\Omega \nabla u_D \cdot \nabla w \,\mathrm{d}x. \tag{2.2}$$

The Lax–Milgram Lemma guarantees existence of a unique solution $v \in H^1(\Omega)$ to (2.2). Here, we use standard notation for Lebesgue and Sobolev functions.

For a finite dimensional subspace $\mathcal{S} \subseteq H^1(\Omega)$ and an approximation $U_D \in \mathcal{S}$ of $u_D$ we define $\mathcal{S}_D := \mathcal{S} \cap H^1_D(\Omega)$ and aim to solve the following variational formulation: Find $V \in \mathcal{S}_D$ such that, for all $W \in \mathcal{S}_D$, there holds

$$\int_\Omega \nabla V \cdot \nabla W \,\mathrm{d}x = \int_\Omega f W \,\mathrm{d}x + \int_{\Gamma_N} g W \,\mathrm{d}s - \int_\Omega \nabla U_D \cdot \nabla W \,\mathrm{d}x. \tag{2.3}$$

For a basis $(N_z : z \in \mathcal{N})$ of $\mathcal{S}$ and a basis $(N_z : z \in \mathcal{K})$ of $\mathcal{S}_D$, with $\mathcal{K} \subseteq \mathcal{N}$, formulation (2.3) is equivalent to: Find $V \in \mathcal{S}_D$ such that, for all $z \in \mathcal{K}$, there holds

$$\int_\Omega \nabla V \cdot \nabla N_z \,\mathrm{d}x = \int_\Omega f N_z \,\mathrm{d}x + \int_{\Gamma_N} g N_z \,\mathrm{d}s - \int_\Omega \nabla U_D \cdot \nabla N_z \,\mathrm{d}x. \tag{2.4}$$

With the representations $V = \sum_{z\in\mathcal{K}} v_z N_z$ and $U_D = \sum_{z\in\mathcal{N}} u_z N_z$ formulation (2.4) leads to the linear system of equations

$$Avv = b, \tag{2.5}$$

where $A \in \mathbb{R}^{\mathcal{K}\times\mathcal{K}}$ and $b \in \mathbb{R}^{\mathcal{K}}$ are given by

$$A = \left( \int_\Omega \nabla N_z \cdot \nabla N_{z'} \,\mathrm{d}x \right)_{z,z'\in\mathcal{K}} \tag{2.6}$$

and

$$b = \left( \int_\Omega f N_z \,\mathrm{d}x + \int_{\Gamma_N} g N_z \,\mathrm{d}s - \sum_{z'\in\mathcal{N}} u_{z'} \int_\Omega \nabla N_{z'} \cdot \nabla N_z \,\mathrm{d}x \right)_{z\in\mathcal{K}}. \tag{2.7}$$

Then, $A$ is a positive definite matrix and there exists unique $v \in \mathbb{R}^{\mathcal{K}}$ which defines an approximation $U = V + U_D \in \mathcal{S}$ of the solution of (2.2).

## 3. Decomposition of $\Omega$ and data representation

### 3.1. Curved edges

We assume that $\overline{\Omega}$ is decomposed into finitely many finite element domains $T \in \mathscr{T}$ with curved boundaries and which either have three or four vertices. To guarantee that neighboring elements match we suppose that each of the four or three edges (or sides) of elements with, respectively, four or three vertices are defined through a reference parameterization. If $A$ and $B$ are the endpoints of an edge $E$ which may be curvilinear with a point $C$ on $E$ then $E$ is given by the parameterization

$$\Phi_E : E_{\mathrm{ref}} \to \mathbb{R}^2, \quad t \mapsto A(1-t)/2 + B(1+t)/2 + \tilde{C}(1-t)(1+t), \tag{3.1}$$

where $\tilde{C} = C - (A+B)/2$ and $E_{\mathrm{ref}} = [-1, 1]$ as in Fig. 2. We will assume that the restriction of $\Phi_E$ to $(-1, 1)$ is an immersion. This is guaranteed if $A$, $B$, and $C$ are distinct and either $C$ lies on the line segment connecting $A$ and $B$ or $A$, $B$, and $C$ are not colinear.

### 3.2. Curved quadrilaterals

Given any element $T \in \mathscr{T}$ with four vertices $P_1^{(T)}$, $P_2^{(T)}$, $P_3^{(T)}$, and $P_4^{(T)}$ in the plane, the boundary $\partial T$ consists of four smooth parameterized curves. Those curves interpolate two vertices $A = P_j^{(T)}$ and $B = P_{(j+1)/4}^{(T)}$, where $(j+1)/4$ is the remainder after division of $j+1$ by 4, and a given point $C = P_{j+4}^{(T)}$ as in (3.1). Moreover, we allow a node $P_9^{(T)}$ in the interior of $T$. The nodes $P_5^{(T)}, \ldots, P_9^{(T)}$ are optional; if for $j \in \{1, \ldots, 4\}$ the node $P_{j+4}^{(T)}$ is not initially specified it is obtained by linear interpolation of $P_j^{(T)}$

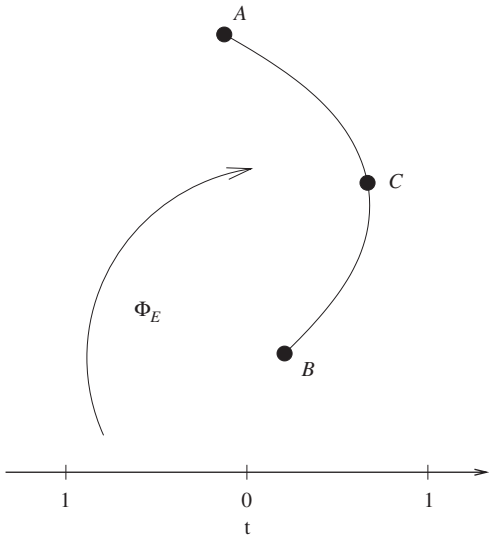


Fig. 2. Immersion $\Phi_E$ that parameterizes an edge $E$ defined through the points $A$, $B$, $C$.

and $P^{(T)}_{(j+1)/4}$, i.e.,

$$P^{(T)}_{j+4} := (P^{(T)}_j + P^{(T)}_{(j+1)/4})/2.$$

Similarly, if $P^{(T)}_9$ is not specified initially then we employ

$$P^{(T)}_9 := -(P^{(T)}_1 + P^{(T)}_2 + P^{(T)}_3 + P^{(T)}_4)/4 + (P^{(T)}_5 + P^{(T)}_6 + P^{(T)}_7 + P^{(T)}_8)/2.$$

For a representation of the elements with four vertices we define a reference element $Q_{\mathrm{ref}}$ and functions $\varphi_1, \ldots, \varphi_9 \in H^1(Q_{\mathrm{ref}})$ such that each element $T \in \mathscr{T}$ with four vertices is the image of the map

$$\Phi_T = \sum_{j=1}^{9} C^{(T)}_j \varphi_j : Q_{\mathrm{ref}} \to \mathbb{R}^2. \tag{3.2}$$

The coefficients $C^{(T)}_1, \ldots, C^{(T)}_9 \in \mathbb{R}^2$ are related to the given vertices $P^{(T)}_1, \ldots, P^{(T)}_4$ of $T$, to the points $P^{(T)}_5, \ldots, P^{(T)}_8$ (either initially prescribed or obtained by interpolation) on the boundary of $T$, and to the midpoint $P^{(T)}_9$ (either initially prescribed or obtained by interpolation) of $T$ for $j = 1, \ldots, 4$ in the following way,

$$\begin{aligned}
C^{(T)}_j &:= P^{(T)}_j,\\
C^{(T)}_{j+4} &:= P^{(T)}_{j+4} - (P^{(T)}_j + P^{(T)}_{(j+1)/4})/2,\\
C^{(T)}_9 &:= P^{(T)}_9 + (P^{(T)}_1 + P^{(T)}_2 + P^{(T)}_3 + P^{(T)}_4)/4 - (P^{(T)}_5 + P^{(T)}_6 + P^{(T)}_7 + P^{(T)}_8)/2.
\end{aligned} \tag{3.3}$$

Note that $C^{(T)}_{j+4} = 0$ if $P^{(T)}_{j+4}$ is not initially specified or if it is the midpoint of the line segment connecting $P^{(T)}_j$ and $P^{(T)}_{(j+1)/4}$. Similarly, $C^{(T)}_9 = 0$ if $P^{(T)}_9$ is not initially specified or if, e.g., $T$ is a square and $P^{(T)}_9$ is the midpoint of $T$.

For $Q_{\mathrm{ref}} := [-1, 1]^2$ the functions $\varphi_1, \ldots, \varphi_9$ are defined by

$$\begin{aligned}
&\varphi_1(\xi,\eta) := (1-\xi)(1-\eta)/4, && \varphi_2(\xi,\eta) := (1+\xi)(1-\eta)/4,\\
&\varphi_3(\xi,\eta) := (1+\xi)(1+\eta)/4, && \varphi_4(\xi,\eta) := (1-\xi)(1+\eta)/4,\\
&\varphi_5(\xi,\eta) := (1-\xi^2)(1-\eta)/2, && \varphi_6(\xi,\eta) := (1+\xi)(1-\eta^2)/2,\\
&\varphi_7(\xi,\eta) := (1-\xi^2)(1+\eta)/2, && \varphi_8(\xi,\eta) := (1-\xi)(1-\eta^2)/2,\\
&\varphi_9(\xi,\eta) := (1-\xi^2)(1-\eta^2).
\end{aligned}$$

Note that owing to this definition the vertices of $T_{\mathrm{ref}}$ are mapped to the vertices of $T$. Fig. 3 displays the functions $\varphi_1$, $\varphi_5$, and $\varphi_9$.

### 3.3. Curved triangles

Given any element $T \in \mathscr{T}$ with three prescribed vertices $P^{(T)}_1, P^{(T)}_2, P^{(T)}_3$, we assume that the boundary $\partial T$ consists of three smooth parameterized curves. Each of those curves interpolates vertices $A = P^{(T)}_j$ and $B = P^{(T)}_{(j+1)/3}$, where $(j+1)/3$ denotes the remainder after division of $j+1$ by 3, and a point $C = P^{(T)}_{j+3}$

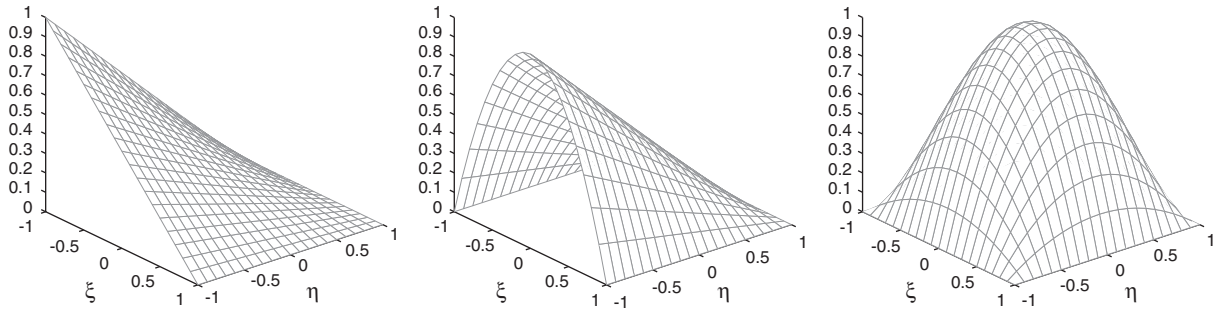


Fig. 3. Shape functions $\varphi_1$ (left), $\varphi_5$ (middle), and $\varphi_9$ (right) on the reference square.

as in (3.1). The nodes $P_4^{(T)}$, $P_5^{(T)}$, $P_6^{(T)}$ are optional; if for $j \in \{1, 2, 3\}$ the point $P_{j+3}^{(T)}$ is not initially specified it is obtained by linear interpolation of $P_j^{(T)}$ and $P_{(j+1)/3}^{(T)}$, i.e.,

$$P_{j+3}^{(T)} := \left(P_j^{(T)} + P_{(j+1)/3}^{(T)}\right) \Big/ 2.$$

For a representation of the elements with three vertices we define a reference element $T_{\mathrm{ref}}$ and functions $\varphi_1, \ldots, \varphi_6 \in H^1(T_{\mathrm{ref}})$ such that each element $T \in \mathcal{T}$ with three vertices is the image of the map

$$\Psi_T = \sum_{j=1}^{6} D_j^{(T)} \psi_j : T_{\mathrm{ref}} \to \mathbb{R}^2. \tag{3.4}$$

The coefficients $D_1^{(T)}, \ldots, D_6^{(T)} \in \mathbb{R}^2$ are related to the given vertices $P_1^{(T)}$, $P_2^{(T)}$, $P_3^{(T)}$ of $T$ and to the nodes $P_4^{(T)}$, $P_5^{(T)}$, $P_6^{(T)}$ (either initially prescribed or obtained by interpolation) on the boundary of $T$ for $j = 1, 2, 3$ in the following way,

$$D_j^{(T)} := P_j^{(T)} \text{ and } D_{j+3}^{(T)} := P_{j+3}^{(T)} - \left(P_j^{(T)} + P_{(j+1)/3}^{(T)}\right) \Big/ 2. \tag{3.5}$$

Note that $D_{j+3}^{(T)} = 0$ if $P_{j+3}^{(T)}$ is not initially specified or if it is the midpoint of the line segment connecting $P_j^{(T)}$ and $P_{(j+1)/3}^{(T)}$. We choose $T_{\mathrm{ref}} := \{(r, s) \in \mathbb{R}^2 : r, s \geqslant 0,\ r + s \leqslant 1\}$ and define

$$\begin{aligned}
&\psi_1(r, s) := 1 - r - s, \quad \psi_2(r, s) := r,\\
&\psi_3(r, s) := s, \quad \psi_4(r, s) := 4r(1 - r - s),\\
&\psi_5(r, s) := 4rs, \quad \psi_6(r, s) := 4s(1 - r - s).
\end{aligned}$$

As in the case of elements with four vertices, the vertices of $T_{\mathrm{ref}}$ are mapped to the vertices of $T$ under the mapping $\Psi_T$. Fig. 4 displays the functions $\psi_1$ and $\psi_4$.

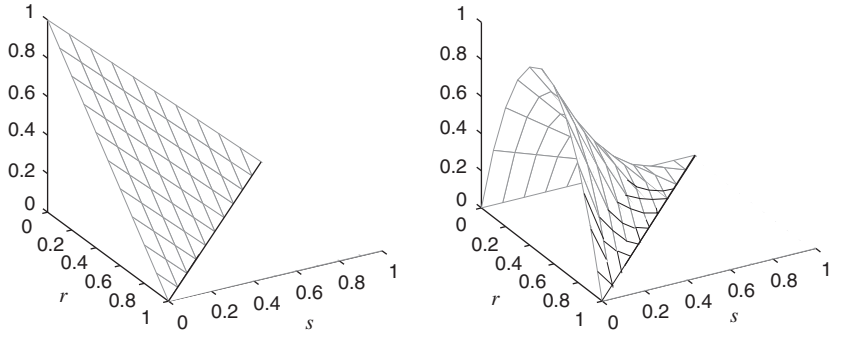


Fig. 4. Shape functions $\psi_1$ (left) and $\psi_4$ (right) on the reference triangle.

### 3.4. Assumptions on the triangulation to ensure $C^0$ conformity

We make the following assumptions on the triangulation $\mathcal{T}$ which imply restrictions on the choice of the vertices, points on the sides of elements, and points in the interior of elements. The assumptions imply that the elements with three and four vertices define a proper decomposition of $\Omega$ in the sense that edges (or sides) of neighboring elements match and that the mappings $\Phi_T$ and $\Psi_T$ are diffeomorphisms.

(1) (a) There exist $\mathcal{T}_4, \mathcal{T}_3 \subseteq \mathcal{T}$ such that $\mathcal{T}_4 \cup \mathcal{T}_3 = \mathcal{T}$ and $\mathcal{T}_4 \cap \mathcal{T}_3 = \emptyset$.
   (b) For each $T \in \mathcal{T}_4$ there exist $\{1, \dots, 4\} \subseteq J_T \subseteq \{1, \dots, 9\}$ and initially prescribed points $P_j^{(T)} \in \mathbb{R}^2$, $j \in J_T$, such that $T$ is the image of $Q_{\mathrm{ref}}$ under $\Phi_T$.
   (c) For each $T \in \mathcal{T}_3$ there exist $\{1, 2, 3\} \subseteq K_T \subseteq \{1, \dots, 6\}$ and initially prescribed points $P_j^{(T)} \in \mathbb{R}^2$, $j \in K_T$, such that $T$ is the image of $T_{\mathrm{ref}}$ under $\Psi_T$.
(2) The closure of $\Omega$ is covered exactly by $\mathcal{T}$, i.e., $\overline{\Omega} = \bigcup_{T \in \mathcal{T}} T$ and the interior of the elements is non-intersecting, i.e., $\mathrm{int}(T) \cap \mathrm{int}(T') = \emptyset$ for all $T, T' \in \mathcal{T}$.
(3) If $T \cap T' = \{x\}$ for $T, T' \in \mathcal{T}$ and some $x \in \mathbb{R}^2$ then $x$ is a vertex of both elements $T$ and $T'$.
(4) If $T \cap T' \supseteq \{x, y\}$ for $T, T' \in \mathcal{T}$ and distinct $x, y \in \mathbb{R}^2$ then $T$ and $T'$ share an entire side.
(5) There exists $c > 0$ such that $|\det D\Phi_T| > c$ for all $T \in \mathcal{T}_4$ and $|\det D\Psi_T| > c$ for all $T \in \mathcal{T}_3$.
(6) The parts $\Gamma_D$ and $\overline{\Gamma}_N$ of the boundary $\partial\Omega$ are matched exactly by the union of entire sides of elements.

### 3.5. Data structures

The relevant information about the elements $T \in \mathcal{T}$ are stored in three data files. The file `coordinates.dat` contains the coordinates of the vertices, the nodes defining the sides of the elements, and the nodes in the interior of the elements. Hence, `coordinates.dat` is a table with two columns which define the coordinates of the points. A numbering of these initially prescribed points is defined by the numbers of the corresponding rows in the file.

The files `elements4.dat` and `elements3.dat` specify the elements with four and three vertices, i.e., the elements in $\mathcal{T}_4$ and $\mathcal{T}_3$, through the numbers of the points, respectively, through the number of the corresponding row, in `coordinates.dat` (Fig. 5).

coordinates.dat

```
 0.0    0.0
 1.0    0.0
 1.0    1.0
 0.0    1.0
 0.6    0.08
 1.1    0.6
 0.6    0.92
-0.2    0.4
 0.5    0.6
 2.0   -0.5
 2.0    1.5
-1.0   -0.5
-1.0    0.5
-1.0    1.5
-0.75   1.0
-1.25   0.0
```

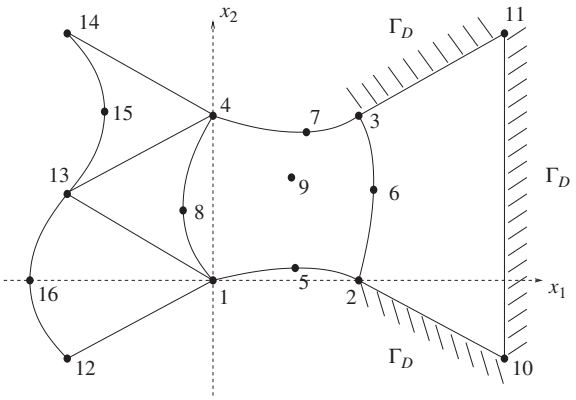


Fig. 5. Example of an admissible triangulation. On the left is the content of the complete file `coordinates.dat`.

Hence, each row in the file `elements4.dat` has nine entries. We use the convention that the first to fourth entries are positive integers that specify the vertices $P_1^{(T)}$, $P_2^{(T)}$, $P_3^{(T)}$, and $P_4^{(T)}$, respectively, of the element in mathematical positive orientation. The fifth to eighth entries are nonnegative integers which are either positive and then specify a point $P_5^{(T)}$, $P_6^{(T)}$, $P_7^{(T)}$, or $P_8^{(T)}$, respectively, on a side of an element by the coordinates given in the file `coordinates.dat` or it is zero which means that it is not specified. Similarly, the ninth entry is a nonnegative integer which is either a positive number and then defines $P_9^{(T)}$ or it is zero.

Analogously, each row in the file `elements3.dat` has six entries. We use the convention that the first to third entries are positive integers that specify the vertices $P_1^{(T)}$, $P_2^{(T)}$, and $P_3^{(T)}$, respectively, of the element in mathematical positive orientation. The fourth to sixth entries are nonnegative integers which are either positive and then specify the points $P_4^{(T)}$, $P_5^{(T)}$, or $P_6^{(T)}$, respectively, on a side of an element or it is zero which means that it is not specified. The following two files define the five elements shown in Fig. 3.

```
elements4.dat                    elements3.dat
1   2    3   4  5  6  7  8  9    1  13   12  0  16   0
2  10   11   3  0  0  0  6  0    1   4   13  8   0   0
                                  4  14   13  0  15   0
```

Finally, we define the parts $\Gamma_D$ and $\Gamma_N$ of the boundary through files `Dirichlet.dat` and `Neumann.dat`. We define each curve which is a side of an element on $\partial\Omega$ by specifying the points that define the curve through providing the numbers of the end-points and the possible point on the curve. Note that by assumptions on the triangulation this curve has to be an entire side of an element so that the third point is specified, i.e., the third entry of the corresponding row in the file is positive, if and only if it was used to define a side of an element. The files `Dirichlet.dat` and `Neumann.dat` define $\Gamma_D$ and $\Gamma_N = \partial\Omega\backslash\Gamma_D$ from Fig. 3.

```
Dirichlet.dat Neumann.dat
 2 10 0  3  4  7
10 11 0  4 14  0
11  3 0 14 13 15
       13 12 16
       12  1  0
        1  2  5
```

### 3.6. Subordinated Ansatz space

With the help of the diffeomorphisms $\Phi_T$ and $\Psi_T$ for $T \in \mathscr{T}_4$ and $T \in \mathscr{T}_3$, respectively, and the functions $\varphi_1, \ldots, \varphi_9$ and $\psi_1, \ldots, \psi_6$ we define a discrete subspace $\mathscr{S} \subseteq H^1(\Omega)$ as follows. The union of all positive numbers occurring in the files `elements4.dat` and `elements3.dat` defines the set of nodes $\mathscr{N}$, i.e.,

$$\mathscr{N} := \left\{ z \in \mathbb{R}^2 : \exists T \in \mathscr{T}_4 \, \exists j \in J_T, \, z = P_j^{(T)} \right\} \cup \left\{ z \in \mathbb{R}^2 : \exists T \in \mathscr{T}_3 \, \exists j \in K_T, \, z = P_j^{(T)} \right\}.$$

Given a node $z \in \mathscr{N}$, an element $T \in \mathscr{T}_4$ and $j \in J_T$ or $T \in \mathscr{T}_3$ and $j \in K_T$, such that $z = P_j^{(T)} \in T$ we define

$$N_z|_T := \begin{cases} \varphi_j \circ \Phi_T^{-1} & \text{if } z \in T \text{ and } T \in \mathscr{T}_4, \\ \psi_j \circ \Psi_T^{-1} & \text{if } z \in T \text{ and } T \in \mathscr{T}_3, \\ 0 & \text{if } z \notin T. \end{cases}$$

One easily checks $N_z \in H^1(\Omega)$. Then, $\mathscr{S}$ consists of all functions which are linear combinations of functions $N_z$,

$$\begin{aligned} \mathscr{S} &:= \left\{ \sum_{z \in \mathscr{N}} \alpha_z N_z : \forall z \in \mathscr{N}, \alpha_z \in \mathbb{R} \right\} \\ &= \left\{ v \in H^1(\Omega) : \forall T \in \mathscr{T}_4 \forall j \in J_T \exists \beta_j \in \mathbb{R}, v|_T = \sum_{j \in J_T} \beta_j \varphi_j \circ \Phi_T^{-1}, \right. \\ &\qquad \left. \forall T \in \mathscr{T}_3 \forall j \in K_T \exists \gamma_j \in \mathbb{R}, v|_T = \sum_{j \in K_T} \gamma_j \psi_j \circ \Psi_T^{-1} \right\}. \end{aligned}$$

Letting $\mathscr{K} := \mathscr{N} \backslash \Gamma_D$, the space $\mathscr{S}_D = \mathscr{S} \cap H_D^1(\Omega)$ is the span of all $N_z$ with $z \in \mathscr{K}$,

$$\mathscr{S}_D := \left\{ \sum_{z \in \mathscr{K}} \alpha_z N_z : \forall z \in \mathscr{K}, \alpha_z \in \mathbb{R} \right\}.$$

## 4. Computation of the discrete system

To compute the entries of the matrix $A$ in (2.6) we have to calculate the integrals

$$\int_\Omega \nabla N_z \cdot \nabla N_{z'}\,\mathrm{d}x = \sum_{T\in\mathscr{T}} \int_T \nabla N_z \cdot \nabla N_{z'}\,\mathrm{d}x$$

for all $z, z' \in \mathscr{K}$. Since $\operatorname{supp} N_z \cap \operatorname{supp} N_{z'} \neq \emptyset$ if and only if $z$ and $z'$ belong to the same element it suffices to compute for each $T \in \mathscr{T}_4$ the matrix $M^{(T)} = (M^{(T)}_{jk})_{j,k\in J_T}$ defined by

$$M^{(T)}_{jk} = \int_T \nabla(\varphi_j \circ \Phi_T^{-1}) \cdot \nabla(\varphi_k \circ \Phi_T^{-1})\,\mathrm{d}x$$

and for each $T \in \mathscr{T}_3$ the matrix $M^{(T)} = (M^{(T)}_{jk})_{j,k\in K_T}$ defined by

$$M^{(T)}_{jk} = \int_T \nabla(\psi_j \circ \Psi_T^{-1}) \cdot \nabla(\psi_k \circ \Psi_T^{-1})\,\mathrm{d}x.$$

We will compute matrices $M^{(T)} \in \mathbb{R}^{9\times 9}$ and $M^{(T)} \in \mathbb{R}^{6\times 6}$ for $T \in \mathscr{T}_4$ and $T \in \mathscr{T}_3$ and then use only those entries of $M^{(T)}$ that correspond to indices $j, k \in J_T$ and $j, k \in K_T$, respectively.

### 4.1. Local stiffness matrix for elements with four vertices

Employing the substitution rule for the diffeomorphism $\Phi_T : Q_{\mathrm{ref}} \to T$ and using the identity $(D\Phi_T^{-1}) \circ \Phi_T = (D\Phi_T)^{-1}$ we have

$$\begin{aligned} M^{(T)}_{jk} &= \int_T \nabla(\varphi_j \circ \Phi_T^{-1}) \cdot \nabla(\varphi_k \circ \Phi_T^{-1})\,\mathrm{d}x \\ &= \int_{Q_{\mathrm{ref}}} \nabla(\varphi_j \circ \Phi_T^{-1})(\Phi_T(\xi,\eta))(\nabla(\varphi_k \circ \Phi_T^{-1})(\Phi_T(\xi,\eta)))^{\mathrm{T}} |\det D\Phi_T(\xi,\eta)|\,\mathrm{d}(\xi,\eta) \\ &= \int_{Q_{\mathrm{ref}}} (\nabla\varphi_j(\xi,\eta) D\Phi_T(\xi,\eta)^{-1})(\nabla\varphi_k(\xi,\eta) D\Phi_T(\xi,\eta)^{-1})^{\mathrm{T}} |\det D\Phi_T(\xi,\eta)|\,\mathrm{d}(\xi,\eta). \end{aligned}$$

In order to evaluate $D\Phi_T$ we temporarily compute missing, i.e., not initially specified, points $P^{(T)}_{j+4}$ for $j = 1,\dots,4$ and $P^{(T)}_9$ by interpolation. The interpolation coefficients are stored in the array `K` for nodes $P^{(T)}_5, \dots, P^{(T)}_8$ on the boundary of $T$ and the coefficients to compute the possibly missing point $P^{(T)}_9$ inside $T$ are contained in `L`.

```
K=[1,1,0,0;0,1,1,0;0,0,1,1;1,0,0,1]/2;
L=[-1,-1,-1,-1,2,2,2,2]/4;
```

The boolean arrays `(elements4(j,5:8)==0)'*[1,1]` and `(elements4(j,9)==0)'*[1,1]` guarantee that only those nodes are interpolated which are missing.

```
J_T=find(elements4(j,:));
P=zeros(9,2);
P(J_T,:)=coordinates(elements4(j,J_T),:);
P(5:8,:)=P(5:8,:) + ((elements4(j,5:8)==0)'* [1,1]) .* (K * P(1:4,:));
P(9,:)=P(9,:) + ((elements4(j,9)==0)'* [1,1]) .* (L * P(1:8,:));
```

Having the complete set of points, we compute the coefficients $C_j^{(T)}$, $j = 1, \ldots, 9$.

```
C(1:4,:)=P(1:4,:);
C(5:8,:)=P(5:8,:) - (K * P(1:4,:));
C(9,:)=P(9,:) - (L * P(1:8,:));
```

The local stiffness matrix is then approximated using a quadrature rule on $Q_{\mathrm{ref}}$ which is defined by points $(\xi_m, \eta_m) \in Q_{\mathrm{ref}}$ and weights $\gamma_m$ for $m = 1, \ldots, K_4$,

$$M_{jk}^{(T)} \approx \sum_{m=1}^{K_4} \gamma_m (\nabla \varphi_j(\xi_m, \eta_m) D\Phi_T(\xi_m, \eta_m)^{-1})(\nabla \varphi_k(\xi_m, \eta_m) D\Phi_T(\xi_m, \eta_m)^{-1})^{\mathrm{T}} \times |\det D\Phi_T(\xi_m, \eta_m)|.$$

We suppose that the values $\varphi_j(\xi_m, \eta_m)$, $\partial_\xi \varphi_j(\xi_m, \eta_m)$, and $\partial_\eta \varphi_j(\xi_m, \eta_m)$, for $j = 1, \ldots, 9$ and $m = 1, \ldots, K_4$ are stored in $K_4 \times 9$ arrays `phi`, `phi_xi`, and `phi_eta`, respectively. The weights are contained in the $1 \times K_4$ array `gamma`. This allows to compute $M^{(T)}$ in a loop over $m = 1, \ldots, K_4$ simultaneously for $j, k = 1, \ldots, 9$.

```
for m=1 : size(gamma,2)
  D_Phi=[phi_xi(m,:);phi_eta(m,:)] * C;
  F=inv(D_Phi) * [phi_xi(m,:);phi_eta(m,:)];
  det_D_Phi(m)=abs(det(D_Phi));
  M=M + gamma(m) * (F' * F) * det_D_Phi(m);
end
```

Since we do not incorporate functions that correspond to interpolated auxiliary points, we only add those components of $M^{(T)}$ to the global stiffness matrix $A$ that were originally prescribed and which are stored in the array `J_T`. Note that the union of all positive entries in `J_T` equals the set $J_T$.

```
A(elements4(j,J_T),elements4(j,J_T))=...
 A(elements4(j,J_T),elements4(j,J_T)) + M(J_T,J_T);
```

### 4.2. Local stiffness matrix for elements with three vertices

For an element $T \in \mathcal{T}_3$ there holds

$$M_{jk}^{(T)} = \int_{T_{\mathrm{ref}}} (\nabla\psi_j(r,s) D\Psi_T(r,s)^{-1})(\nabla\psi_k(r,s) D\Psi_T(r,s)^{-1})^{\mathrm{T}} |\det D\Psi_T(r,s)| \,\mathrm{d}(r,s).$$

To compute $M^{(T)}$ we first interpolate missing points $P_{j+3}^{(T)}$ for $j = 1, 2, 3$ employing interpolation coefficients that are stored in an array `N`.

```
N=[1,1,0;0,1,1;1,0,1]/2;
```

The boolean array `(elements3(j,4:6)==0)'*[1,1]` guarantees that only the missing points are interpolated.

```
K_T=find(elements3(j,:));
P=zeros(6,2);
P(K_T,:)=coordinates(elements3(j,K_T),:);
P(4:6,:)=P(4:6,:) + ((elements3(j,4:6)==0)'* [1,1]) .* (N * P(1:3,:)).
```

Then, the coefficients $D_j^{(T)}$, $j = 1, \ldots, 6$ are computed within the following two lines.

```
D(1:3,:)=P(1:3,:);
D(4:6,:)=P(4:6,:) - (N * P(1:3,:)).
```

The integrals that contribute to the local stiffness matrix are approximated using a quadrature rule on $T_{\mathrm{ref}}$ that is defined by points $(r_m, s_m) \in T_{\mathrm{ref}}$ and weights $\kappa_m$ for $m = 1, \ldots, K_3$,

$$M_{jk}^{(T)} \approx \sum_{m=1}^{K_3} \kappa_m (\nabla\psi_j(r_m,s_m) D\Psi_T(r_m,s_m)^{-1})(\nabla\psi_k(r_m,s_m) D\Psi_T(r_m,s_m)^{-1})^{\mathrm{T}} \times |\det D\Psi_T(r_m,s_m)|.$$

We assume that the values $\psi_j(r_m,s_m)$, $\partial_r\psi_j(r_m,s_m)$, and $\partial_s\psi_j(r_m,s_m)$ are stored in $K_3 \times 6$ arrays `psi`, `psi_r`, and `psi_s`, respectively. The weights are stored in the $1 \times K_3$ array `kappa`. For $m = 1, \ldots, K_3$ the contributions to the matrix $M^{(T)}$ are then computed simultaneously for $j, k = 1, \ldots, 6$ with the following commands.

```
for m=1 : size(kappa, 2)
  D_Psi=[psi_r(m,:);psi_s(m,:)] * D;
  F=inv(D_Psi) * [psi_r(m,:);psi_s(m,:)];
  det_D_Psi(m)=abs(det(D_Psi));
  M=M + kappa(m) * (F' * F) * det_D_Psi(m);
end
```

By our conventions, interpolated points are no nodes so that we only add those entries of `M` to the global stiffness matrix which correspond to initially defined points. The indices of those nodes are stored in the

array K_T. The positive entries in K_T are the elements of $K_T$.

```
A(elements3(j,K_T),elements3(j,K_T)) = ...
A(elements3(j,K_T),elements3(j,K_T)) + M(K_T,K_T);
```

### 4.3. Volume forces

The integral on the right hand side of (2.4) that involves the volume force $f$, i.e.,

$$\int_\Omega f N_z \,\mathrm{d}x = \sum_{T\in\mathscr{T}} \int_T f N_z \,\mathrm{d}x,$$

is computed in the same loops over elements in $\mathscr{T}_3$ and $\mathscr{T}_4$ as the local stiffness matrices $M^{(T)}$. We assume that $f$ is continuous and employ the same quadrature rules as above, i.e.,

$$\begin{aligned}\int_T f(x)\varphi_j \circ \Phi_T^{-1}(x)\,\mathrm{d}x &= \int_{Q_{\mathrm{ref}}} f\circ\Phi_T(\xi,\eta)\varphi_j(\xi,\eta)|\det D\Phi_T(\xi,\eta)|\,\mathrm{d}(\xi,\eta)\\ &\approx \sum_{m=1}^{K_4} \gamma_m f(\Phi_T(\xi_m,\eta_m))\varphi_j(\xi_m,\eta_m)|\det D\Phi_T(\xi_m,\eta_m)|\end{aligned}$$

if $T\in\mathscr{T}_4$ and

$$\int_T f(x)\varphi_j \circ \Psi_T^{-1}(x)\,\mathrm{d}x \approx \sum_{m=1}^{K_3} \kappa_m f(\Psi_T(r_m,s_m))\psi_j(r_m,s_m)|\det D\Psi_T(r_m,s_m)|$$

if $T\in\mathscr{T}_3$. The sum for $T\in\mathscr{T}_4$ is realized simultaneously for $j=1,\ldots,9$ and $m=1,\ldots,K_4$ within the following two lines. We add those contributions to $b$ that correspond to initially specified nodes.

```
d=gamma .* det_D_Phi .* f(phi * C)'* phi;
b(elements4(j,J_T))=b(elements4(j,J_T)) + d(J_T)';
```

Analogously, for $T\in\mathscr{T}_3$ we compute the contributions to $b$ in the following two lines:

```
d=kappa .* det_D_Psi .* f(psi * D)'* psi;
b(elements3(j,K_T))=b(elements3(j,K_T)) + d(K_T)';
```

In the above commands the function f.m returns the values of $f$ at a list of given points, see Appendix B.

### 4.4. Outer body forces

To incorporate Neumann boundary conditions we have to compute integrals of the form

$$\int_{\Gamma_N} gN_z|_{\Gamma_N}\,\mathrm{d}s = \sum_{T\in\mathscr{T}} \int_{\partial T\cap\Gamma_N} gN_z|_{\Gamma_N}\,\mathrm{d}s.$$

Each proper curve $E = \partial T \cap \Gamma_N$ is either defined through points $A = P_j^{(T)}$, $B = P_{(j+1)/4}^{(T)}$, and $C = P_{j+4}^{(T)}$ for some $j \in \{1, \dots, 4\}$ and $T \in \mathscr{T}_4$, or through points $A = P_j^{(T)}$, $B = P_{(j+1)/3}^{(T)}$, and $C = P_{j+3}^{(T)}$ for some $j \in \{1, 2, 3\}$ and $T \in \mathscr{T}_3$. Then, the curve $E$ is parameterized by

$$\Phi_E : E_{\mathrm{ref}} \to \mathbb{R}^2, \quad t \mapsto A(1-t)/2 + B(1+t)/2 + \tilde{C}(1-t)(1+t)$$

for $\tilde{C} = C - (A+B)/2$ and $E_{\mathrm{ref}} = [-1, 1]$. Note that $A$, $B$, and $C$ correspond to the points specified in the file `Neumann.dat`. The three functions $(1-t)/2$, $(1+t)/2$, and $(1-t)(1+t)$ coincide with the functions $\varphi_j(t, -1)$ for $j = 1, 2, 5$, respectively. We then compute, for all such $E$ and $j = 1, 2, 5$,

$$\int_E g(s)\varphi_j \circ \Phi_E^{-1}(s)\,\mathrm{d}s = \int_{(-1,1)} g(\Phi_E(t))\varphi_j(t, -1)\|\Phi_E'(t)\|\,\mathrm{d}t, \tag{4.1}$$

where $\|\cdot\|$ is the Euclidean norm. The latter integral is approximated by a quadrature rule on $E_{\mathrm{ref}}$ defined through points $t_m \in E_{\mathrm{ref}}$ and weights $\delta_m$ for $m = 1, \dots, K_N$. In a loop over all edges on $\Gamma_N$ we compute missing nodes, approximate the integral in (4.1), and add the numbers which correspond to initially defined points to the global vector $b$. The $K_N \times 3$ arrays `phi_E` and `phi_E_dt` contain the values $\varphi_j(t_m, -1)$ and $\partial_t\varphi_j(t_m, -1)$. The weights for the quadrature rule are stored in the $1 \times K_N$ array `delta_E`. Notice that in the practical realization below the coefficients $A$, $B$, and $\tilde{C}$ are stored in the array `G`.

```
J_E=find(Neumann(j,:));
P=zeros(3,2);
P(J_E,:)=coordinates(Neumann(j,J_E),:);
P(3,:)=P(3,:) + ((Neumann(j,3)==0)' * [1,1]) .* (P(1,:) +
P(2,:))/2;
G(1:2,:)=P(1:2,:);
G(3,:)=P(3,:) - (P(1,:) + P(2,:))/2;
norm_Phi_E_dt=sqrt(sum((phi_E_dt * C)'^2));
d=delta_E .* g(phi_E * G)' .* norm_Phi_E_dt * phi_E;
b(Neumann(j,J_E))=b(Neumann(j,J_E)) + d(J_E)';
```

As before, a function `g.m` returns the values of $g$ in a list of given points, see Appendix B.

### 4.5. Dirichlet conditions

To incorporate Dirichlet boundary conditions we define a function $U_D \in \mathscr{S}$,

$$U_D = \sum_{z \in \mathscr{N}} u_z N_z,$$

that satisfies $U_D(z) = u_D(z)$ for all $z \in \mathscr{N} \cap \Gamma_D$ and $U_D(z) = 0$ for all $z \in \mathscr{K}$. We set $u_z = 0$ for all $z \in \mathscr{K}$. Note that for $z \in \mathscr{N} \cap \Gamma_D$ which is a vertex of an element we have $N_x(z) = 0$ for all $x \in \mathscr{N}\backslash\{z\}$ so that $u_z = u_D(z)$. For $z \in \mathscr{N} \cap \Gamma_D$ which is not a vertex of an element there are exactly two functions $N_x$, $N_y$ for $x, y \in \mathscr{N}\backslash\{z\}$ such that $N_x(z) = N_y(z) \neq 0$ so that we have to set $u_z = u_D(z) - u_x N_x(z) - u_y N_y(z) = u_D(z) - (u_x + u_y)/2$. The following lines compute the coefficients

of the function $U_D$ in terms of the basis functions $N_z$.

```
u(unique(Dirichlet(:,1:2))) = u_D(coordinates(unique(Dirichlet(:,1:2)),:));
ind = find(Dirichlet(:,3));
u(Dirichlet(ind,3)) = u_D(coordinates(Dirichlet(ind,3),:)) - ...
  (u(Dirichlet(ind,1)) + u(Dirichlet(ind,2)))/2;
```

The function $u_D$ is represented by a file `u_D.m`, which returns the values of $u_D$ in a list of given points, see Appendix B. We then subtract $Au$ from the right-hand side as in (2.4).

```
b = b - A * u;
```

### 4.6. Solving the linear system of equations

Having computed the stiffness matrix and the right-hand side, the computation of the solution $V$ in the free nodes $\mathscr{K}$ is done in the next two lines.

```
freeNodes = setdiff(1:size(coordinates,1),unique(Dirichlet));
v(freeNodes) = A(freeNodes,freeNodes) \ b(freeNodes);
```

The Matlab command `x = A\b` efficiently solves a linear system of equations $Ax = b$.

### 4.7. Displaying the solution

The discrete solution can be visualized with curved edges by the functions `submeshplot3.m` for the triangles in the mesh and `submeshplot4.m` for the quadrilaterals. Since Matlab's internal functions fail to interpolate P2/Q2 polynomials reliably, this is performed on each triangle/quadrilateral. The solution is interpolated on a submesh with a chosen granularity to be adjusted to the complexity of the plot. The original mesh is added to the plot by the routine `drawgrid.m`, which draws the grid with the same granularity as the meshes. The program is shown in Appendix C.

## 5. Numerical examples

This section on seven illustrative examples concerns flows, semiconductors, corner singularities, curved boundaries, and hanging nodes.

### 5.1. Reduced flow problem

A reduced model for a stationary flow problem reads

$$-\Delta u = 0 \text{ in } \Omega,\ u = 1 \text{ on } \Gamma_1,\ u = -1 \text{ on } \Gamma_2,\ \partial u/\partial n = 0 \text{ on } \Gamma_N = \partial\Omega\backslash(\Gamma_1 \cup \Gamma_2).$$

Here, $\Omega := (0, 55) \times (0, 20)\backslash([20, 25] \times [15, 20] \cup [30, 35] \times [15, 20] \cup \{20, 35\} \times [10, 15])$. Note that $\Omega$ has two slits and is hence not a Lipschitz domain. We set $\Gamma_1 := [0, 20] \times \{20\}$ and $\Gamma_2 := [35, 55] \times \{20\}$. Fig. 6 shows the numerical solution on a triangulation with 126 elements, each of them with four vertices.

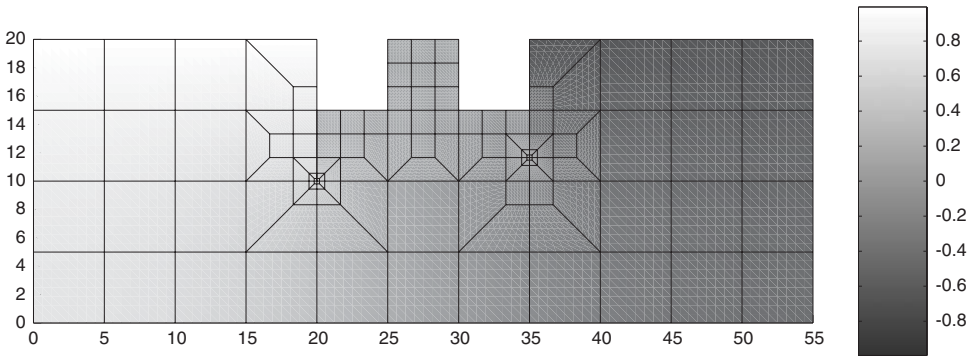


Fig. 6. Numerical solution for a reduced flow problem.

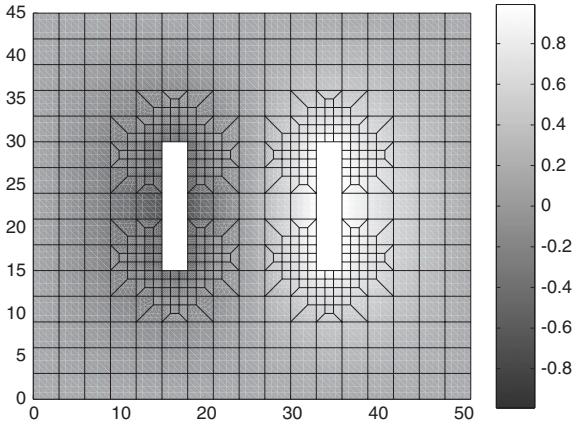


Fig. 7. Numerical solution for the simulation of a semiconductor.

### 5.2. Simulation of a semiconductor

The charge density in a semiconductor can be modeled by the equations

$$-\Delta u = 0 \text{ in } \Omega,\ u = -1 \text{ on } \Gamma_1,\ u = 1 \text{ on } \Gamma_2 \text{ and } u = 0 \text{ on } \Gamma_3$$

on the domain $\Omega = (0, 51) \times (0, 45) \backslash (([15, 18] \cup [33, 36]) \times [15, 30])$ and its boundary parts $\Gamma_1 = \partial([15, 18] \times [15, 30])$, $\Gamma_2 = \partial([33, 36] \times [15, 30])$, and $\Gamma_3 = [0, 51] \times \{0, 45\} \cup \{0, 51\} \times [0, 45]$. Fig. 7 shows the numerical solution on a triangulation with 565 elements each of them with four vertices.

### 5.3. Solution with a corner singularity

Fig. 8 shows the numerical solution for a subproblem in linear elasticity on a part of the unit disk. The gradient of the solution has a singularity at the origin. The problem is described by the equations

$$-\Delta u = 0 \text{ in } \Omega,\ u = 0 \text{ on } \Gamma_D \text{ and } \partial u/\partial n = g \text{ on } \Gamma_N.$$

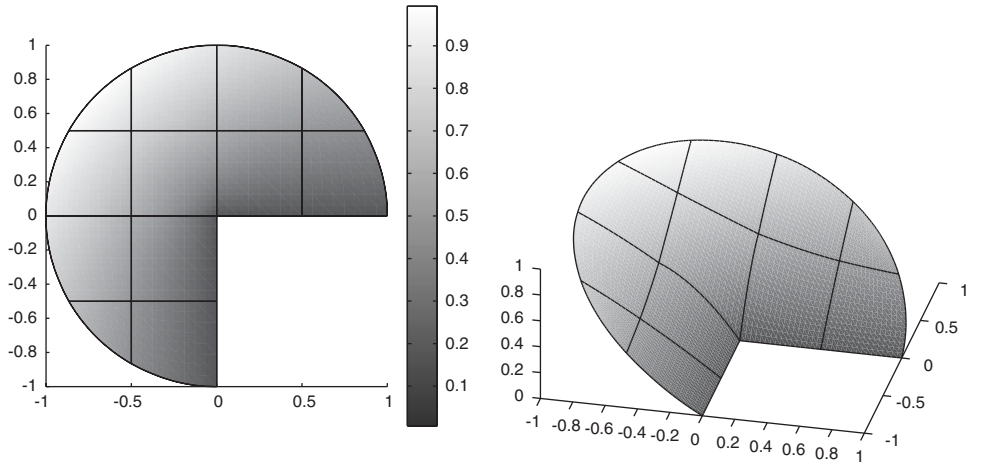


Fig. 8. Numerical solution of an example with a corner singularity.

Here, $\Omega := \{(x_1, x_2) \in \mathbb{R}^2 : x_1^2 + x_2^2 < 1\} \backslash [-1, 0]^2$, $\Gamma_D := [0, 1] \times \{0\} \cup \{0\} \times [-1, 0]$, and $\Gamma_N = \partial\Omega \backslash \Gamma_D$. The function $g$ is in polar coordinates given by

$$g(r, \phi) = (2/3) r^{-1/3} \sin(2\phi/3).$$

Note that in this example we cannot recover the domain $\Omega$ exactly, but very accurately with the non-affine elements.

### 5.4. Domain with piecewise curved boundary

Given the points $A_1 = (-41, -62)$, $A_2 = (41, -62)$, $A_3 = (41, -42)$, $A_4 = (51, -42)$, $A_5 = (71, 42)$, $A_6 = (-2.6386, 57.9227)$, $A_7 = (-41, -41)$, $A_8 = (0, -26)$, $A_9 = (0, -45)$, and $A_{10} = A_8$, and circles of radii $r_1 = 26$ and $r_2 = 41\sqrt{2}$ around $M_1 = (0, 0)$ and of radii $r_3 = 17$ and $r_4 = \sqrt{34^2 + 21^2}$ around $M_2 = (37, 63)$ the slid domain $\Omega$ is defined as in Fig. 9. The domain is discretized into 13 elements with four vertices as shown in the right plot of Fig. 9.

We used our program to solve the equations

$$\begin{aligned} &-\Delta u = 0 \quad \text{in } \Omega, \quad u = 0 \quad \text{on } \Gamma_D^1, \\ &u(x, y) = \operatorname{sign}(x)(26 - y)/52 \quad \text{for } (x, y) \in \Gamma_D^2, \\ &u = 1 \quad \text{on } \Gamma_D^3 \text{ and } \partial u/\partial n = 0 \quad \text{on } \Gamma_N = \partial\Omega \backslash (\Gamma_D^1 \cup \Gamma_D^2 \cup \Gamma_D^3). \end{aligned}$$

The numerical solution is displayed in Fig. 10.

### 5.5. Hanging nodes

The following example illustrates the possible treatment of hanging nodes in the algorithms of this paper. Given the mesh of Fig. 5, suppose that the first quadrilateral element with nodes $1, \ldots, 9$ is refined in four sub-quadrilaterals as shown in Fig. 11 with additional nodes $17, \ldots, 28$. The new data

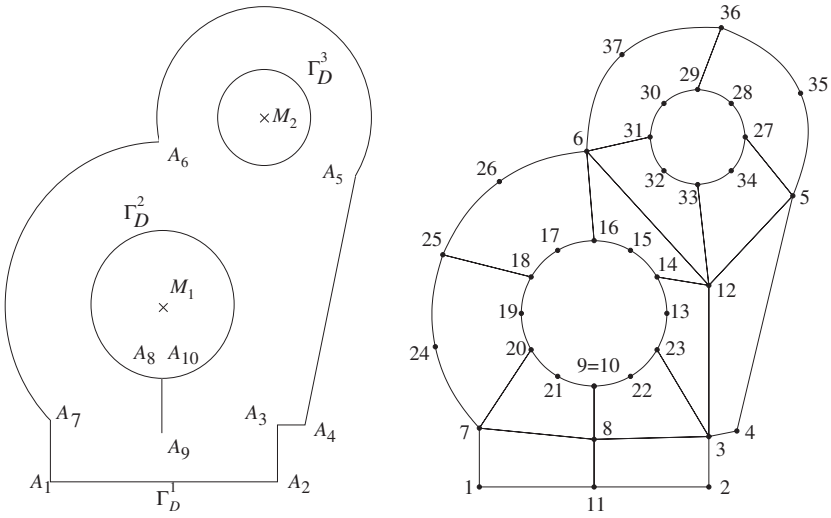


Fig. 9. Domain with piecewise curved boundary and its triangulation after [3].

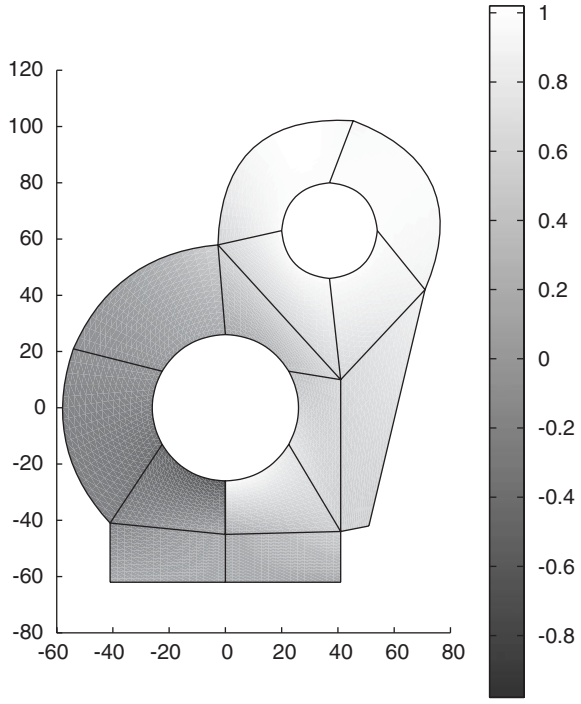


Fig. 10. Numerical solution for the problem defined on a domain with piecewise curved boundary.

file `coordinates.dat` is partly shown below, the complete matrix is obtained by concatenation of the entries `coordinates(1:16,:)` of Fig. 5 and `coordinates(17:28,:)` shown below.

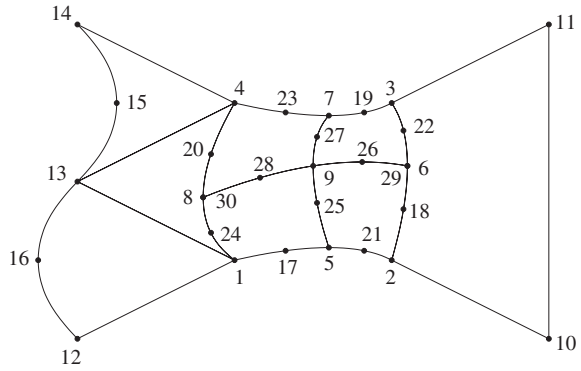


Fig. 11. Triangulation with hanging nodes of Section 5.5 based on a refinement of the triangulation of Fig. 5.

In the geometry at hand, the nodes 18, 22, 29 and 20, 24, 30 belong to a neighbor element of elements 2, 3, 4 and 5. Those nodes are called hanging nodes. The corresponding data files are displayed below with `elements3.dat` and `Dirichlet.dat` unchanged.

```
coordinates(17:32,:)
 0.325   0.06
 1.075   0.325
 0.825   0.94
−0.15    0.675
 0.825   0.06
 1.075   0.825
 0.325   0.94
−0.15    0.175
 0.525   0.365
 0.8125  0.625
 0.525   0.785
 0.1625  0.525
 1.1     0.6
−0.2     0.4
```

```
elements4.dat
 2   10   11    3    0    0    0    6   0
 1    5    9   30   17   25   28   24   0
 5    2   29    9   21   18   26   25   0
 9   29    3    7   26   22   19   27   0
30    9    7    4   28   27   23   20   0
```

```
Neumann.dat
3    7  19
7    4  23
4   14   0
```

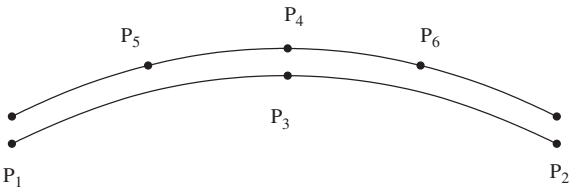


Fig. 12. Upper and lower side of a curved edge from Fig. 2 in the discussion of two types of hanging nodes.

```
14   13   15
13   12   16
12    1    0
 1    5   17
 5    2   21
```

The definition of a hanging node is in fact more complicated. The situation along the two edges with nodes 4, 8, 1 and 2, 6, 3 is depicted with $(P_1, P_2, P_3)$ in Fig. 12. The six nodes in Fig. 12 correspond to six degrees of freedom with nodal values at $P_1$, $P_2$ and $P_4$ and edge-midpoints at $P_3$, $P_5$ and $P_6$. The hierarchical organisation of the shape functions of Fig. 3 or Fig. 4 leads to a different treatment of the points $P_3$ and $P_4$. On the lower element, the degrees of freedom $V_1$, $V_2$, $V_3$ associated with $(P_1, P_2, P_3)$ determine the displacement $V(x(t))$ and the geometry of the edge $x(t)$ through

$$\begin{pmatrix} V(x(t)) \\ x(t) \end{pmatrix} = \begin{pmatrix} V_1 \\ P_1 \end{pmatrix} \frac{1-t}{2} + \begin{pmatrix} V_2 \\ P_2 \end{pmatrix} \frac{1+t}{2} + \left( \begin{pmatrix} V_3 \\ P_3 \end{pmatrix} - \frac{1}{2} \left( \begin{pmatrix} 0 \\ P_1 \end{pmatrix} + \begin{pmatrix} 0 \\ P_2 \end{pmatrix} \right) \right) (1-t)(1+t) \tag{5.1}$$

for any $x(t) = \Phi_E(t)$ on the edge parameterization as shown for $-1 \leqslant t \leqslant 1$ as in (3.1). A corresponding representation holds along the two edges $(P_1, P_4, P_5)$ and $(P_4, P_3, P_6)$ of Fig. 12.

The global continuity of the piecewise polynomial displacement $V$ requires that $P_5$ and $P_6$ belong to the edge $(P_1, P_2, P_3)$ and, for simplicity, we suppose that the corresponding parameterizations are $t = -\frac{1}{2}$ and $t = \frac{1}{2}$, namely

$$8P_5 = 3P_1 - P_2 + 6P_3 \text{ and } 8P_6 = 3P_2 - P_1 + 6P_3.$$

Then, the continuity condition at $P_3 = P_4$ (they share the coordinates, but refer to different degrees of freedom $V_3$ and $V_4$) reads

$$V_4 = \tfrac{1}{2}V_1 + \tfrac{1}{2}V_2 + V_3 \tag{5.2}$$

(because $V_4 = V(P_3)$ with $t = 0$ in (5.1). After some straightforward calculations on the continuity condition on $P_5$ and $P_6$, i.e. at $P_{j+4}$ for $j = 1$ or $j = 2$, respectively, one obtains

$$3V_j + V_{3-j} + 3V_3 = 4V_{j+4} + 2V_4 + 2V_j. \tag{5.3}$$

All three equations can be written as

$$\begin{bmatrix} 1 & 1 & 2 & -2 & 0 & 0 \\ 1 & 1 & 3 & -2 & -4 & 0 \\ 1 & 1 & 3 & -2 & 0 & -4 \end{bmatrix} \begin{bmatrix} V_1 \\ \vdots \\ V_6 \end{bmatrix} = 0.$$

The `3x6` dimensional coefficient matrix is called `M` in the Matlab code below. A Lagrange multiplier technique enforces the three conditions per hanging node in the discrete system.

The data for hanging nodes is stored in the file `hn.dat`. The six columns contain the degrees of freedom in the order depicted in Fig. 12, with one row for each hanging node. The data file corresponding to the mesh given in Fig. 5 is given below.

```
hn.dat
4   1   8   30   20   24
2   3   6   29   18   22
```

The Matlab realisation is given where only the additional lines are displayed. These are to be inserted between the Dirichlet conditions as described in Section 4.5 and the (modified) solution of the linear system of equations (Section 4.6).

If `lhn` hanging nodes are specified for a test problem, the stiffness matrix is augmented with matrices `B` and `B'`, and the solution vector is augmented by the `3*lhn` Lagrange multipliers, yielding the modified system

$$\begin{bmatrix} A & B' \\ B & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix},$$

where `B` has `3*lhn` rows, three for each hanging node, containing the entries of `M` on the columns of the nodes on the edge.

```
% Hanging nodes
  eval('load hanging_nodes.dat','hn=[];');
  if isempty(hn)
  M=[1,1,2,-2,0,0;1,1,3,-2,-4,0;1,1,3,-2,0,-4];
  B=sparse(3*size(hn,1), size(coordinates,1));
  for j=1:size(hn,1)
    B((1:3)+(j-1)*3,hn(j,:))=M;
  end
  lambdas=size(coordinates,1)+(1:3*size(hn,1));
  A=[A,B';B,sparse(3*size(hn,1), 3*size(hn,1))];
  b=[b;zeros(3*size(hn,1),1)];
  v=[v;zeros(3*size(hn,1),1)];
  else
  lambdas=[];
  end
% Compute solution in free nodes
  freeNodes=[setdiff(1:size(coordinates,1),unique(Dirichlet)),
lambdas];
```

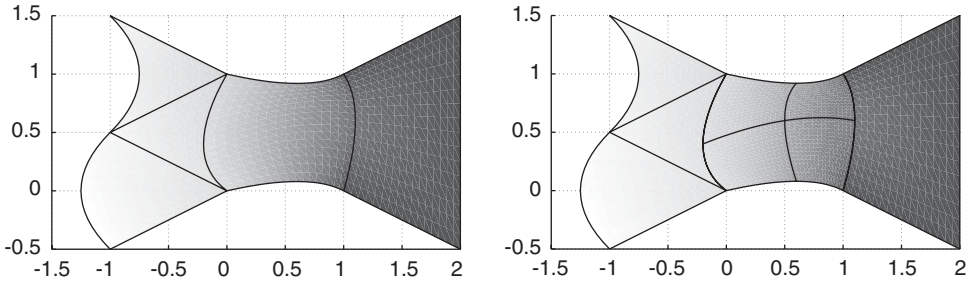


Fig. 13. Numerical solution without and with hanging nodes.

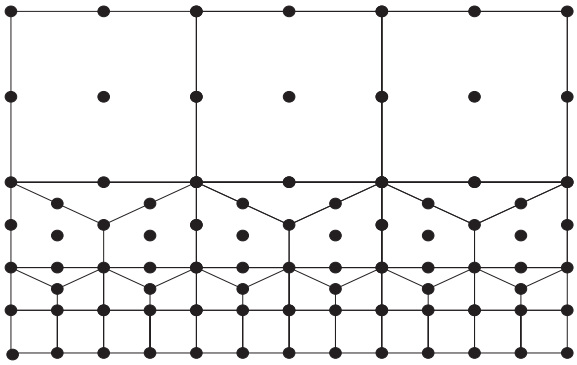

Fig. 14. Refinement of large elements with quadratic ansatz functions to smaller elements with bilinear ansatz functions.

Note that the indices of the Lagrange multipliers stored in `lambdas` contribute to the free nodes and are included in the solution of the linear system of equations.

The solution of the testcase without hanging nodes as depicted in Fig. 5 and the solution of the testcase with hanging nodes (Fig. 11) are shown for comparison in Fig. 13.

### *5.6. Locally refined triangulations*

Solutions of elliptic boundary value problems typically have singularities at re-entrant corners (cf. Example 5.3). The simultaneous usage of linear ansatz-functions on small elements and the usage of quadratic functions on larger elements can lead to very efficient approximations. In order to refine larger elements to smaller elements and to keep the conformity assumptions stated in Section 3.4 we propose to employ decompositions as in Fig. 14. This is an alternative to introducing hanging nodes.

## 6. Quadrature rules

This section defines some quadrature rules that can be employed in our Matlab code. More details and other quadrature rules can be found in [10]. The proposed routines provide the values needed for

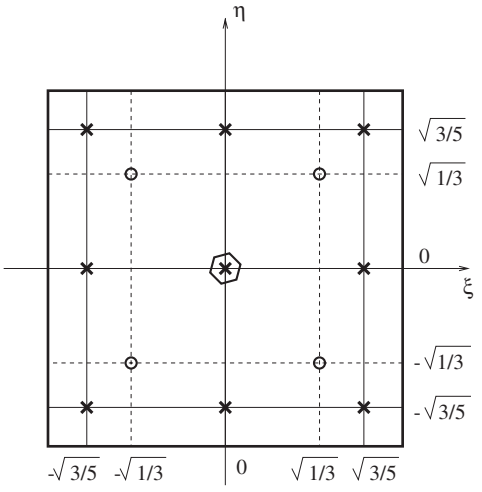


Fig. 15. Quadrature rules on $Q_{\mathrm{ref}}$ with one (hexagon), four (circles), and nine (crosses) nodes.

the approximation of local stiffness matrices and for the incorporation of volume forces and Neumann boundary conditions.

### 6.1. Quadrature rules on $Q_{\mathrm{ref}}$

We employ Gaussian quadrature rules with one, four, and nine nodes on the reference square $Q_{\mathrm{ref}}$. Fig. 15 displays the location of the points $(\xi_m, \eta_m)$, $m = 1, \ldots, K_4$, in $Q_{\mathrm{ref}}$ for $K_4 = 1, 4, 9$. The following function `quad4.m` computes the values $\varphi_j(\xi_m, \eta_m)$, $\partial_\xi \varphi_j(\xi_m, \eta_m)$, and $\partial_\eta \varphi_j(\xi_m, \eta_m)$ for $j = 1, \ldots, 9$, $m = 1, \ldots, K_4$, and stores them in $K_4 \times 9$ arrays `phi`, `phi_xi`, and `phi_eta`, respectively.

```
function [phi,phi_xi,phi_eta,gamma]=quad4(K_4);
    switch K_4
        Case 1
            xi=0;
            eta=0;
            gamma=4;
        Case 4
            xi=sqrt(1/3) * [-1,1,1,-1]';
            eta=sqrt(1/3) * [-1,-1,1,1]';
            gamma=[1,1,1,1];
        otherwise
            xi=sqrt(3/5) * [-1,0,1,-1,0,1,-1,0,1]';
            eta=sqrt(3/5) * [-1,-1,-1,0,0,0,1,1,1]';
            gamma=[25,40,25,40,64,40,25,40,25]/81;
    end
```

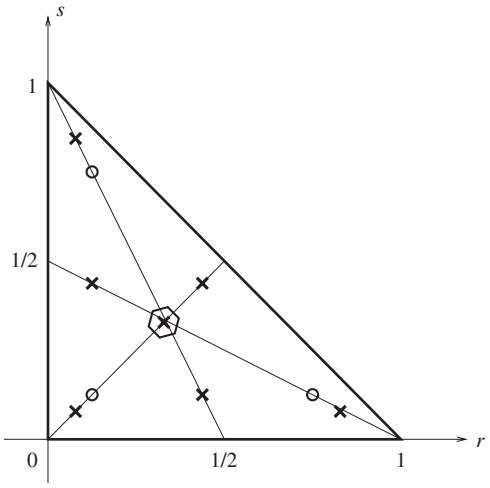


Fig. 16. Quadrature rules on $T_{\mathrm{ref}}$ with one (hexagon), three (circles), and seven (crosses) nodes.

```
phi = [(1-xi).*(1-eta)/2,(1+xi).*(1-eta)/2,...
   (1+xi).*(1+eta)/2,(1-xi).*(1+eta)/2,...
   (1-xi.^2).*(1-eta),(1+xi).*(1-eta.^2),...
   (1-xi.^2).*(1+eta),(1-xi).*(1-eta.^2),...
   2*(1-xi.^2).*(1-eta.^2)]/2;
phi_xi = [-(1-eta)/2,(1-eta)/2,(1+eta)/2,-(1+eta)/2,...
   -2*xi.*(1-eta),1-eta.^2,-2*xi.*(1+eta),-1+eta.^2,...
   -4*xi.*(1-eta.^2)]/2;
phi_eta = [-(1-xi)/2,-(1+xi)/2,(1+xi)/2,(1-xi)/2,...
   -1+xi.^2,-2*(1+xi).*eta,1-xi.^2,-2*(1-xi).*eta,...
   -4*(1-xi.^2).*eta]/2;
```

### 6.2. *Quadrature rules on $T_{\mathrm{ref}}$*

On $T_{\mathrm{ref}}$ we employ quadrature rules with one, three, or seven points.The points $(r_m, s_m)$, $m=1,\ldots,K_3$, for $K_3 = 1, 3, 7$ are shown in the plot of Fig. 16; their exact values can be found in the following code which stores the values $\psi_j(r_m, s_m)$, $\partial_r\psi_j(r_m, s_m)$, and $\partial_s\psi_j(r_m, s_m)$, for $j=1,\ldots,6$ and $m=1,\ldots,K_3$ in $K_3 \times 6$ arrays `psi`, `psi_r`, and `psi_s`, respectively.

```
function [psi,psi_r,psi_s,kappa] = quad3(K_3);
    switch K_3
    Case 1
          r = 1/3;
          s = 1/3;
          kappa = 1/2;
```

```
Case 3
  r=[1,4,1]'/6;
  s=[1,1,4]'/6;
  kappa=[1,1,1]/6;
otherwise
  pos=[6-sqrt(15),9+2*sqrt(15),6+sqrt(15),9-2*sqrt(15),7]/21;
  r=pos([1,2,1,3,3,4,5])';
 s=pos([1,1,2,4,3,3,5])';
 wts=[155-sqrt(15),155+sqrt(15),270]/2400;
 kappa=wts([1,1,1,2,2,2,3]);
end
one=ones(size(kappa,2),1);
psi=[1-r-s,r,s,4*r.*(1-r-s),4*r.*s,4*s.*(1-r-s)];
psi_r=[-one,one,0*one,4*(1-2*r-s),4*s,-4*s];
psi_s=[-one,0*one,one,-4*r,4*r,4*(1-r-2*s)];
```

### 6.3. Quadrature rules on $E_{\mathrm{ref}}$

On $E_{\mathrm{ref}}$ we use $K_N = 1$ with $t_1 = 0$ and $\delta_1 = 2$ or $K_N = 3$ with $t_1 = -\sqrt{3/5}$, $t_2 = 0$, $t_3 = \sqrt{3/5}$ and corresponding weights $\delta_1 = \delta_3 = 5/9$ and $\delta_2 = 8/9$. As above, we store the values of $\varphi_j(t_m, -1)$ and $\varphi_j'(t_m, -1)$ for $j = 1, 2, 5$ at the quadrature points $t_j$ in $K_N \times 3$ arrays `phi_E` and `phi_E_dt`, respectively. The weights are stored in the $1 \times K_N$ array `delta_E`.

```
function [phi_E,phi_E_dt,delta_E]=quadN(K_N);
  switch K_N
    Case 1
       t=0;
    delta_E=2;
  otherwise
    t=sqrt(3/5) * [-1,0,1]';
    delta_E=[5,8,5]/9;
  end
  one=ones(size(delta_E,2),1);
  phi_E=[1-t,1+t,2*(1-t).*(1+t)]/2;
  phi_E_dt=[-one,one,-4*t]/2;
```

**Acknowledgements**

The first author acknowledges support by the DFG through the priority program 1095 Analysis, Modeling and Simulation of Multiscale Problems. The paper was finished when the second author enjoyed hospitality by the Isaac Newton Institute for Mathematical Sciences, Cambridge, UK. The support by the EPSRC (N09176/01), FWF (P15274 and P16461), the DFG Multiscale Central Program is thankfully acknowledged. The third author acknowledges support by the DFG through the project "Platten und Schalen mit plastischer Verformung".

## Appendix A. The complete Matlab code

The following Matlab code implements the approximation scheme described in this article.

```
% Initialize
    load coordinates.dat;
    eval('load elements3.dat','elements3=[];');
    eval('load elements4.dat','elements4=[];');
    load Dirichlet.dat;
    eval('load Neumann.dat','Neumann=[];');
    A=sparse(size(coordinates,1),size(coordinates,1));
    b=zeros(size(coordinates,1),1); u=b; v=b;

% Local stiffness matrix and volume forces for elements with
three vertices
  [psi,psi_r,psi_s,kappa]=quad3(7);
  N=[1,1,0;0,1,1;1,0,1]/2;
  for j=1 : size(elements3,1)
  K_T=find(elements3(j,:));
  P=zeros(6,2);
  P(K_T,:)=coordinates(elements3(j,K_T),:);
  P(4:6,:)=P(4:6,:) + ((elements3(j,4:6)==0)' *
[1,1]) .* (N * P(1:3,:));
  D(1:3,:)=P(1:3,:);
  D(4:6,:)=P(4:6,:) - (N * P(1:3,:));
  M=zeros(6,6);
  for m=1 : size(kappa,2)
  D_Psi=[psi_r(m,:);psi_s(m,:)] * D;
  F=inv(D_Psi) * [psi_r(m,:);psi_s(m,:)];
  det_D_Psi(m)=abs(det(D_Psi));
  M=M + kappa(m) * (F' * F) * det_D_Psi(m);
end
A(elements3(j,K_T),elements3(j,K_T))=...
   A(elements3(j,K_T),elements3(j,K_T)) + M(K_T,K_T);
d=kappa .* det_D_Psi .* f(psi * D)' * psi;
b(elements3(j,K_T))=b(elements3(j,K_T)) + d(K_T)';
end

% Local stiffness matrix and volume forces for elements with
four vertices
  [phi,phi_xi,phi_eta,gamma]=quad4(9);
  K=[1,1,0,0;0,1,1,0;0,0,1,1;1,0,0,1]/2;
  L=[-1,-1,-1,-1,2,2,2,2]/4;
  for j=1 : size(elements4,1)
  J_T=find(elements4(j,:));
```

```
  P=zeros(9,2);
  P(J_T,:)=coordinates(elements4(j,J_T),:);
  P(5:8,:)=P(5:8,:) + ((elements4(j,5:8)==0)' * [1,1]) .*
(K * P(1:4,:));
  P(9,:)=P(9,:) + ((elements4(j,9)==0)' * [1,1]) .* (L *
P(1:8,:));
  C(1:4,:)=P(1:4,:);
  C(5:8,:)=P(5:8,:) - (K * P(1:4,:));
  C(9,:)=P(9,:) - (L * P(1:8,:));
  M=zeros(9,9);
  for m=1 : size(gamma,2)
   D_Phi=[phi_xi(m,:);phi_eta(m,:)] * C;
   F=inv(D_Phi) * [phi_xi(m,:);phi_eta(m,:)];
   det_D_Phi(m)=abs(det(D_Phi));
   M=M + gamma(m) * (F' * F) * det_D_Phi(m);
end
A(elements4(j,J_T),elements4(j,J_T))=...
   A(elements4(j,J_T),elements4(j,J_T)) + M(J_T,J_T);
d=gamma .* det_D_Phi .* f(phi * C)' * phi;
b(elements4(j,J_T))=b(elements4(j,J_T)) + d(J_T)';
end

% Neumann conditions
[phi_E,phi_E_dt,delta_E]=quadN(3);
for j=1 : size(Neumann,1)
J_E=find(Neumann(j,:));
P=zeros(3,2);
P(J_E,:)=coordinates(Neumann(j,J_E),:);
P(3,:)=P(3,:) + ((Neumann(j,3)==0)' * [1,1]) .* (P(1,:)
+ P(2,:))/2;
G(1:2,:)=P(1:2,:);
G(3,:)=P(3,:) - (P(1,:) + P(2,:))/2;
norm_Phi_E_dt=sqrt(sum((phi_E_dt * G)'.^2));
d=delta_E .* g(phi_E * G)' .* norm_Phi_E_dt * phi_E;
b(Neumann(j,J_E))=b(Neumann(j,J_E)) + d(J_E)';
end

% Dirichlet conditions
ind=find(Dirichlet(:,3));
u(unique(Dirichlet(:,1:2)))=u_D(coordinates(unique(Dirichlet
(:,1:2)),:));
u(Dirichlet(ind,3))=u_D(coordinates(Dirichlet(ind,3),:)) - ...
 (u(Dirichlet(ind,1)) + u(Dirichlet(ind,2)))/2;
b=b - A * u;
```

```
% Hanging nodes
      eval('load hanging_nodes.dat','hn=[];');
      if ~isempty(hn)
      M=[1,1,2,-2,0,0;1,1,3,-2,-4,0;1,1,3,-2,0,-4];
      B=sparse(3*size(hn,1), size(coordinates,1));
      for j=1:size(hn,1)
            B((1:3)+(j-1)*3,hn(j,:))=M;
      end
      lambdas=size(coordinates,1)+(1:3*size(hn,1));
      A=[A,B';B,sparse(3*size(hn,1), 3*size(hn,1))];
      b=[b;zeros(3*size(hn,1),1)];
      v=[v;zeros(3*size(hn,1),1)];
      else
      lambdas=[];
      end
% Compute solution in free nodes
freeNodes=[setdiff(1:size(coordinates,1),unique(Dirichlet)),
lambdas];
v(freeNodes)=A(freeNodes,freeNodes) b(freeNodes);
if ~isempty(hn)
v(size(coordinates,1)+1:end,:)=[];
end
% Display solution
submeshplot3(coordinates, elements3, v+u, granularity);
hold on
submeshplot4(coordinates, elements4, v+u, granularity);
drawgrid(coordinates, elements3, elements4, v+u, granularity);
hold off
```

## Appendix B. Implementation of right-hand sides

The following functions are examples for realizations of possible right-hand sides $u_D$, $g$, and $f$. They are stored in files `u_D.m`, `g.m`, and `f.m`, respectively.

```
function val=u_D(x);
  val=zeros(size(x,1),1);
function val=g(x);
  val=zeros(size(x,1),1);
function val=f(x);
  val=ones(size(x,1),1);
```

## Appendix C. Matlab routine to display the numerical solution

The following Matlab routines display the numerical solution. We only show the surface drawing function for quadrilaterals for the sake of brevity, the corresponding function for triangles

submesh-plot3.m is trivially similar.

```
function h=submeshplot4(coordinates, elements, u, granularity)
[Y,X]=meshgrid(-granularity:2:granularity,
-granularity:2:granularity);
sm_coords_ref=[X(:), Y(:)]/granularity;

% generate triangles on the reference quadrilateral
% as patch doesn't interpolate nicely, have P1 on the submesh
N=granularity + 1;
pnts=reshape(1:N*N, N, N);
pnts_ll=pnts(1:end-1, 1:end-1); %% lower left
pnts_lr=pnts(1:end-1, 2:end); %% lower right
pnts_ul=pnts(2:end, 1:end-1); %% upper left
pnts_ur=pnts(2:end, 2:end); %% upper right
sm_elems=[pnts_ll(:), pnts_ul(:), pnts_lr(:); ...
 pnts_ul(:), pnts_ur(:), pnts_lr(:)];

% generate the patches for each triangle and interpolate solution
vertices=[];
coords=[];
U=[];
inc=size(sm_coords_ref,1);
pm=1 - sm_coords_ref;
pp=1 + sm_coords_ref;
p2=1 - sm_coords_ref.^2;
psi=[pm(:,1).*pm(:,2), pp(:,1).*pm(:,2), ...
  pp(:,1).*pp(:,2), pm(:,1).*pp(:,2)]/4;
psi=[psi, [p2(:,1).*pm(:,2), p2(:,2).*pp(:,1), ...
 p2(:,1).*pp(:,2), p2(:,2).*pm(:,1)]/2, p2(:,1).*p2(:,2)];

% compute offsets on edges
     edgeOff=zeros(4,2,size(elements,1));
     ind1=find(elements(:,5:8));
     if ~isempty(ind1)
     [r,c]=ind2sub([size(elements,1),4],ind1);
     ind2=sub2ind([size(elements,1),4], r, rem(c,4)+1);
     indM=ind1 + 4*size(elements,1);
     tmp=coordinates(elements(indM),:) - ...
         (coordinates(elements(ind1),:) +
       coordinates(elements(ind2),:))/2;
   ind=sub2ind([4,size(elements,1)*2],c,2*r-1);
   edgeOff(ind)=tmp(:,1);
   edgeOff(ind+4)=tmp(:,2);
   end
```

```
% compute offsets on the center
  centerOff=zeros(size(elements,1),2);
  ind=find(elements(:,9));
  if ~isempty(ind)
  contEdge=reshape(sum(edgeOff(:,:,ind),1)/2, 2, length(ind))';
  linearmid=mean(reshape(coordinates(elements(ind,1:4),:), ...
   [length(ind),4,2]), 2);
centerOff(ind,:)=coordinates(elements(ind,9),:) - ...
   reshape(linearmid, length(ind), 2) - contEdge;
end

% assemble the submeshes
  for n=1:size(elements,1)
  vertices=[vertices; sm_elems + inc*(n-1)];
  K_T=find(elements(n,:));
  uloc=zeros(9,1);
  uloc(K_T)=u(elements(n,K_T));
  coords=[coords; psi*[coordinates(elements(n,1:4),:); ...
 edgeOff(:,:,n); centerOff(n,:)]];
U=[U; psi*uloc];
end

% plot
  col=mean(U(vertices),2);
  hh=trisurf(vertices, coords(:,1), coords(:,2), U, col,
'edgecolor','none');
  if nargout
  h=hh;
end
```

The routine to draw the mesh, `drawgrid.m`, plots the mesh for both the triangles and quadrilaterals.

```
function h=drawgrid(coordinates, elements3,
elements4, u, granularity)
 % create the edges
  if ~isempty(elements3)
  vert=reshape([elements3(:,1:3), elements3(:,[2,3,1])], ...
     3*size(elements3,1), 2);
  middle=reshape(elements3(:,4:6), 3*size(elements3,1), 1);
  else
  vert=[];
  middle=[];
  end
  if ~isempty(elements4)
  vert=[vert; reshape([elements4(:,1:4),
  elements4(:,[2,3,4,1])], ...
```

```
    4*size(elements4,1), 2)];
  middle=[middle; reshape(elements4(:,5:8),
  4*size(elements4,1), 1)];
  end
  [verts,I]=unique(sort(vert, 2),'rows');
  mids=middle(I);

% curved edges
I=find(mids);
if ~isempty(I)
offset=coordinates(mids(I),:) - ...
  (coordinates(verts(I,1),:) + coordinates(verts(I,2),:))/2;
l=(0:granularity)/granularity;
lx=coordinates(verts(I,1),1)*l + ...
  coordinates(verts(I,2),1)*(1-l) + offset(:,1)*((1-l).*l)*4;
ly=coordinates(verts(I,1),2)*l + ...
  coordinates(verts(I,2),2)*(1-l) + offset(:,2)*((1-l).*l)*4;
U=u(verts(I,1))*l + u(verts(I,2))*(1-l) + u(mids(I))*((1-l).*l)*4;
hh=plot3(lx', ly', U', 'k-');
else
hh=[];
end
hld=ishold;
hold on;

% linear edges
    I=find(~mids);
    if ~isempty(I)
    lx=reshape(coordinates(verts(I,:),1), length(I), 2);
    ly=reshape(coordinates(verts(I,:),2), length(I), 2);
    U=reshape(u(verts(I,:)), length(I), 2);
    hh=[hh; plot3(lx', ly', U', 'k-')];
    end
    if ~ishold
    hold off;
    end
    if nargout
    h=hh;
    end
```

## References

[1] J. Alberty, C. Carstensen, S.A. Funken, Remarks around 50 lines of Matlab: short finite element implementation, Numer. Algorithms 20 (1999) 117–137.

[2] J. Alberty, C. Carstensen, S.A. Funken, R. Klose, Matlab implementation of the finite element method in elasticity, Computing 69 (2002) 239–263.
[3] I. Babuska, T. Strouboulis, The Finite Element Method and Its Reliability, Oxford University Press, Oxford, 2001.
[4] K.-J. Bathe, Finite-Elemente-Methoden [Finite-element procedures in engineering analysis], Springer, Berlin, 1986.
[5] D. Braess, Finite Elemente, Springer, Berlin, 1991.
[6] S.C. Brenner, L.R. Scott, The Mathematical Theory of Finite Element Methods, Texts in Applied Mathematics, vol. 15, Springer, Berlin, 1994.
[7] C. Carstensen, R. Klose, Elastoviscoplastic finite element analysis in 100 lines of Matlab, J. Numer. Math. 10 (2002) 157–192.
[8] P.G. Ciarlet, The Finite Element Method for Elliptic Problems, North-Holland, Amsterdam, 1978.
[9] W.J. Gordon, Blending-function methods for bivariate and multivariate interpolation and approximation, SIAM J. Numer. Math. 8 (1971) 158–177.
[10] H.-R. Schwarz, Methode der Finiten Elemente, Teubner, Stuttgart, 1991.