Applied Mathematics and Computation 267 (2015) 252–263

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Fast MATLAB assembly of FEM matrices in 2D and 3D: Edge elements

I. Anjam [a,*], J. Valdman [b,c]

[a] Department of Mathematical Information Technology, University of Jyväskylä, Finland
[b] Institute of Mathematics and Biomathematics, University of South Bohemia, České Budějovice, Czech Republic
[c] Institute of Information Theory and Automation of the ASCR, Prague, Czech Republic

ARTICLE INFO

Keywords:
MATLAB code vectorization
Finite element method
Edge element
Raviart–Thomas element
Nédélec element

ABSTRACT

We propose an effective and flexible way to assemble finite element stiffness and mass matrices in MATLAB. We apply this for problems discretized by edge finite elements. Typical edge finite elements are Raviart–Thomas elements used in discretizations of $H(\mathrm{div})$ spaces and Nédélec elements in discretizations of $H(\mathrm{curl})$ spaces. We explain vectorization ideas and comment on a freely available MATLAB code which is fast and scalable with respect to time.

© 2015 Elsevier Inc. All rights reserved.

1. Introduction

Elliptic problems containing the full gradient operator ∇ of scalar or vector arguments are formulated in weak forms in H^1 Sobolev spaces and discretized using nodal finite element functions. Efficient MATLAB vectorization of the assembly routine of stiffness matrices for the linear nodal finite element was explained by Rahman and Valdman in [11]. The focus of this paper is generalizing the ideas of [11] to arbitrary finite elements, including higher order elements, and vector problems operating with the divergence operator div and the rotation operator curl. Such problems appear in electromagnetism and are also related to various mixed or dual problems in mechanics. Weak forms of these problems are defined in $H(\mathrm{div})$ and $H(\mathrm{curl})$ Sobolev spaces. A finite element discretization is done in terms of edge elements, typically Raviart–Thomas elements [12] for $H(\mathrm{div})$ problems and Nédélec elements [9] for $H(\mathrm{curl})$ problems. Edge element basis functions are not defined on the nodes of 2D triangular or 3D tetrahedral meshes, but on edges and faces. Edge elements provide only partial continuity over element boundaries: continuity of normal vector component for $H(\mathrm{div})$ problems and continuity of tangential vector component for $H(\mathrm{curl})$ problems.

The method of finite elements applied to $H(\mathrm{div})$ and $H(\mathrm{curl})$ problems and its implementation has been well documented, see for instance [16] including higher order polynomials defined through hierarchical bases. A user can find many software codes (for instance NGSOLVE [14] or HERMES [15]) written in object oriented languages allowing for higher order elements defined on elements with curved boundaries. These codes are very powerful, capable of high complexity computations and they provide certain flexibility via user interface. However, such codes are not so easy to understand and modify unless one is quite familiar with the code. We believe that our MATLAB code is more convenient for students and researchers who wish to become familiar with edge elements and prefer to have their own implementation. We consider the lowest order linear

* Corresponding author at: P.O. Box 35 (Agora), FI-40014 University of Jyväskylä, Finland.
E-mail addresses: immanuel.anjam@jyu.fi (I. Anjam), jvaldman@prf.jcu.cz (J. Valdman).

http://dx.doi.org/10.1016/j.amc.2015.03.105
0096-3003/© 2015 Elsevier Inc. All rights reserved.

edge elements defined on 2D triangles and 3D tetrahedra only. However, it is straightforward to extend the code to use higher order elements, since the assembly routines remain almost the same regardless of the element order.

There are plenty of papers [4,7,8,5] dedicated to implementing vectorized FEM assembly routines for nodal elements in MATLAB. In [8] the authors also discuss the Raviart–Thomas element in 3D, but do not provide the program code. The iFEM package [4] has efficient implementation of FEM assembly routines for various different linear and higher order elements. The paper [2] considers the implementation of Raviart–Thomas elements (in a non-vectorized way), providing a good inspiration for the implementation of a multigrid based solver for $H(\mathrm{div})$ majorant minimization [17] by the second author.

Our implementation generalizes the approach of [11] to work with arbitrary affine finite elements. It is based again on operations with long vectors and arrays in MATLAB and it is reasonably scalable for large size problems. On a typical computer with a decent processor and enough system memory, the 2D/3D assemblies of FEM matrices are very fast. For example, a 2D assembly of matrices with around 10 million rows takes less than a minute. Vectorization of calculations typically requires more system memory, but the performance degrades only when the system memory becomes full. The software described in this paper is available for download at MATLAB Central at

`http://www.mathworks.com/matlabcentral/fileexchange/46635`

It also includes an implementation of the linear nodal finite element in 2D and 3D with the generalized approach, so an interested reader can compare the performance of the code described in [11]. The key idea in our generalized approach is to vectorize the integration procedure of scalar and vector valued functions on affine meshes. This allows also for fast evaluation of norms of functions.

The paper is divided as follows: In Section 2 we briefly describe the implemented linear edge elements. In Section 3 we go through the particular constructions related to the implementation of the elements and vectorization details. We also show the performance of the vectorized assembly routines with respect to time and scalability. Section 4 illustrates two applications of edge elements: a functional majorant minimization in a posteriori error analysis and solving of an electromagnetic problem.

2. Linear edge elements

We denote by Ω an open, bounded, and connected Lipschitz domain in $\mathbb{R}^d$, where $d \in \{2,3\}$ denotes the space dimension. The divergence (2D and 3D) and rotation (3D) of a vector valued function $w : \Omega \to \mathbb{R}^d$ are defined as

$$\mathrm{div}\, w := \sum_{i=1}^{d} \partial_i w_i \quad \text{and} \quad \mathrm{curl}\, w := \begin{pmatrix} \partial_2 w_3 - \partial_3 w_2 \\ \partial_3 w_1 - \partial_1 w_3 \\ \partial_1 w_2 - \partial_2 w_1 \end{pmatrix}.$$

We consider two types of rotation operators in 2D, the vector operator $\underline{\mathrm{curl}}$ and the scalar operator curl

$$\underline{\mathrm{curl}} f := \begin{pmatrix} \partial_2 f \\ -\partial_1 f \end{pmatrix} \quad \text{and} \quad \mathrm{curl}\, w := \partial_1 w_2 - \partial_2 w_1$$

applied to a scalar function $f : \Omega \to \mathbb{R}$ and to a vector function $w : \Omega \to \mathbb{R}^2$. The operator $\underline{\mathrm{curl}}$ is frequently called the "co-gradient" in literature, and is often denoted by $\nabla^\perp$. The operators give rise to the standard Sobolev spaces:

$$H(\mathrm{div}, \Omega) := \{ v \in L^2(\Omega, \mathbb{R}^d) | \mathrm{div}\, v \in L^2(\Omega) \},$$
$$H(\mathrm{curl}, \Omega) := \begin{cases} \{ v \in L^2(\Omega, \mathbb{R}^3) \mid \mathrm{curl}\, v \in L^2(\Omega, \mathbb{R}^3) \} & \text{if } d = 3 \\ \{ v \in L^2(\Omega, \mathbb{R}^2) \mid \mathrm{curl}\, v \in L^2(\Omega) \} & \text{if } d = 2 \end{cases},$$

where L^2 denotes the space of square Lebesque integrable functions. We will denote the L^2-norm of scalar and vector valued functions by $\|\cdot\| := \|\cdot\|_{L^2(\Omega)}$. Assuming that Ω is discretized by a triangular (2D) or a tetrahedral (3D) mesh $\mathcal{T}$, Raviart–Thomas and Nédélec elements represent basis functions in $H(\mathrm{div}, \mathcal{T})$ and $H(\mathrm{curl}, \mathcal{T})$ spaces.

In the case of the lowest order (linear) Raviart–Thomas and Nédélec elements, there is one global degree of freedom (dof), i.e., one global basis function, related to either each edge (2D and 3D), or each face (3D) of a mesh $\mathcal{T}$. Due to construction, the global Raviart–Thomas basis functions and the Nédélec basis functions in 2D are nonzero only in the two elements who share the edge/face that is related to the basis function. In 3D the global Nédélec basis function is nonzero in all the elements sharing the related edge, and the number of these elements is usually more than two.

We denote the global edge/face basis functions by η^{RT} and η^{Ned}, and by $x = (x_1, x_2, x_3)^{\mathrm{T}}$ the spatial variable in Ω. The notation for reference basis functions and spatial variable is obtained by simply adding the hat $\hat{\cdot}$, i.e., $\hat{x}$ denotes the spatial variable in the reference element $\hat{K}$. We will use the unit triangle in 2D and the unit tetrahedron in 3D as the reference elements. We denote by $\hat{e}_i$ the ith edge of the reference triangle or tetrahedron, and by $\hat{f}_i$ the ith face of the reference tetrahedron. The numbering of the edges and faces, i.e., the numbering of the degrees of freedom in the reference elements, can be seen in Fig. 1. In the following, F_K denotes the affine element mapping $F_K(\hat{x}) := B_K \hat{x} + b_K$ from the reference element $\hat{K}$ to an element K in the mesh.

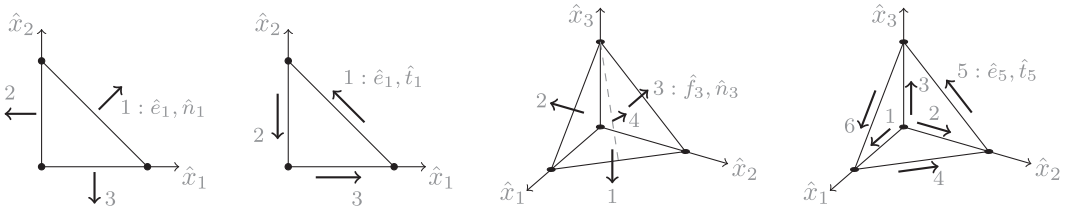

Fig. 1. Degrees of freedom of linear edge elements in the reference configuration $\hat{K}$.

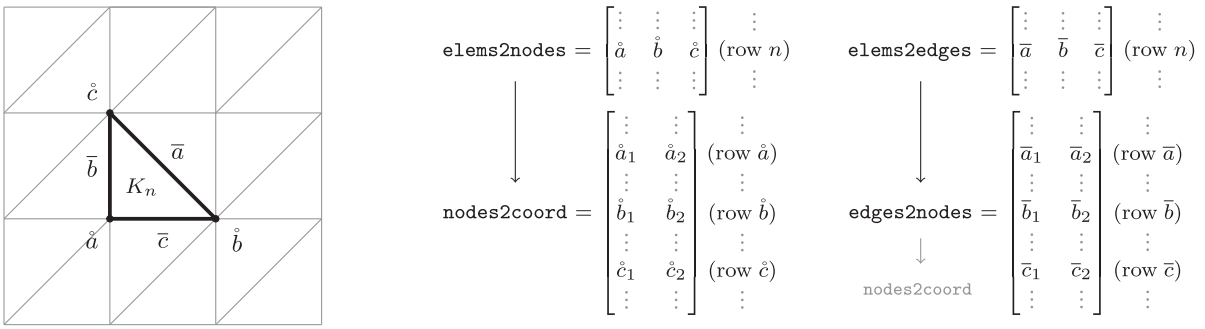

Fig. 2. Elements by their nodes and edges, i.e., global numbering of degrees of freedom for 2D linear finite elements.

A finite element is defined by the triplet $\{\hat{K},\hat{R},\hat{A}\}$, where $\hat{K}$ is the reference configuration, $\hat{R}$ the finite space of functions defined on the reference configuration, and $\hat{A}$ is the set of linearly independent degrees of freedom. The reference configurations we have already chosen. We also need a mapping which takes functions from $\hat{R}$ and maps them from $\hat{K}$ to an element K on the mesh $\mathcal{T}$. These mappings are called Piola mappings.

2.1. Raviart–Thomas element

The linear Raviart–Thomas element is based on the spaces (see, e.g., [9,12])

$$\mathbf{2D}:\ \hat{R}=\left\langle\begin{pmatrix}1\\0\end{pmatrix},\begin{pmatrix}0\\1\end{pmatrix},\begin{pmatrix}\hat{x}_1\\\hat{x}_2\end{pmatrix}\right\rangle,\quad \mathbf{3D}:\ \hat{R}=\left\langle\begin{pmatrix}1\\0\\0\end{pmatrix},\begin{pmatrix}0\\1\\0\end{pmatrix},\begin{pmatrix}0\\0\\1\end{pmatrix},\begin{pmatrix}\hat{x}_1\\\hat{x}_2\\\hat{x}_3\end{pmatrix}\right\rangle,$$

and the degrees of freedom for $\hat{u}\in\hat{R}$ read as

$$\mathbf{2D}:\ \hat{A}=\left\{\hat{\alpha}_i(\hat{u})=\int_{\hat{e}_i}\hat{n}_i\cdot\hat{u}\,\mathrm{d}\hat{s},\ i\in\{1,2,3\}\right\},\quad \mathbf{3D}:\ \hat{A}=\left\{\hat{\alpha}_i(\hat{u})=\int_{\hat{f}_i}\hat{n}_i\cdot\hat{u}\,\mathrm{d}\hat{s},\ i\in\{1,2,3,4\}\right\}\tag{1}$$

for every edge $\hat{e}_i$ in 2D, or face $\hat{f}_i$ in 3D, in the corresponding reference elements $\hat{K}$. There are three dofs in 2D and four in 3D. Here $\hat{n}_i$ is the normal unit vector of the edge $\hat{e}_i$, or the face $\hat{f}_i$. Here one has to choose which of the two possible unit normal vectors to use. The standard choice of outer unit normals is depicted in Fig. 1. The requirement $\hat{\alpha}_i(\hat{\eta}_j^{\mathrm{RT}})=\delta_{ij}$ (where δ_{ij} is the Kronecker delta) gives us the reference basis functions of the Raviart–Thomas element:

$$\mathbf{2D}:\ \hat{\eta}_1^{\mathrm{RT}}(\hat{x})=\begin{pmatrix}\hat{x}_1\\\hat{x}_2\end{pmatrix},\quad \hat{\eta}_2^{\mathrm{RT}}(\hat{x})=\begin{pmatrix}\hat{x}_1-1\\\hat{x}_2\end{pmatrix},\quad \hat{\eta}_3^{\mathrm{RT}}(\hat{x})=\begin{pmatrix}\hat{x}_1\\\hat{x}_2-1\end{pmatrix},$$

$$\mathbf{3D}:\ \hat{\eta}_1^{\mathrm{RT}}(\hat{x})=\begin{pmatrix}\hat{x}_1\\\hat{x}_2\\\hat{x}_3-1\end{pmatrix},\quad \hat{\eta}_2^{\mathrm{RT}}(\hat{x})=\begin{pmatrix}\hat{x}_1\\\hat{x}_2-1\\\hat{x}_3\end{pmatrix},\quad \hat{\eta}_3^{\mathrm{RT}}(\hat{x})=\begin{pmatrix}\hat{x}_1-1\\\hat{x}_2\\\hat{x}_3\end{pmatrix},\quad \hat{\eta}_4^{\mathrm{RT}}(\hat{x})=\begin{pmatrix}\hat{x}_1\\\hat{x}_2\\\hat{x}_3\end{pmatrix}.$$

In order to preserve normal continuity of the reference basis functions, we need to use the so-called Piola mappings. The values and the divergence values are mapped as follows (see, e.g., [3]):

$$\eta^{\mathrm{RT}}|_K(x) = \frac{1}{\det B_K} B_K\, \hat{\eta}^{\mathrm{RT}}(F_K^{-1}(x)) \quad \text{and} \quad \operatorname{div} \eta^{\mathrm{RT}}|_K(x) = \frac{1}{\det B_K} \operatorname{div} \hat{\eta}^{\mathrm{RT}}(F_K^{-1}(x)). \tag{2}$$

2.2. Nédélec element

The linear Nédélec element is based on the spaces (see, e.g., [9,13])

$$\mathbf{2D}: \ \hat{R} = \left\langle \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} \hat{x}_2 \\ -\hat{x}_1 \end{pmatrix} \right\rangle, \quad \mathbf{3D}: \ \hat{R} = \left\langle \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ \hat{x}_3 \\ \hat{x}_2 \end{pmatrix}, \begin{pmatrix} \hat{x}_3 \\ 0 \\ \hat{x}_1 \end{pmatrix}, \begin{pmatrix} \hat{x}_2 \\ \hat{x}_1 \\ 0 \end{pmatrix} \right\rangle,$$

and the degrees of freedom for $\hat{u} \in \hat{R}$ in both dimensions are related to the edges of the elements:

$$\hat{A} = \left\{ \hat{\alpha}_i(\hat{u}) = \int_{\hat{e}_i} \hat{t}_i \cdot \hat{u} \, \mathrm{d}\hat{s}, \ i \in \{1, 2, \ldots\} \right\} \tag{3}$$

for every edge $\hat{e}_i$ in the reference configuration $\hat{K}$. There are three dofs in 2D and six in 3D. Here $\hat{t}_i$ is the tangential unit vector of the edge $\hat{e}_i$. Similarly to the Raviart–Thomas element, one has to choose which direction for the unit tangential vectors to use. Our choice is depicted in Fig. 1. The requirement $\hat{\alpha}_i(\hat{\eta}_j^{\mathrm{Ned}}) = \delta_{ij}$ gives us the reference basis functions of the Nédélec element:

$$\mathbf{2D}: \ \hat{\eta}_1^{\mathrm{Ned}}(\hat{x}) = \begin{pmatrix} -\hat{x}_2 \\ \hat{x}_1 \end{pmatrix}, \quad \hat{\eta}_2^{\mathrm{Ned}}(\hat{x}) = \begin{pmatrix} -\hat{x}_2 \\ \hat{x}_1 - 1 \end{pmatrix}, \quad \hat{\eta}_3^{\mathrm{Ned}}(\hat{x}) = \begin{pmatrix} 1 - \hat{x}_2 \\ \hat{x}_1 \end{pmatrix},$$

$$\mathbf{3D}: \ \hat{\eta}_1^{\mathrm{Ned}}(\hat{x}) = \begin{pmatrix} 1 - \hat{x}_3 - \hat{x}_2 \\ \hat{x}_1 \\ \hat{x}_1 \end{pmatrix}, \quad \hat{\eta}_2^{\mathrm{Ned}}(\hat{x}) = \begin{pmatrix} \hat{x}_2 \\ 1 - \hat{x}_3 - \hat{x}_1 \\ \hat{x}_2 \end{pmatrix}, \quad \hat{\eta}_3^{\mathrm{Ned}}(\hat{x}) = \begin{pmatrix} \hat{x}_3 \\ \hat{x}_3 \\ 1 - \hat{x}_2 - \hat{x}_1 \end{pmatrix},$$

$$\hat{\eta}_4^{\mathrm{Ned}}(\hat{x}) = \begin{pmatrix} -\hat{x}_2 \\ \hat{x}_1 \\ 0 \end{pmatrix}, \quad \hat{\eta}_5^{\mathrm{Ned}}(\hat{x}) = \begin{pmatrix} 0 \\ -\hat{x}_3 \\ \hat{x}_2 \end{pmatrix}, \quad \hat{\eta}_6^{\mathrm{Ned}}(\hat{x}) = \begin{pmatrix} \hat{x}_3 \\ 0 \\ -\hat{x}_1 \end{pmatrix}.$$

Again, we need to use a Piola mapping in order to preserve the tangential continuity (see, e.g., [9,13]). The values are mapped as follows:

$$\eta^{\mathrm{Ned}}|_K(x) = B_K^{-\mathrm{T}}\, \hat{\eta}^{\mathrm{Ned}}\left(F_K^{-1}(x)\right). \tag{4}$$

The rotation is mapped differently depending on the dimension:

$$\mathbf{2D}: \ \operatorname{curl} \eta^{\mathrm{Ned}}|_K(x) = \frac{1}{\det B_K} \operatorname{curl} \hat{\eta}^{\mathrm{Ned}}\left(F_K^{-1}(x)\right), \tag{5}$$

$$\mathbf{3D}: \ \operatorname{curl} \eta^{\mathrm{Ned}}|_K(x) = \frac{1}{\det B_K} B_K \operatorname{curl} \hat{\eta}^{\mathrm{Ned}}\left(F_K^{-1}(x)\right). \tag{6}$$

2.3. Orientation of local degrees of freedom

In order to obtain the global basis functions η^{RT} and η^{Ned}, the transformations described in the previous sections are not enough. A global basis function is related to more than one element. No consideration has been yet made in making sure that the local orientation of the degrees of freedom (1) and (3) in these different elements is the same. The orientation must be same in order for the Raviart–Thomas and Nédélec elements to produce functions whose normal component, or tangential component (respectively) are continuous at element interfaces.

Take for example the Raviart–Thomas element in 2D. Let K_n and K_m be two elements in a mesh which share an edge (see Fig. 3), and let η^{RT} be the global basis function related to this edge. We denote by $\hat{\eta}_k^{\mathrm{RT}}$ and $\hat{\eta}_l^{\mathrm{RT}}$ the reference basis functions which we will transform from $\hat{K}$ to K_n and K_m respectively, in order to obtain the global basis function.

By taking a look at the dofs (1), we see that we are always using the outer unit normals to compute the local basis functions. If we simply use the transformation (2), the normal component of the values at the edge might be the opposite of each other. This depends wether or not the element mappings F_{K_n} and F_{K_m} preserve orientation. If $\det B_{K_n} > 0$, and $\det B_{K_m} < 0$, the element mapping F_{K_n} preserves the counter-clockwise orientation of the reference element, and F_{K_m} is oriented clockwise. This means that on the common edge the orientation is in the same direction, and the transformation (2) is enough for both elements. However, otherwise the orientation on the common edge will be the opposite, and one of the transformations must be multiplied by -1. The global basis function is thus obtained by

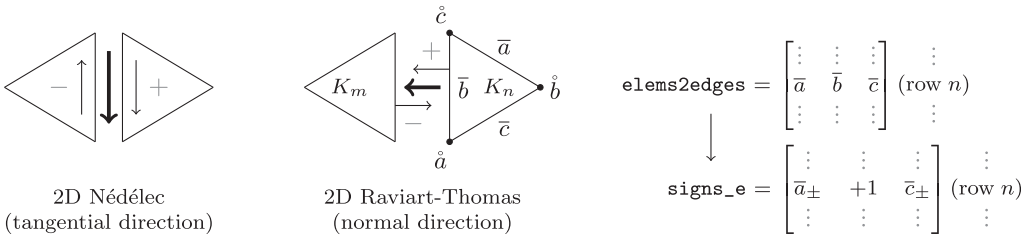

Fig. 3. Orientation of 2D edge elements sharing an edge, when both elements are oriented counter-clockwise, and $\mathring{c} > \mathring{a}$. The thick line denotes the "positive direction".

$$\eta^{\mathrm{RT}}|_{K_n}(x) = \left[\mathrm{sign}^k_{K_n}\right] \frac{1}{\det B_{K_n}} B_{K_n}\, \hat{\eta}^{\mathrm{RT}}_k\left(F^{-1}_{K_n}(x)\right) \quad \text{and} \quad \eta^{\mathrm{RT}}|_{K_m}(x) = \left[\mathrm{sign}^l_{K_m}\right] \frac{1}{\det B_{K_m}} B_{K_m}\, \hat{\eta}^{\mathrm{RT}}_l\left(F^{-1}_{K_m}(x)\right), \tag{7}$$

where

$$\begin{aligned}
&\left[\mathrm{sign}^k_{K_n}\right] = +1,\ \left[\mathrm{sign}^l_{K_m}\right] = +1 \quad \text{if} \quad \det B_{K_n} > 0,\ \det B_{K_m} < 0,\\
&\left[\mathrm{sign}^k_{K_n}\right] = +1,\ \left[\mathrm{sign}^l_{K_m}\right] = -1 \quad \text{if} \quad \det B_{K_n} > 0,\ \det B_{K_m} > 0,\\
&\left[\mathrm{sign}^k_{K_n}\right] = -1,\ \left[\mathrm{sign}^l_{K_m}\right] = +1 \quad \text{if} \quad \det B_{K_n} < 0,\ \det B_{K_m} < 0,\\
&\left[\mathrm{sign}^k_{K_n}\right] = -1,\ \left[\mathrm{sign}^l_{K_m}\right] = -1 \quad \text{if} \quad \det B_{K_n} < 0,\ \det B_{K_m} > 0.
\end{aligned}$$

We call these values the sign data related to each of the two elements. Note that the above means that the global basis function is obtained simply by

$$\eta^{\mathrm{RT}}|_{K_n}(x) = +\frac{1}{|\det B_{K_n}|} B_{K_n}\, \hat{\eta}^{\mathrm{RT}}_k\left(F^{-1}_{K_n}(x)\right) \quad \text{and} \quad \eta^{\mathrm{RT}}|_{K_m}(x) = -\frac{1}{|\det B_{K_m}|} B_{K_m}\, \hat{\eta}^{\mathrm{RT}}_l\left(F^{-1}_{K_m}(x)\right),$$

but we will use (7) since it is more convenient to implement in program code.

The situation is the same for 3D Raviart–Thomas element and the 2D Nédélec element. For the Nédélec element in 3D one needs to be more careful since a global basis function may be nonzero in a relatively large patch of elements, and the relevant orientation is related to an edge. Note also that the same sign data must be used when transforming the divergence or rotation of the basis functions.

2.4. Finite element matrices

We are interested in assembly of the mass matrices $\mathbf{M}^{\mathrm{RT}}$, $\mathbf{M}^{\mathrm{Ned}}$ and the stiffness matrices $\mathbf{K}^{\mathrm{RT}}$, $\mathbf{K}^{\mathrm{Ned}}$ defined by

$$\begin{aligned}
\mathbf{M}^{\mathrm{RT}}_{ij} &= \int_\Omega \eta^{\mathrm{RT}}_i \cdot \eta^{\mathrm{RT}}_j \,\mathrm{dx}, & \mathbf{M}^{\mathrm{Ned}}_{ij} &= \int_\Omega \eta^{\mathrm{Ned}}_i \cdot \eta^{\mathrm{Ned}}_j \,\mathrm{dx},\\
\mathbf{K}^{\mathrm{RT}}_{ij} &= \int_\Omega \mathrm{div}\, \eta^{\mathrm{RT}}_i\, \mathrm{div}\, \eta^{\mathrm{RT}}_j \,\mathrm{dx}, & \mathbf{K}^{\mathrm{Ned}}_{ij} &= \int_\Omega \mathrm{curl}\, \eta^{\mathrm{Ned}}_i \cdot \mathrm{curl}\, \eta^{\mathrm{Ned}}_j \,\mathrm{dx},
\end{aligned}$$

where the indexes i and j are the global numbering of the degrees of freedom, i.e., they are related to the edges or faces of a mesh. By using the Piola mappings (with correct orientations), we are able to assemble the local matrices using the reference element.

By using (7), the local matrices related to the global matrices $\mathbf{M}^{\mathrm{RT}}$ and $\mathbf{K}^{\mathrm{RT}}$ can be calculated on each element $K \in \mathcal{T}$ by

$$\begin{aligned}
\mathbf{M}^{\mathrm{RT},K}_{kl} &= \frac{1}{|\det B_K|} \int_{\hat{K}} \left[\mathrm{sign}^k_K\right] B_K \hat{\eta}^{\mathrm{RT}}_k(\hat{x}) \cdot \left[\mathrm{sign}^l_K\right] B_K \hat{\eta}^{\mathrm{RT}}_l(\hat{x}) \,\mathrm{d\hat{x}},\\
\mathbf{K}^{\mathrm{RT},K}_{kl} &= \frac{1}{|\det B_K|} \int_{\hat{K}} \left[\mathrm{sign}^k_K\right] \mathrm{div}\, \hat{\eta}^{\mathrm{RT}}_k(\hat{x}) \ \left[\mathrm{sign}^l_K\right] \mathrm{div}\, \hat{\eta}^{\mathrm{RT}}_l(\hat{x}) \,\mathrm{d\hat{x}},
\end{aligned} \tag{8}$$

where $\hat{K}$ is the reference element. The indexes k and l run through all the local basis functions in the element: $k, l \in \{1, 2, 3\}$ in 2D, and $k, l \in \{1, 2, 3, 4\}$ in 3D.

Similarly, by using (4) (and considering the correct orientations, see Section 2.3) the local mass matrices related to the global mass matrix $\mathbf{M}^{\mathrm{Ned}}$ can be calculated on each element $K \in \mathcal{T}$ by

$$\mathbf{M}^{\mathrm{Ned},K}_{kl} = |\det B_K| \int_{\hat{K}} \left[\mathrm{sign}^k_K\right] B_K^{-\mathrm{T}} \hat{\eta}^{\mathrm{Ned}}_k(\hat{x}) \cdot \left[\mathrm{sign}^l_K\right] B_K^{-\mathrm{T}} \hat{\eta}^{\mathrm{Ned}}_l(\hat{x}) \,\mathrm{d\hat{x}},$$

and by using (5) and (6) the local stiffness matrices related to the global stiffness matrix $\mathbf{K}^{\mathrm{Ned}}$ can be calculated by

$$\mathbf{2D}: \ \mathbf{K}_{kl}^{\mathrm{Ned},K} = \frac{1}{|\det B_K|}\int_{\hat{K}} \left[\mathrm{sign}_K^k\right] \mathrm{curl}\,\hat{\eta}_k^{\mathrm{Ned}}(\hat{x}) \ \left[\mathrm{sign}_K^l\right] \mathrm{curl}\,\hat{\eta}_l^{\mathrm{Ned}}(\hat{x})\,\mathrm{d}\hat{x},$$

$$\mathbf{3D}: \ \mathbf{K}_{kl}^{\mathrm{Ned},K} = \frac{1}{|\det B_K|}\int_{\hat{K}} \left[\mathrm{sign}_K^k\right] B_K \mathrm{curl}\,\hat{\eta}_k^{\mathrm{Ned}}(\hat{x}) \cdot \left[\mathrm{sign}_K^k\right] B_K \mathrm{curl}\,\hat{\eta}_l^{\mathrm{Ned}}(\hat{x})\,\mathrm{d}\hat{x}.$$

The indexes k and l run through all the local basis functions in the reference element: $k,l \in \{1,2,3\}$ in 2D, and $k,l \in \{1,2,3,4,5,6\}$ in 3D.

3. Implementation of edge elements

We denote by $\#\omega$ the number of elements in the set ω, and by $\mathcal{N},\mathcal{E},\mathcal{F}$, and $\mathcal{T}$ the sets of nodes, edges, faces, and elements, respectively. Note that faces $\mathcal{F}$ exist only in 3D. We need the following structures representing the mesh in order to implement edge elements. The second column states the size of the structure, and the third column the meaning of the structure.

`nodes2coord`	$\#\mathcal{N} \times 2/3$	nodes defined by their two/three coordinates in 2D/3D (in [11] `coordinates`)
`edges2nodes`	$\#\mathcal{E} \times 2$	edges defined by their two nodes in 2D/3D
`faces2nodes`	$\#\mathcal{F} \times 3$	faces defined by their three nodes in 3D

With these matrices available, we can then express every element by the list of its nodes, edges, or faces:

`elems2nodes`	$\#\mathcal{T} \times 3/4$	elements by their three/four nodes in 2D/3D (in [11] `elements`)
`elems2edges`	$\#\mathcal{T} \times 3/6$	elements by their three/six edges in 2D/3D
`elems2faces`	$\#\mathcal{T} \times 4$	elements by their four faces in 3D

In 2D both the linear Raviart–Thomas element and the linear Nédélec element have a degree of freedom related to each of the three edges of the reference triangle, totalling three dofs. In 3D the linear Nédélec element has a dof related to each of the six edges, and the Raviart–Thomas element will have a dof related to each of the four faces. Thus, the global numbering of degrees of freedom is given by the row indices of `edges2nodes` or `faces2nodes`. For a particular element K in the mesh, the global dofs related to it are then given by the structures `elems2edges` or `elems2faces`, respectively. For nodal elements the global numbering of dofs is given by the row indices of `nodes2coord`, and the dofs related to a particular element K are given by `elems2nodes`. In Fig. 2 we have further illustrated the structure of the mesh data in 2D.

Since the degrees of freedom are integrals over edges or faces, we need to pay attention to orientation (see Sections 2.3 and 2.4). In practice we need to know how every edge/face of every element is oriented. Orientation is naturally given either by $+1$ or -1. We need the following structures:

`signs_e`	$\#\mathcal{T} \times 3/6$	$+1$ or -1 for every edge of an element, corresponding to `elems2edges`
`signs_f`	$\#\mathcal{T} \times 4$	$+1$ or -1 for every face of an element in 3D, corresponding to `elems2faces`

In 2D obtaining the sign data for an element K_i can be conveniently done by examining `elems2nodes(i,:)`. The first edge of K_i (`elems2edges(i,1)`) is the edge from node 2 (`elems2nodes(i,2)`) to node 3 (`elems2nodes(i,3)`). We can then simply agree that if the global node indices satisfy `elems2nodes(i,2) > elems2nodes(i,3)`, we assign `signs_e(i,1) = 1`, and `signs_e(i,1) = -1` otherwise. This gives us the signs, or their opposites, as described in Section 2.3. This sign data can be used for both Raviart–Thomas element and the Nédélec element in 2D. The data structures are illustrated in Fig. 3.

The procedure of determining the signs for the 3D elements is straightforwards as well, but we will not comment on it here. In our software package the edges in 2D and 3D are calculated by the function `get_edges()` and the orientation related to edges is calculated by the function `signs_edges()`. In 3D the faces are calculated by the function `get_faces()`, and the orientation related to faces is calculated by `signs_faces()`.

3.1. Vectorized integration procedure

As stated in the introduction, the key idea of this paper is to vectorize the integration procedure of an arbitrary function on an arbitrary mesh. The main ingredient is how to efficiently use integration quadratures via the reference element. We demonstrate our idea by explaining how to calculate the (squared) L^2-norm $\|\mathtt{f}\|^2$ of a function $\mathtt{f} \in L^2(\Omega)$. Provided that we have the structures `nodes2coord` and `elems2nodes` available, this is achieved (in the folder `/example_majorant/`) with the following two lines:

```
[B_K, b_K, B_K_det] = affine_transformations(nodes2coord, elems2nodes);
f_L2norm = norm_L2(elems2nodes, B_K, b_K, B_K_det, f);
```

On the first line we obtain the affine transformations $F_K(\hat{x}) = B_K\hat{x} + b_K$ and determinants of B_K for all elements K:

`B_K`	$d \times d \times \#\mathcal{T}$	matrix parts B_K
`b_K`	$\#\mathcal{T} \times d$	vector parts b_K
`B_K_det`	$\#\mathcal{T} \times 1$	the determinants $\det B_K$

The calculation of this data is done in a vectorized manner. On the second line we calculate the norm. The code of the function `/example_majorant/norm_L2.m` is

```
    function fnorm = norm_L2( elems2nodes, B_K, b_K, B_K_det, f );
 1  B_K_detA       = abs(B_K_det);
 2  dim            = size(B_K,1);
 3  [ip,w,nip]     = intquad(6,dim);
 4  elems          = size(elems2nodes,1);
 5  fnorm          = zeros(nelems,1);
 6  for i = 1:nip
 7     F_K_ip      = squeeze(amsv(B_K, ip(i,:)))'+ b_K;
 8     fval        = f(F_K_ip);
 9     fnorm       = fnorm + w(i).* B_K_detA.* fval.^2;
10  end
11  fnorm = sum(fnorm);
```

On line 2 we deduce the dimension of the mesh. On line 3 the function `[ip,w,nip] = intquad(po,dim)` returns an integration quadrature of order `po` in the reference element. We use integration quadratures for triangles and tetrahedrons from [6,18], respectively. In this example we use quadrature order 6, so the calculation of the L^2-norm is exact (up to machine precision) for polynomials of order 3 and less. The quadrature consists of the integration points `ip` and the weighs `w`. The variable `nip` is the number of integration points. On the lines 4 and 5 we deduce the number of elements in the mesh in order to initialize the structure `fnorm`.

Note that the `for`-loop on line 6 is not over elements, but over integration points. This is what we mean by vectorization of the `for`-loop over elements. Essentially we are replacing this loop with another, much smaller loop. Of course, since all the affine mappings and other data has to be available for all elements at the same time, this method requires more system memory.

On line 7 we transform the `i`th integration point to the mesh for all elements K at the same time, and put this data into the structure `F_K_ip`. We have used here some functionality from the folder `/path/library_vectorization/`, which was also used in the vectorization of nodal elements in [11]. This folder contains functions which perform certain operations between matrices and vectors, and does them in a vectorized manner. The function `amsv.m` from this folder takes in the matrices `B_K` and does the necessary multiplication with the `i`th integration point `ip(i,:)` for all entries simultaneously. On line 8 we calculate the values of the function `f` on all of these points, and on line 9 we add the contributions of the `i`th integration point to `fnorm`.

After going through all the integration points, the structure `fnorm` contains the elementwise contributions of the norm, i.e., `fnorm(i)`$= \|\mathtt{f}\|^2_{K_i}$, where K_i is the element described by its nodes in `elems2nodes(i,:)`. On the last line the elementwise contributions are summed together to obtain $\|\mathtt{f}\|^2$.

3.2. Vectorized finite element assembly routine

The vectorized integration procedure of the previous section can be directly applied for finite element matrix assembly routines. As an example, we go through the needed program code for calculating the stiffness matrix $\mathbf{K}^{\mathrm{RT}}$ with Raviart–Thomas elements. We assume we have the mesh in the form of the structures `nodes2coord` and `elems2nodes`, i.e., we have the node coordinates, and the representation of elements by their nodes.

```
[B_K,~,B_K_det]                  = affine_transformations(nodes2coord,elems2nodes);
[elems2faces,faces2nodes]        = get_faces(elems2nodes);
signs_f                          = signs_faces(nodes2coord,elems2faces,faces2nodes,B_K);
```

On the first line we obtain the affine transformation matrices and the determinants. On the second line we obtain the structure `elems2faces`, which is the representation of elements by their faces. Note that indeed the numbers in `elems2faces` are indices to `faces2nodes`. More importantly, `elems2faces` is the global numbering of the degrees of freedom for all elements K in the mesh. On the third line we calculate the orientations for faces in 3D. Then, we call

```
K_RT0 = stiffness_matrix_RT0(elems2faces, B_K_det, signs_f);
M_RT0 = mass_matrix_RT0(elems2faces, B_K, B_K_det, signs_f);
```

to assemble the stiffness and mass matrices. The main part of the function `stiffness_matrix_RT0` is the vectorized assembly routine:

```
    function STIFF   = stiffness_matrix_RT0( elems, B_K_det, signs )
 1  dim              = size(elems,2)-1;
 2  nelems           = size(elems,1);
 3  B_K_detA         = abs(B_K_det);
 4  [ip,w,nip]       = intquad(1,dim);
 5  [~,dval,nbasis] = basis_RT0(ip);
 6  STIFF            = zeros(nbasis,nbasis,nelems);
 7  for i = 1:nip
 8      for m = 1:nbasis
 9          for k = m:nbasis
10              STIFF(m,k,:) = squeeze(STIFF(m,k,:)) + ...
11                             w(i).* B_K_detA.^(-1).*...
12                             ( signs(:,m).* dval(i,:,m) ).* ...
13                             ( signs(:,k).* dval(i,:,k) );
14          end
15      end
16  end
17  STIFF = copy_triu(STIFF);
18  ...
```

Note that this function does the assembly in both 2D and 3D, depending on the input variable `elems`.

On lines 1–4 we deduce the dimension of the problem, deduce the number of elements in the mesh, calculate the absolute values of the determinants, and obtain the first order integration quadrature on the reference element. This is enough since for the linear Raviart–Thomas element the basis function divergences are constants. The function `[val,dval,nbasis] = basis_RT0(ip)` returns the values `val` and divergence values `dval` of the linear Raviart–Thomas reference basis functions at the integration points. Since we are assembling the stiffness matrix, we need only the divergence values. The variable `nbasis` is the number of basis functions per element. On line 6, the variable `STIFF` is initialized to be of suitable size to contain all the local element matrices.

Note again that the outer `for`-loop on line 7 is not over elements, but over integration points. On lines 10–13 we assemble the local matrix entry `(m,k)` (for the integration point `i`) for all elements at the same time. The assembly is done according to (8). Note that since the matrix is symmetric, it is sufficient to assemble only the diagonal and upper triangular entries, hence the indexing on the loop in line 9 begins from the previous loop index `m`, and not 1. On line 17 the symmetric entries are copied to the lower triangular part of `STIFF`. After this, the global matrix is assembled from the local matrices in `STIFF`, but this part of the code we have excluded here.

This assembly routine consists only of the normal matrix operations of MATLAB. However, on most of the assembly routines we need to perform more complicated array operations. This functionality is provided by functions in `/path/library_vectorization/`.

3.3. Performance in 2D and 3D

For investigating the performance of our vectorized assembly routines, we chose an L-shaped domain in 2D, and the unit cube for 3D. The results were performed with MATLAB 7.13.0.564 (R2011b) on a computer with 64 Intel(R) Xeon(R) CPU E7–8837 processors running at 2.67 GHz, and 1 TB system memory. The computer is located at the University of Jyväskylä. Results can be seen in Tables 1 and 2.

Uniform refinement results in 4 times more triangles in 2D, and 8 times more tetrahedra in 3D. Thus, in each refinement step the optimal increase in time would be 4 in 2D and 8 in 3D. We see from Tables 1 and 2 that both 2D and 3D assembly routines scale with satisfactory performance as the problem size is increased. In 2D, on level 14 we already had over 2.4 billion elements, and the 1 TB system memory was still occupied by level 13 matrices. This forced the computer to start using swap memory, which considerably slowed the calculation of the new ~ 2.4 billion $\times$ 2.4 billion matrices for level 14.

Table 1
2D assembly times (in seconds) for an L-shaped domain $\Omega := (0,1)^2 \setminus (1/2,1)^2$. Values in brackets are the increase in time compared to the previous step (the optimal increase is 4).

Level	Size of matrices	Assembly of							
		$\mathbf{K}^{\mathrm{RT}}$		$\mathbf{M}^{\mathrm{RT}}$		$\mathbf{K}^{\mathrm{Ned}}$		$\mathbf{M}^{\mathrm{Ned}}$	
5	9 344	0.03	–	0.06	–	0.03	–	0.03	–
6	37 120	0.11	(3.6)	0.51	(8.5)	0.11	(3.6)	0.47	(15.6)
7	147 968	0.41	(3.7)	1.08	(2.1)	0.40	(3.6)	1.02	(2.1)
8	590 848	1.70	(4.1)	3.59	(3.3)	1.82	(4.5)	3.65	(3.5)
9	2 361 344	7.49	(4.4)	12.82	(3.5)	7.49	(4.1)	12.94	(3.5)
10	9 441 280	30.89	(4.1)	52.09	(4.0)	30.83	(4.1)	54.86	(4.2)
11	37 756 928	132.95	(4.3)	216.64	(4.1)	132.56	(4.2)	230.44	(4.2)
12	151 011 328	597.37	(4.4)	919.36	(4.2)	583.86	(4.4)	931.79	(4.0)
13	604 012 544	2620.11	(4.3)	3969.16	(4.3)	2840.51	(4.8)	4121.33	(4.4)
14	2 415 984 640	18333.25	(6.9)	33328.58	(8.3)	26781.41	(9.4)	37009.85	(8.9)

Table 2
3D assembly times (in seconds) for the unit cube $\Omega := (0,1)^3$. Values in brackets are the increase in time compared to the previous step (the optimal increase is 8).

Level	Size of matrices	Assembly of matrices				Size of matrices	Assembly of			
		$\mathbf{K}^{\mathrm{RT}}$		$\mathbf{M}^{\mathrm{RT}}$			$\mathbf{K}^{\mathrm{Ned}}$		$\mathbf{M}^{\mathrm{Ned}}$	
1	2 808	0.02	–	0.09	–	1 854	0.05	–	0.09	–
2	21 600	0.14	(7.0)	0.39	(4.3)	13 428	0.30	(6.0)	0.79	(8.7)
3	169 344	0.82	(5.8)	2.18	(5.5)	102 024	1.92	(6.4)	4.53	(5.7)
4	1 340 928	7.15	(8.7)	15.35	(7.0)	795 024	15.44	(8.0)	33.43	(7.3)
5	10 672 128	59.37	(8.3)	125.71	(8.1)	6 276 384	129.91	(8.4)	282.14	(8.4)
6	85 155 840	503.89	(8.4)	1054.49	(8.3)	49 877 568	1125.08	(8.6)	2291.50	(8.1)
7	680 361 984	4437.84	(8.8)	8717.70	(8.2)	397 689 984	10232.01	(9.0)	20028.06	(8.7)

It is also notable that in 3D the calculation of the matrices $\mathbf{K}^{\mathrm{Ned}}$ and $\mathbf{M}^{\mathrm{Ned}}$ for the Nédélec element takes over twice the time compared to the calculation of $\mathbf{K}^{\mathrm{RT}}$ and $\mathbf{M}^{\mathrm{RT}}$ even though there are more degrees of freedom for the Raviart–Thomas matrices. The reason becomes evident when comparing the amount of algebraic operations that need to be calculated: for example, the divergences of Raviart–Thomas basis functions in 3D are scalar valued, but the rotations of Nédélec basis functions in 3D are vector valued.

4. Examples of vectorized FEM computations using edge elements

4.1. Minimization of functional majorant using Raviart–Thomas elements

Let us consider a scalar boundary value (Poisson's) problem

$$\begin{aligned} -\triangle u &= f \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned}$$

for a function $u \in \mathring{H}^1(\Omega) := \{v \in L^2(\Omega) | \nabla v \in L^2(\Omega, \mathbb{R}^d),\ v = 0 \text{ on } \partial\Omega\}$ and a given right hand side $f \in L^2(\Omega)$. The exact solution u is sought from the weak formulation

$$\int_\Omega \nabla u \cdot \nabla w \,\mathrm{d}x = \int_\Omega f w \,\mathrm{d}x \quad \forall w \in \mathring{H}^1(\Omega). \tag{9}$$

Assume that $v \in \mathring{H}^1(\Omega)$ is an approximation of the exact solution u of (9). Then, the functional type a posteriori error estimate from [10] states that

$$\|\nabla(u - v)\| \leqslant \|\nabla v - y\| + C_F \|\mathrm{div} y + f\| =: M(\nabla v, f, C_F, y), \quad \forall y \in H(\mathrm{div}, \Omega), \tag{10}$$

where M is called a functional majorant. The global constant C_F represents the smallest possible constant from the Friedrichs' inequality $\|w\| \leqslant C_F \|\nabla w\|$ which holds for all $w \in \mathring{H}^1(\Omega)$. Note that the estimate (10) is sharp: by choosing $y = \nabla u$, the inequality changes into an equality. By this we immediately see that minimizing M with respect to y provides us a way to obtain approximations of the flux ∇u. Since M contains nondifferentiable norm terms, we apply the Young's inequality $(a+b)^2 \leqslant (1+\beta)a^2 + (1+\frac{1}{\beta})b^2$ valid for all $\beta > 0$ to obtain

$$(\|\nabla v - y\| + C_F\|\mathrm{div} y + f\|)^2 \leqslant \left(1+\frac{1}{\beta}\right)\|\nabla v - y\|^2 + (1+\beta)C_F^2\|\mathrm{div} y + f\|^2 =: \mathcal{M}(\nabla v, f, C_F, \beta, y).$$

The majorant $\mathcal{M}$ arguments v and f are known, and upper bounds of C_F are also known. The parameter $\beta > 0$ and the function $y \in H(\mathrm{div}, \Omega)$ are free parameters. For a fixed value of β, the majorant represents a quadratic functional in y. Global minimization of $\mathcal{M}$ with respect to y results in the following problem for y:

$$(1+\beta)C_F^2 \int_\Omega \mathrm{div}\, y\, \mathrm{div}\, \phi\, \mathrm{d}x + \left(1+\frac{1}{\beta}\right)\int_\Omega y\cdot\phi\, \mathrm{d}x = -(1+\beta)C_F^2\int_\Omega f\, \mathrm{div}\,\phi\, \mathrm{d}x + \left(1+\frac{1}{\beta}\right)\int_\Omega \nabla v\cdot\phi\, \mathrm{d}x \quad \forall \phi \in H(\mathrm{div}, \Omega) \tag{11}$$

On the other hand, for a fixed y,

$$\beta = \frac{\|\nabla v - y\|}{C_F\|\mathrm{div} y + f\|} \tag{12}$$

minimizes $\mathcal{M}$ amongst all $\beta > 0$. It suggests the following solution algorithm:

Algorithm 1 (*Majorant minimization algorithm*). Let $\beta > 0$ be given (for example, set $\beta = 1$).

(a) Compute y (using current value of β) by minimizing the quadratic problem $\mathcal{M}(\nabla v, f, C_F, \beta, y) \to \min$.
(b) Update β (using y calculated in step (a)) from (12). If the convergence in y is not achieved then go to step (a).

We solved the quadratic minimization problem in (a) by discretizing the problem (11) with the linear Raviart–Thomas elements. For this both of the FEM matrices $\mathbf{M}^{\mathrm{RT}}$ and $\mathbf{K}^{\mathrm{RT}}$ were needed (see also [17]).

Example 1. In 2D we choose the unit square $\Omega := (0,1)^2$, and in 3D the unit cube $\Omega := (0,1)^3$. We choose the bubble function

$$u(x) := \prod_{i=1}^{d} x_i(x_i - 1)$$

as the exact solution in both 2D and 3D. It is clear that $u \in H^1(\Omega)$ in both dimensions.

On Tables 3 and 4 we have calculated the majorant $\mathcal{M}$ values with Algorithm 1 for four different meshes in 2D and 3D, respectively. The program code can be found in the folder `/example_majorant/`. The approximation v was calculated with linear nodal finite elements. For measuring the quality of the chosen free parameters β and y, we have also included the values of the so-called efficiency index $I_{\mathrm{eff}} := \sqrt{\mathcal{M}}/\|\nabla(u - v)\| \geqslant 1$. The approximation v and flux approximation y of the smallest 2D mesh are depicted in Fig. 4. The iterations of Algorithm 1 were stopped if the distance of the previous value of the majorant to the new value (normalized with the previous value) was less than 10^{-4}.

4.2. Solving the eddy-current problem using Nédélec elements

We split the boundary into two parts: $\partial\Omega := \Gamma_D \cup \Gamma_N$ such that $\Gamma_D \cap \Gamma_N = \emptyset$. The 2D eddy-current problem reads as

Table 3
Majorant calculation for four meshes in 2D.

Iter	$\#\mathcal{T} = 512$			$\#\mathcal{T} = 131\,072$			$\#\mathcal{T} = 2\,097\,152$			$\#\mathcal{T} = 33\,554\,432$		
	β	$\mathcal{M}$	I_{eff}	β	$\mathcal{M}$	I_{eff}	β	$\mathcal{M}$	I_{eff}	β	$\mathcal{M}$	I_{eff}
1	1.000	0.026203	1.72	1.000	0.001648	1.73	1.000	0.000412	1.73	1.000	0.000103	1.73
2	3.208	0.023159	1.52	3.294	0.001453	1.52	3.294	0.000363	1.52	3.294	0.000091	1.52
3	3.268	0.023159	1.52	3.294	0.001453	1.52	3.294	0.000363	1.52	3.294	0.000091	1.52

Table 4
Majorant calculation for four meshes in 3D.

Iter	$\#\mathcal{T} = 10\,368$			$\#\mathcal{T} = 82\,944$			$\#\mathcal{T} = 663\,552$			$\#\mathcal{T} = 5\,308\,416$		
	β	$\mathcal{M}$	I_{eff}	β	$\mathcal{M}$	I_{eff}	β	$\mathcal{M}$	I_{eff}	β	$\mathcal{M}$	I_{eff}
1	1.000	0.011794	1.59	1.000	0.006176	1.59	1.000	0.003135	1.59	1.000	0.001574	1.59
2	3.128	0.010396	1.40	3.512	0.005379	1.39	3.655	0.002721	1.38	3.697	0.001365	1.38
3	3.420	0.010388	1.40	3.622	0.005379	1.39	3.687	0.002720	1.38	3.706	0.001365	1.38
4	3.432	0.010388	1.40									

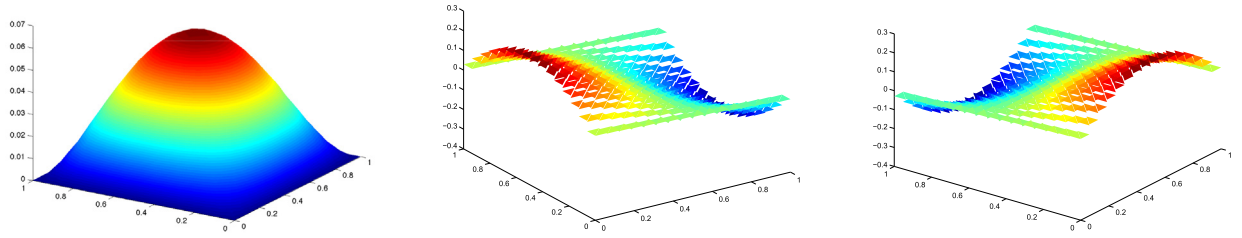

Fig. 4. Discrete solution v (left), and the flux approximation y first and second components (middle and right) on a mesh with 512 elements.

$$\begin{aligned} \underline{\mathrm{curl}}\,\mu^{-1}\mathrm{curl}\,E + \kappa E &= F \text{ in } \Omega, \\ E \times n &= 0 \text{ in } \Gamma_D, \\ \mu^{-1}\mathrm{curl}\,E &= 0 \text{ in } \Gamma_N, \end{aligned}$$

for $E \in H_{\Gamma_D}(\mathrm{curl}, \Omega) := \{v \in H(\mathrm{curl}, \Omega) | v \times n = 0 \text{ on } \Gamma_D\}$, where n denotes the outward unit normal to the boundary $\partial\Omega$. Here the right hand side $F \in L^2(\Omega, \mathbb{R}^2)$, and the positive material parameters $\mu, \kappa \in L^\infty(\Omega)$ are given. The exact solution E is sought from the weak formulation

$$\int_\Omega \mu^{-1}\mathrm{curl}\,E\,\mathrm{curl}\,w\,\mathrm{d}x + \int_\Omega \kappa E \cdot w\,\mathrm{d}x = \int_\Omega F \cdot w\,\mathrm{d}x \quad \forall w \in H_{\Gamma_D}(\mathrm{curl}, \Omega). \tag{13}$$

Example 2 [1]. We choose the unit square $\Omega := (0,1)^2$ with $\kappa = \mu = 1$. We split the domain in two parts across the diagonal, $\Omega_1 := \{x \in \Omega | x_1 > x_2\}$ and $\Omega_2 := \Omega \setminus \overline{\Omega}_1$, in order to define the following discontinuous exact solution:

$$E|_{\Omega_1}(x) := \begin{pmatrix} \sin(2\pi x_1) + 2\pi\cos(2\pi x_1)(x_1 - x_2) \\ \sin\left((x_1 - x_2)^2(x_1 - 1)^2 x_2\right) - \sin(2\pi x_1) \end{pmatrix}, \quad E|_{\Omega_2}(x) := 0.$$

Since on $\Gamma_/ := \{x \in \Omega | x_1 = x_2\}$ we have

$$E|_{\Omega_1} \times n = \frac{1}{\sqrt{2}}\left(2\pi\cos(2\pi x_1)(x_1 - x_2) + \sin\left((x_1 - x_2)^2(x_1 - 1)^2 x_2\right)\right), \quad E|_{\Omega_2} \times n = 0,$$

we see that $E|_{\Omega_1} \times n = 0$ on $\Gamma_/$. We conclude that the tangential component is continuous on $\Gamma_/$, so E belongs to $H(\mathrm{curl}, \Omega)$. Moreover,

$$\mathrm{curl}\,E|_{\Omega_1} = 2x_2(x_1 - x_2)(x_1 - 1)(2x_1 - x_2 - 1)\cos\left(x_2(x_1 - x_2)^2(x_1 - 1)^2\right), \quad \mathrm{curl}\,E|_{\Omega_2} = 0,$$

and clearly $\mathrm{curl}\,E|_{\Omega_1} = 0$ on $\Gamma_/$, i.e., $\mathrm{curl}\,E$ is continuous on $\Gamma_/$. Also, it is easy to see that $\mathrm{curl}\,E$ vanishes on the whole boundary, so it belongs to $H^1(\Omega)$. Thus, the exact solution satisfies zero Neumann boundary condition on the whole boundary, i.e., $\Gamma_D = \emptyset$ and $\Gamma_N = \partial\Omega$.

We denote by v an approximation of the exact solution E of (13). In the discretization of (13) we need both the mass and stiffness matrices $\mathbf{M}^{\mathrm{Ned}}$ and $\mathbf{K}^{\mathrm{Ned}}$. We see from Fig. 5 that the 2D Nédélec element catches the normal discontinuity on the diagonal line $\Gamma_/$. In Table 5 we show how the error measured in the $H(\mathrm{curl}, \Omega)$-norm decreases as the mesh is uniformly refined. The program code can be found in the folder `/example_eddycurrent/`.

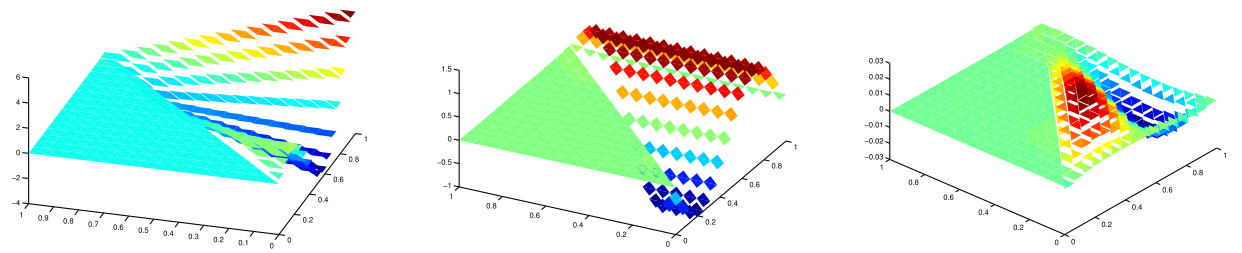

Fig. 5. Discrete solution v first and second components (left and middle), and curl v (right) on a mesh with 512 elements.

Table 5
Exact energy errors of approximations of the 2D eddy-current problem on uniformly refined meshes.

$\#\mathcal{T}$	$\#\mathcal{E}$	$\sqrt{\|E-v\|^2+\|\mathrm{curl}(E-v)\|^2}$
32 768	49 408	2.358185e−02
131 072	197 120	1.179151e−02
524 288	787 456	5.895834e−03
2 097 152	3 147 776	2.947927e−03
8 388 608	12 587 008	1.473965e−03
33 554 432	50 339 840	7.369826e−04

Acknowledgments

The work of the first author was supported by the Väisälä Foundation of the Finnish Academy of Science and Letters. The second author acknowledges the support of the project GA13-18652S (GA CR).

References

[1] I. Anjam, D. Pauly, Functional a posteriori error equalities for conforming mixed approximations of elliptic problems. Preprint, 2014. <http://arxiv.org/abs/1403.2560> (accessed 9.1.2015)
[2] C. Bahriawati, C. Carstensen, Three MATLAB implementations of the lowest-order Raviart–Thomas MFEM with a posteriori error control, CMAM 5 (4) (2005) 333–361.
[3] F. Brezzi, M. Fortin, Mixed and hybrid finite element methods, Springer Series in Computational Mathematics, 15, Springer-Verlag, New York, 1991.
[4] L. Chen, iFEM: an integrated finite element method package in MATLAB. Technical report, University of California at Irvine, 2009. <http://www.math.uci.edu/~chenlong/programming.html> (accessed 9.1.2015)
[5] F. Cuvelier, C. Japhet, G. Scarella, An efficient way to perform the assembly of finite element matrices in Matlab and Octave. Preprint, 2013. <http://arxiv.org/abs/1305.3122> (accessed 9.1.2015)
[6] D.A. Dunavant, High degree efficient symmetrical gaussian quadrature rules for the triangle, Int. J. Numer. Methods Eng. 21 (1985) 1129–1148.
[7] S. Funken, D. Praetorius, P. Wissgott, Efficient implementation of adaptive P1-FEM in MATLAB, Comput. Methods Appl. Math. 11 (2011) 460–490.
[8] A. Hannukainen, M. Juntunen, Implementing the finite element assembly in interpreted languages. Preprint, Aalto University, 2012.
[9] J.C. Nédélec, Mixed finite elements in $\mathbb{R}^3$, Numer. Mat. 35 (1980) 315–341.
[10] P. Neittaanmäki, S. Repin, Reliable methods for computer simulation, Error control and a posteriori estimates, Studies in Mathematics and its Applications, 33, Elsevier, Amsterdam, 2004.
[11] T. Rahman, J. Valdman, Fast MATLAB assembly of FEM matrices in 2D and 3D: Nodal elements, Appl. Math. Comput. 219 (2013) 7151–7158.
[12] P.A. Raviart, J.M. Thomas, A mixed finite element for second order elliptic problems, in: I. Galligani, E. Magenes (Eds.), Mathematical Aspects of Finite Element Methods, Springer-Verlag, New York, 1977, pp. 292–315.
[13] A. Schneebeli, An $H(\mathrm{curl};\Omega)$-conforming FEM: Nédélec's elements of first type. Technical report, 2003. <http://www.dealii.org/reports/nedelec/nedelec.pdf> (accessed 9.1.2015)
[14] J. Shöberl, C++11 implementation of finite elements in NGSolve. ASC Report 30/2014, Institute for Analysis and Scientific Computing, Vienna University of Technology, 2014.
[15] P. Šolín, L. Korous, P. Kus, Hermes2D, a C++ library for rapid development of adaptive hp-FEM and hp-DG solvers, J. Comput. Appl. Math. 270 (2014) 152–165.
[16] P. Šolín, K. Segeth, I. Doležel, Higher-order finite element methods, Studies in Advanced Mathematics, 41, Chapman and Hall, CRC, Boca Raton, Florida, 2003.
[17] J. Valdman, Minimization of functional majorant in a posteriori error analysis based on H(div) multigrid-preconditioned cg method, Adv. Numer. Anal. 2009 (2009). Article ID 164519.
[18] L. Zhang, T. Cui, H. Liu, A set of symmetric quadrature rules on triangles and tetrahedra, J. Comput. Math. 26 (3) (2008) 1–16.