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

I. ANJAM$^1$, J. VALDMAN$^{2,3}$

$^1$Department of Mathematical Information Technology, University of Jyväskylä, Finland
$^2$Institute of Mathematics and Biomathematics, University of South Bohemia, České Budějovice, Czech Republic
$^3$Institute of Information Theory and Automation of the ASCR, Prague, Czech Republic

Abstract: Based on the ideas of the paper [8] by Talal Rahman and Jan Valdman we propose an effective and flexible way to assemble finite element stiffness and mass matrices in MATLAB for problems discretized by edge finite elements. Typical edge finite elements are Raviart-Thomas elements used in discretizations of $H(\text{div})$ spaces and Nédélec elements in discretizations of $H(\text{curl})$ spaces. The major loops in the code have been vectorized using the so called array operation in MATLAB, and no low level languages like C or Fortran have been used for the purpose. The implementation is based on having the vectorization part separated, in other words hidden, from the original code thereby preserving its original structure, and its flexibility as a finite element code. We explain vectorization ideas and comment on a freely available MATLAB code which is fast and scalable with respect to time.

Keywords: MATLAB code vectorization, Finite element, Edge element, Raviart-Thomas element, Nédélec element

1 Introduction

Elliptic problems containing the full gradient operator $\nabla$ 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 nodal finite element functions and a resulting fast assembly of stiffness matrices was explained in [8]. The focus of this paper is the extension of ideas of [8] to vector problems operating with divergence operator div and rotation operator curl. Such problems appear in electromagnetism and also are related to various mixed or dual problems in mechanics. Weak forms of these problems are defined in $H(\text{div})$ and $H(\text{curl})$ Sobolev spaces. A finite element discretization is done in terms of edge elements, typically Raviart-Thomas elements for $H(\text{div})$ problems [9] and Nédélec elements [6] for $H(\text{curl})$ problems. Edge element basis functions are not defined on the nodes of 2D triangular 3D tetrahedral meshes, but on edges and faces. Edge elements provide only partial continuity over element boundaries: continuity of normal vector component for $H(\text{div})$ problems and continuity of tangential vector component for $H(\text{curl})$ problems.

The method of finite elements applied to $H(\text{div})$ and $H(\text{curl})$ problems and its implementation has been well documented, see for instance [11] including high order polynomials defined through hierarchical bases. A user can find many software codes (for instance NGSOLVE by Joachim Schöberl or HERMES by Pavel Šolín) written in object oriented languages allowing for higher order elements defined on elements with curved boundaries. Such codes are very powerful, capable of high complexity computations and provide certain flexibility via user interface. However, if some features are not available, it is usually difficult to understand the source code and modify it. 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 edge elements defined on 2D triangles and 3D tetrahedra only, and hope that a careful reader will be able to modify the code for higher order elements as well. Our vectorization is based on operations with long vectors and arrays in MATLAB. This technique is therefore convenient for implementation also in other programming languages (especially interpreted languages).

There is plenty of literature dedicated to construction of nodal elements in MATLAB, but only a few related to edge elements. The paper [2] provided a good inspiration for the implementation of a multigrid based solver for $H(\text{div})$ majorant minimization [12] by the second author. Our implementation here utilizes vectorization techniques and it is therefore significantly faster in its performance. Even without any parallelization, is allows for assemblies of matrices with around billions of rows on a computer with sufficient operational system memory. The MATLAB software is available for free download and testing at MATLAB Central at http://www.mathworks.com/matlabcentral/fileexchange/46635

The code is built on top of MATLAB codes related to the vectorization of nodal elements described in [8].

In [13] the authors implemented a vectorized code for 2D continuous elements, and compare its performance to the codes of [4, 5, 8]. In [5] the authors implemented linear and higher order continuous elements in 2D and 3D, and also the linear Raviart-Thomas element in 3D. However, the authors provided the code only for the linear continuous

*corresponding author, email: immanuel.anjam@jyu.fi
element. The iFEM package [4] has efficient implementation of FEM assembly routines for various different linear and higher order elements. However, due to page limitations, we are unable to make comparisons to these codes.

The paper is divided as follows: Section 2 shortly describes the implemented linear edge elements without going into details. In Section 3 we go through the particular constructions related to the implementation of these elements. We also show the time performance and scalability of the vectorized assembly routines. Section 4 illustrates two applications of edge elements: a functional majorant minimization in a posteriori analysis and solving of an electromagnetic problem.

2 Linear edge elements

We denote by \( \Omega \) 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

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

We consider two types of rotation operators in 2D, the vector operator \( \text{curl} \) and the scalar operator \( \text{curl} \)

\[
\text{curl } f := \begin{pmatrix}
\partial_2 f \\
-\partial_1 f
\end{pmatrix} \quad \text{and} \quad \text{curl } w := \partial_1 w_2 - \partial_2 w_1.
\]

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

\[
H(\text{div}, \Omega) := \{ v \in L^2(\Omega, \mathbb{R}^d) \mid \text{div } v \in L^2(\Omega) \},
\]

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

where \( L^2 \) denotes the space of square Lebesgue integrable functions. Assuming that \( \Omega \) is discretized by a triangular (in 2D) or a tetrahedral (in 3D) mesh \( \mathcal{T} \), Raviart-Thomas and Nédélec elements represent basis functions in \( H(\text{div}, \mathcal{T}) \) and \( H(\text{curl}, \mathcal{T}) \) spaces. The numbering of the degrees of freedom of the lowest order (linear) elements can be seen in Figure 1. We denote the global edge basis functions by \( \eta^{\text{RTO}} \) and \( \eta^{\text{Ned}} \), and by \( x = (x_1, x_2, x_3)^T \) the spatial variable in \( \Omega \). Similarly, we define the reference basis functions and spatial variable simply by adding the hat \( \hat{\cdot} \), i.e., \( \hat{x} \) denotes the spatial variable in the reference element. The reference basis functions of the Raviart-Thomas element are as follows (see, e.g., [6, 9]):

2D:

\[
\eta_1^{\text{RTO}}(\hat{x}) = \begin{pmatrix}
\hat{x}_1 \\
\hat{x}_2
\end{pmatrix},
\eta_2^{\text{RTO}}(\hat{x}) = \begin{pmatrix}
\hat{x}_1 - 1 \\
\hat{x}_2
\end{pmatrix},
\eta_3^{\text{RTO}}(\hat{x}) = \begin{pmatrix}
\hat{x}_1 \\
\hat{x}_2 - 1
\end{pmatrix},
\]

3D:

\[
\eta_1^{\text{RTO}}(\hat{x}) = \begin{pmatrix}
\hat{x}_1 \\
\hat{x}_2 \\
\hat{x}_3 - 1
\end{pmatrix},
\eta_2^{\text{RTO}}(\hat{x}) = \begin{pmatrix}
\hat{x}_1 - 1 \\
\hat{x}_2 \\
\hat{x}_3
\end{pmatrix},
\eta_3^{\text{RTO}}(\hat{x}) = \begin{pmatrix}
\hat{x}_1 - 1 \\
\hat{x}_2 \\
\hat{x}_3
\end{pmatrix},
\eta_4^{\text{RTO}}(\hat{x}) = \begin{pmatrix}
\hat{x}_1 \\
\hat{x}_2 \\
\hat{x}_3 - 1
\end{pmatrix}.
\]

In the following, \( F_K \) denotes the affine element mapping \( F_K(\hat{x}) := B_K \hat{x} + b_K \) from the reference element to an element \( K \) in the mesh. 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]):

\[
\eta^{\text{RTO}}(x) = \frac{1}{\det B_K} B_K \eta^{\text{RTO}}(F_K^{-1}(x)) \quad \text{and} \quad \text{div } \eta^{\text{RTO}}(x) = \frac{1}{\det B_K} \text{div } \eta^{\text{RTO}}(F_K^{-1}(x)).
\]

Figure 1: Degrees of freedom of linear edge elements in reference configuration.
The reference basis functions of the Nédélec element are (see, e.g., [6]):

**2D:**
\[ \eta_{1}^{\text{Ned0}}(\hat{x}) = \begin{pmatrix} -\hat{x}_2 \\ \hat{x}_1 \end{pmatrix}, \quad \eta_{2}^{\text{Ned0}}(\hat{x}) = \begin{pmatrix} -\hat{x}_2 \\ \hat{x}_1 - 1 \end{pmatrix}, \quad \eta_{3}^{\text{Ned0}}(\hat{x}) = \begin{pmatrix} 1 - \hat{x}_2 \\ \hat{x}_1 \end{pmatrix}, \]

**3D:**
\[ \eta_{1}^{\text{Ned0}}(\hat{x}) = \begin{pmatrix} 1 - \hat{x}_3 - \hat{x}_2 \\ \hat{x}_1 - \hat{x}_2 \end{pmatrix}, \quad \eta_{2}^{\text{Ned0}}(\hat{x}) = \begin{pmatrix} \hat{x}_2 \\ 1 - \hat{x}_3 - \hat{x}_1 \end{pmatrix}, \quad \eta_{3}^{\text{Ned0}}(\hat{x}) = \begin{pmatrix} \hat{x}_3 \\ \hat{x}_1 - \hat{x}_2 \end{pmatrix}, \quad \eta_{4}^{\text{Ned0}}(\hat{x}) = \begin{pmatrix} -\hat{x}_2 \\ \hat{x}_1 \\ 0 \end{pmatrix}, \quad \eta_{5}^{\text{Ned0}}(\hat{x}) = \begin{pmatrix} 0 \\ -\hat{x}_3 \\ \hat{x}_2 \end{pmatrix}, \quad \eta_{6}^{\text{Ned0}}(\hat{x}) = \begin{pmatrix} \hat{x}_3 \\ 0 \\ -\hat{x}_1 \end{pmatrix}. \]

Again, we need to use a Piola mapping in order to preserve the tangential continuity. The values are mapped as follows (see, e.g., [6, 10]):
\[ \eta^{\text{Ned0}}(x) = B_K^{-T} \eta^{\text{Ned0}}(F_K^{-1}(x)). \]

The rotation is mapped differently depending on the dimension:

**2D:**
\[ \text{curl} \eta^{\text{Ned0}}(x) = \frac{1}{\det B_K} \text{curl} \eta^{\text{Ned0}}(F_K^{-1}(x)), \]

**3D:**
\[ \text{curl} \eta^{\text{Ned0}}(x) = \frac{1}{\det B_K} B_K \text{curl} \eta^{\text{Ned0}}(F_K^{-1}(x)). \]

We are interested in assembly of the stiffness matrices $K_{ij}^{\text{RT0}}$, $K_{ij}^{\text{Ned0}}$ and the mass matrices $M_{ij}^{\text{RT0}}$, $M_{ij}^{\text{Ned0}}$ defined by
\[
K_{ij}^{\text{RT0}} = \int_{\Omega} \text{div} \eta_{i}^{\text{RT0}} \text{div} \eta_{j}^{\text{RT0}} \, dx, \quad K_{ij}^{\text{Ned0}} = \int_{\Omega} \text{curl} \eta_{i}^{\text{Ned0}} \cdot \text{curl} \eta_{j}^{\text{Ned0}} \, dx,
\]
\[
M_{ij}^{\text{RT0}} = \int_{\Omega} \eta_{i}^{\text{RT0}} \cdot \eta_{j}^{\text{RT0}} \, dx, \quad M_{ij}^{\text{Ned0}} = \int_{\Omega} \eta_{i}^{\text{Ned0}} \cdot \eta_{j}^{\text{Ned0}} \, dx,
\]
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, we are able to assemble the local matrices using the reference element. For example, by using (1), the local stiffness matrices related to the global matrices $K_{ij}^{\text{RT0}}$ and $M_{ij}^{\text{RT0}}$ can be calculated on each element $K \in T$ by
\[
K_{kl}^{\text{RT0},K} = \frac{1}{|\det B_K|} \int_{K} \text{div} \eta_{k}^{\text{RT0}}(\hat{x}) \text{div} \eta_{l}^{\text{RT0}}(\hat{x}), \quad M_{kl}^{\text{RT0},K} = \frac{1}{|\det B_K|} \int_{K} B_K \eta_{k}^{\text{RT0}}(\hat{x}) \cdot B_K \eta_{l}^{\text{RT0}}(\hat{x}),
\]
where $\hat{K}$ is the reference element, i.e., unit triangle in 2D or unit tetrahedron in 3D. 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.

### 3 Implementation of edge elements

We consider triangles in 2D and tetrahedra in 3D and denote by $\#\omega$ the number of elements in the set $\omega$, 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 matrices 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 [8] 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                                      |

| elems2nodes | $\#\mathcal{T} \times 3/4$ | elements by their three/four nodes in 2D/3D (in [8] 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, if one uses the unit triangle as the reference configuration, both the linear Raviart-Thomas elements and the linear Nédélec elements have a degree of freedom related to each of the three edges, totaling three degrees of freedom. In 3D, if one uses the unit tetrahedron as the reference configuration, the linear Nédélec elements will have a degree of freedom related to each of the six edges, and the Raviart-Thomas element will have a degree of freedom related
As an example, we go through the needed program code for calculating the stiffness matrix $K^{RT_0}$ 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.

```matlab
[B_K,-,B_K.det] = affine_transformations(nodes2coord,elems2nodes);
[elems2faces,faces2nodes] = get_faces(elems2nodes);
signs_f = signs_faces(nodes2coord,elems2faces,faces2nodes,B_K);

3.1 Vectorized assembly routine

In 3D, for the linear Nédélec element we need to again know which edge unit tangential vectors to use. The calculation of the orientations in `signs_edges()` is identical to the 2D case. In 3D, for the linear Raviart-Thomas element we need to know which face unit normal vectors to use. With or without going further into details, the faces are defined by their three nodes in the structure `elems2nodes`. We assume we have the mesh in the form of the structures `nodes2coord` and `elems2faces` to each of the four faces. Thus, the structures `elems2{nodes,edges,faces}` define the numbering of global degrees of freedom for a mesh. In Figure 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 know how they are oriented. First, we need to agree what is the positive orientation for an edge or a face in the mesh we have. Then, we proceed as follows. For every element, we have an affine mapping from the reference element to the element in the mesh. The reference element has a certain orientation for the edges and faces. These directions are depicted in Figure 1. If the orientation of an edge, or the orientation of a face is mapped in the same direction as the positive direction we had agreed upon, we assign +1 for this edge/face. Otherwise we assign −1. Then we have the following structures:

$$
\begin{array}{c|c}
\text{signs}_e & \#T \times 3/6 \\
\text{signs}_f & \#T \times 4
\end{array}
\begin{array}{c}
+1 \text{ or } -1 \text{ for every edge of an element, corresponding to } \text{elems2edges} \\
+1 \text{ or } -1 \text{ for every face of an element in 3D, corresponding to } \text{elems2faces}
\end{array}
$$

The situation in 2D is illustrated in Figure 3. In 2D, for the linear Raviart-Thomas element we need to know which edge unit normal vectors to use, and for the linear Nédélec element we need to know which unit tangential vectors to use. In 2D the orientation tangential vectors can be directly deduced from the mesh data. Each triangle in the mesh data is defined by its three nodes in the structure `elems2nodes`. We can simply agree that a triangle’s edge is positively oriented, if the order of the node indexes in `elems2nodes` is such that the beginning node index is smaller than the ending node index. Assuming that the nodes in `elems2nodes` are all oriented the same way (clockwise or counter-clockwise), we arrive at +1 and -1 for every edge (one edge is shared by only two triangles). Then the same sign data can directly be utilized for the normal directions. In our software package the edges are calculated by the function `get_edges()` and the orientation related to edges is calculated by the function `signs_edges()`.

In 3D, for the linear Nédélec element we need to again know which edge unit tangential vectors to use. Without going further into details, the faces are calculated by the function `get_faces()`, and the orientation related to faces is calculated by `signs_faces()`.

Figure 2: Elements by their nodes and edges, i.e., global numbering of degrees of freedom for 2D linear finite elements. The triangle number is denoted by $n$.

Figure 3: Orientation of 2D edge elements sharing an edge. The thick line denotes the agreed positive direction, and $n$ is the triangle number.
On the first line we obtain the affine transformation matrices for all elements \( B_K (2 \times 2 \times \# T \text{ in 2D, and } 3 \times 3 \times \# T \text{ in 3D}) \). Similarly, \( B_K_{\text{det}} (\# T \times 1) \) contains the determinants of \( B_K \). On the second line we obtain the structure \elem2faces, which is the representation of elements by their faces. Note that indeed the numbers in \elem2faces are indexes to \faces2nodes. More importantly, \elem2faces is the numbering of the degrees of freedom for this particular element. On the third line we calculate the orientations for faces in 3D. Then, we call

\[
K_{RT0} = \text{stiffness\_matrix\_RT0(elems2faces, B_K_{\text{det}}, signs_f)};
M_{RT0} = \text{mass\_matrix\_RT0(elems2faces, B_K, B_K_{\text{det}}, signs_f)};
\]

to assemble the stiffness and mass matrices. The main part of the function \text{stiffness\_matrix\_RT0} is the vectorized assembly

```matlab
1 B_K_{\text{det}}A = abs(B_K_{\text{det}});
2 [~,w,nip] = intquad(1,3);  
3 [~,dval,nbasis] = basis\_RT0(ip);
4 STIFF = zeros(nbasis,nbasis,nelems);
5 for i=1:nip
6   for m=1:nbasis
7     for k=m:nbasis
8       STIFF(m,k,:) = squeeze(STIFF(m,k,:)) + ... 
9         w(i).*B_K_{\text{det}}A.^(-1).*... 
10          ( signs_f(:,m).*dval(i,:,m) ) .* ... 
11            ( signs_f(:,k).*dval(i,:,k) );
12     end
13   end
14 end
15 STIFF = copy\_triu(STIFF);
```

The function \([ip,w,nip] = \text{intquad}(po,\text{dim})\) returns an integration quadrature of order \( po \) in the reference element (triangle if \( \text{dim} = 2 \), and tetrahedron if \( \text{dim} = 3 \)). The quadrature consists of the integration points \( ip \) and the weights \( w \). The variable \( \text{nip} \) is the number of integration points. The function \([\text{val,dval},\text{nbasis}] = \text{basis\_RT0}(ip)\) returns the values \( \text{val} \) and divergence values \( \text{dval} \) 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 \( \text{nbasis} \) is the number of basis functions per element. On line 4, the variable \( \text{STIFF} \) is initialized to be of suitable size to contain all the local element matrices.

Note that the outer \( \text{for} \)-loop on line 5 is not over elements, but over integration points. There is no need to loop over elements, since our data structures contain information for all elements at the same time. Instead, we have a \( \text{for} \)-loop over integration points, which is of course a much smaller loop, resulting in much faster performance. This is what we mean by vectorization of the \( \text{for} \)-loop over elements. 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 lines 8–11 we assemble the local matrix entry \((m,k)\) (for the integration point \( i \)) for all elements at the same time. Here the divergences of the reference basis functions are transformed using the Piola transform (1), and the assembly is done according to (2). 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 7 begins from the previous loop index \( m \), and not 1. On line 15 the symmetric entries are copied to the lower triangular part of \( \text{STIFF} \). After this, the global matrix is assembled from the local matrices in \( \text{STIFF} \), 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 more complicated array operations have to be made. The functions providing this functionality are in a directory called “library\_vectorization”, which was also used in the vectorization of nodal elements in [8].

### 3.2 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 2.67GHz, 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 \( \sim 2.4 \text{ billion} \times 2.4 \text{ billion} \) 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 | $K^{RTO}$ | $M^{RTO}$ | assembly of $K^{Ned0}$ | $M^{Ned0}$ |
|-------|-----------------|-----------|-----------|------------------------|-----------|
| 5     | 9344            | 0.03      | -         | 0.03                   | -         |
| 6     | 37120           | 0.11 (3.6)| 0.51 (8.5)| 0.11 (3.6)             | 0.47 (15.6)|
| 7     | 147968          | 0.41 (3.7)| 1.08 (2.1)| 0.40 (3.6)             | 1.02 (2.1)|
| 8     | 590848          | 1.70 (4.1)| 3.59 (3.3)| 1.82 (4.5)             | 3.65 (3.5)|
| 9     | 2361344         | 7.49 (4.4)| 12.82 (3.5)| 7.49 (4.1)             | 12.94 (3.5)|
| 10    | 9441280         | 30.89 (4.1)| 52.09 (4.0)| 30.83 (4.1)             | 54.86 (4.2)|
| 11    | 37756928        | 132.95 (4.3)| 216.64 (4.1)| 132.56 (4.2)             | 230.44 (4.2)|
| 12    | 15101328        | 597.37 (4.4)| 919.36 (4.2)| 583.86 (4.4)             | 931.79 (4.0)|
| 13    | 60401254        | 2620.11 (4.3)| 3969.16 (4.3)| 2840.51 (4.8)             | 4121.33 (4.4)|
| 14    | 241598460       | 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 | $K^{RTO}$ | $M^{RTO}$ | assembly of $K^{Ned0}$ | $M^{Ned0}$ |
|-------|-----------------|-----------|-----------|------------------------|-----------|
| 1     | 2808            | 0.02      | -         | 0.09                   | -         |
| 2     | 21600           | 0.14 (7.0)| 0.39 (4.3)| 13428                 | 0.30 (6.0) | 0.79 (8.7)|
| 3     | 169344          | 0.82 (5.8)| 2.18 (5.5)| 102024                | 1.92 (6.4) | 4.53 (5.7)|
| 4     | 1340928         | 7.15 (8.7)| 15.35 (7.0)| 795024                | 15.44 (8.0)| 35.45 (7.3)|
| 5     | 1067248         | 59.37 (8.3)| 125.71 (8.1)| 6276384              | 129.91 (8.4)| 282.14 (8.4)|
| 6     | 8515584         | 503.89 (8.4)| 1054.49 (8.3)| 4987756             | 1125.08 (8.6)| 2291.50 (8.1)|
| 7     | 680361984       | 4437.84 (8.8)| 8717.70 (8.2)| 39768984            | 10232.01 (9.0)| 20028.06 (8.7)|

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

$$-\Delta u = f \text{ in } \Omega, \quad u = 0 \text{ on } \partial\Omega$$

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

$$\int_{\Omega} \nabla u \cdot \nabla w \, dx = \int_{\Omega} f w \, dx \quad \forall w \in \dot{H}^1(\Omega).$$

Assume that $v \in \dot{H}^1(\Omega)$ is an approximation of the exact solution $u$ of (3). Then, a functional error estimate from [7] states that

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

where $M$ is a functional majorant. The norm $|||| := ||||_{L^2(\Omega)}$ denotes the $L^2$-norm for scalar and vector valued functions. The global constant $C_F$ represents the smallest possible constant from the Friedrichs’ inequality $||w|| \leq C_F ||\text{div} w||$ which holds for all $w \in \dot{H}^1(\Omega)$. Note that the estimate (4) 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 $\nabla u$. Since the $M$ contains nondifferentiable norm terms, we apply the Young’s inequality $(a+b)^2 \leq (1+\beta)a^2 + (1+\beta)b^2$ valid for all $\beta > 0$ to obtain

$$(||\nabla u - y|| + C_F ||\text{div} y + f||)^2 \leq \left(1 + \frac{1}{\beta}\right) ||\nabla u - y||^2 + (1+\beta)C_F^2 ||\text{div} y + f||^2 =: M(\nabla u, f, C_F, \beta, y).$$

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

$$(1+\beta)C_F^2 \int_{\Omega} \text{div} y \, \text{div} \phi \, dx + \left(1 + \frac{1}{\beta}\right) \int_{\Omega} y \cdot \phi \, dx = -(1+\beta)C_F^2 \int_{\Omega} f \, \text{div} \phi \, dx + \left(1 + \frac{1}{\beta}\right) \int_{\Omega} \nabla \phi \cdot \phi \, dx \; \forall \phi \in H(\text{div},\Omega)$$

(5)

On the other hand, for a fixed $y$,

$$\beta = \frac{||\nabla u - y||}{C_F ||\text{div} y + f||}$$

(6)

minimizes $M$ amongst all $\beta > 0$. It suggests the following solution algorithm:
Table 3: Majorant calculation for four meshes in 2D.

| Iter | \(\beta\) | \#T = 512 | \(\beta\) | \#T = 131 072 | \(\beta\) | \#T = 2 097 152 | \(\beta\) | \#T = 33 554 432 |
|------|-----------|----------|-----------|-------------|-----------|--------------|-----------|------------------|
|      | M | \(I_{\text{eff}}\) | M | \(I_{\text{eff}}\) | M | \(I_{\text{eff}}\) | M | \(I_{\text{eff}}\) |
| 1    | 1.000 | 0.026203 | 1.72 | 1.000 | 0.001648 | 1.73 | 1.000 | 0.006412 | 1.73 |
| 2    | 3.208 | 0.023159 | 1.52 | 3.294 | 0.001453 | 1.52 | 3.294 | 0.006363 | 1.52 |
| 3    | 3.268 | 0.023159 | 1.52 | 3.294 | 0.001453 | 1.52 | 3.294 | 0.006363 | 1.52 |

Table 4: Majorant calculation for four meshes in 3D.

| Iter | \(\beta\) | \#T = 10 368 | \(\beta\) | \#T = 82 944 | \(\beta\) | \#T = 663 552 | \(\beta\) | \#T = 5 308 416 |
|------|-----------|----------|-----------|-------------|-----------|--------------|-----------|------------------|
|      | M | \(I_{\text{eff}}\) | M | \(I_{\text{eff}}\) | M | \(I_{\text{eff}}\) | M | \(I_{\text{eff}}\) |
| 1    | 1.000 | 0.011794 | 1.59 | 1.000 | 0.006176 | 1.59 | 1.000 | 0.003135 | 1.59 |
| 2    | 3.128 | 0.010396 | 1.40 | 3.512 | 0.005379 | 1.39 | 3.655 | 0.002721 | 1.38 |
| 3    | 3.420 | 0.010388 | 1.40 | 3.622 | 0.005379 | 1.39 | 3.687 | 0.002720 | 1.38 |
| 4    | 3.432 | 0.010388 | 1.40 |                  |          |          | 3.706 | 0.001365 | 1.38 |

Algorithm 1 (Majorant minimization algorithm). Let \(\beta > 0\) be given.

(a) Compute \(y\) (using current value of \(\beta\)) by minimizing the quadratic problem \(M(\nabla v, f, CF, \beta, y) \rightarrow \min\).

(b) Update \(\beta\) (using \(y\) calculated in step (a)) from (6). 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 (5) with the linear Raviart-Thomas elements. For this both of the FEM matrices \(M^{\text{RTO}}\) and \(K^{\text{RTO}}\) were needed (see also [12]).

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 \(u(x) := \prod_{i=1}^d x_i(x_i - 1) \in H^1(\Omega)\) as the exact solution in both 2D and 3D.

On Tables 3 and 4 we have calculated the majorant \(M\) values with Algorithm 1 for four different meshes in 2D and 3D, respectively. The approximation \(v\) was calculated with linear nodal finite elements. For measuring the quality of the chosen free parameters \(\beta\) and \(y\), we have also included the values of the so-called efficiency index \(I_{\text{eff}} := \sqrt{M}/ \|\nabla (u - v)\| \geq 1\). The approximation \(v\) and flux approximation \(y\) of the smallest 2D mesh are depicted in Figure 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 \(10^{-4}\).

Figure 4: Discrete solution \(v\) (left), and the flux approximation \(y\) first and second components (middle and right). The used mesh consists of 512 elements.

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

\[
\begin{align*}
\operatorname{curl} \mu^{-1} \operatorname{curl} E + \kappa E &= F \quad \text{in } \Omega, \\
E \times n &= 0 \quad \text{in } \Gamma_D, \\
\mu^{-1} \operatorname{curl} E &= 0 \quad \text{in } \Gamma_N, 
\end{align*}
\]

(7)

for a searched function \(E \in \tilde{H}_D(\operatorname{curl}, \Omega):= \{v \in H(\operatorname{curl}, \Omega) \mid v \times n = 0 \text{ on } \Gamma_D\}\), 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 generalized formulation of (7) is

\[
\int_{\Omega} \mu^{-1} \operatorname{curl} E \operatorname{curl} w \, dx + \int_{\Omega} \kappa E \cdot w \, dx = \int_{\Omega} F \cdot w \, dx \quad \forall w \in \tilde{H}_D(\operatorname{curl}, \Omega).
\]

(8)

Example 2 ([1], p. 40). We choose the unit square \(\Omega := (0,1)^2\) with \(\kappa = \mu = 1\). We split the domain in two parts \(\Omega_1 := \{x \in \Omega \mid x_1 > x_2\}\) and \(\Omega_2 := \Omega \setminus \Omega_1\) in order to define the following discontinuous exact solution:

\[
E|_{\Omega_1}(x) := \begin{bmatrix}
\sin(2\pi x_1) + 2\pi \cos(2\pi x_1)(x_1-x_2) \\
\sin((x_1-x_2)^2(x_1-1)^2x_2) - \sin(2\pi x_1)
\end{bmatrix}, \quad E|_{\Omega_2}(x) := 0.
\]
Table 5: Exact energy errors of approximations of the 2D eddy-current problem on uniformly refined meshes.

| #T  | #E  | \(\|E - v\|^2 + \|\text{curl}(E - v)\|^2\) |
|-----|-----|---------------------------------|
| 512 | 800 | 1.879812e-01                    |
| 2 048 | 3 136 | 9.424178e-02                  |
| 8 192 | 12 416 | 4.715474e-02                  |
| 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                  |
| 8 588 608 | 12 587 008 | 1.473965e-03                  |
| 33 554 432 | 50 339 840 | 7.369826e-04                  |

The exact solution has a continuous tangential component at \(x_1 = x_2\), so \(E \in H(\text{curl}, \Omega)\). Moreover,

\[
\text{curl } E|_{\Omega_1} = 2x_2(x_1 - x_2)(x_1 - 1)(2x_1 - x_2 - 1) \cos (x_2(x_1 - x_2)^2(x_1 - 1)^2), \quad \text{curl } E|_{\Omega_2} = 0.
\]

Clearly the rotation of \(E\) is continuous at \(x_1 = x_2\), and \(\text{curl } E \in \dot{H}^1(\Omega)\). Thus, the exact solution satisfies full Neumann boundary condition, so we set \(\Gamma_D = \emptyset\), and \(\Gamma_N = \partial \Omega\).

We denote by \(v\) an approximation of the exact solution \(E\) of the system (7). In the discretization of (8) we need both the mass and stiffness matrices \(M^{\text{Ned}}\) and \(K^{\text{Ned}}\). We see from Figure 5 that the 2D Nédélec element catches the normal discontinuity on \(x_1 = x_2\). In Table 5 we show how the error decreases as the mesh is uniformly refined.

**Acknowledgements** 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, *A posteriori error control for Maxwell and elliptic type problems*, PhD thesis, Jyväskylä studies in computing, ISSN 1456-5390; 190, ISBN 978-951-39-5730-8, 2014.

[2] C. Bahriawati and C. Carstensen, *Three MATLAB implementations of the lowest-order Raviart-Thomas MFEM with a posteriori error control*, CMAM, volume 5, 2005.

[3] F. Brezzi and 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://math.uci.edu/~chenlong/programming.html](http://math.uci.edu/~chenlong/programming.html) (12.09.2014)

[5] A. Hannukainen and M. Juntunen, *Implementing the finite element assembly in interpreted languages*, 2003.

[6] J. C. Nédélec, *Mixed finite elements in \(\mathbb{R}^3\)*, Numerische Mathematik, 35, 315–341, 1980.

[7] P. Neittaanmäki and S. Repin, *Reliable methods for computer simulation. Error control and a posteriori estimates*, Elsevier, Amsterdam, 2004.

[8] T. Rahman and J. Valdman, *Fast MATLAB assembly of FEM matrices in 2D and 3D: nodal elements*, Applied Mathematics and Computation 219, 7151–7158, 2013.

[9] P. A. Raviart and J. M. Thomas, *Primal hybrid finite element methods for 2nd order elliptic equations*, Math. Comput. 31, 391–413, 1977.

[10] A. Schneebeli, *An \(H(\text{curl}; \Omega)\)-conforming FEM: Nédélec’s elements of first type*, technical report, 2003. [http://www.dealii.org/reports/nedelec/nedelec.pdf](http://www.dealii.org/reports/nedelec/nedelec.pdf) (12.09.2014)

[11] P. Solin, K. Sejogo, and I. Dokezel, *Higher-order finite element methods*, Chapman, 2003.

[12] J. Valdman, *Minimization of functional majorant in a posteriori error analysis based on \(H(\text{div})\) multigrid-preconditioned CG method*, Advances in Numerical Analysis, vol. 2009, Article ID 164519, 2009.

[13] F. Cuveller, C. Japhet, and G. Scarella, *An efficient way to perform the assembly of finite element matrices in Matlab and Octave*, 2013. [http://arxiv.org/abs/1305.3122](http://arxiv.org/abs/1305.3122) (12.09.2014)