1st Pan-American Congress on Computational Mechanics - PANACM 2015
XI Argentine Congress on Computational Mechanics - MECOM 2015
S. Idelsohn, V. Sonzogni, A. Coutinho, M. Cruchaga, A. Lew & M. Cerrolaza (Eds)

ASSESSMENT OF EDGE-BASED FINITE ELEMENT TECHNIQUE FOR GEOPHYSICAL ELECTROMAGNETIC PROBLEMS: EFFICIENCY, ACCURACY AND RELIABILITY

OCTAVIO CASTILLO[1], JOSEP DE LA PUENTE[1], VLADIMIR PUZYREV[1] AND JOSÉ MARÍA CELA[1]

[1]Barcelona Supercomputing Center
Nexus I - Campus Nord UPC - Barcelona - Spain
contact: octavio.castillo@bsc.es

Key words: Finite Element, Geophysics, Electromagnetism and Edge Elements

Abstract. In Finite Element Methods for solving electromagnetic field problems, the use of Edge Elements has become very popular. In fact, Edge Elements are often said to be a cure to many difficulties that are encountered (particularly eliminating spurious solutions) and are claimed to yield accurate results [9, 16, 17]. We will shortly describe the mathematical formulation of linear edge elements and we go through the particular issues related to the implementation of these elements in order to solve geophysical electromagnetic problems. In particular, we describe a simple, flexible and parallel Fortran 90 implementation for Edge Elements. The code is based on an abstract data structure, which allows to use different kinds of solvers with little effort. The result is an implementation that allows users to specify Edge-based Finite Element variational forms of $H(curl)$. Finally, we also show the performance of the code in terms of efficiency, accuracy and reliability, which will shape our future line of work in order to solve more complex problems.

1 INTRODUCTION

The Sobolev spaces $H(div)$ and $H(curl)$ play an important role in many applications of Edge-based Finite Element methods (Edge Elements) to partial differential equations (PDE). Examples include second-order elliptic PDE's, Maxwell's Equations for electromagnetism, and the linear elasticity equations, among others. Edge-based Finite Element methods may provide advantages over standard H^1 Finite Element discretizations in terms of improved flexibility, stability and robustness. However, implementing $H(div)$ and $H(curl)$ methods requires additional code complexity for constructing basis functions

and evaluating variational forms, which together with its relative novelty helps explaing their relative scarcity in practice.

The most important difference between the Nodal Finite Element (Lagrange FE) and the Edge-based Finite Element method (Nédélec Elements) is that the basis functions are not defined on the nodes of 2D triangular and 3D tetrahedral meshes, but on edges and faces, respectively [14]. Edge Elements provide only partial continuity over element boundaries: continuity of the normal vector component for $H(div)$ problems and continuity of the tangential vector component for $H(curl)$ problems.

The method of FE applied to $H(div)$ and $H(curl)$ problems and its implementation has been well documented [1, 16, 17]. Users can find many software codes such as NGSOLVE [3] or FEniCS [12], which are written in object oriented languages allowing for higher order elements defined on elements with curved boundaries. Such codes are well suited for high complexity computations and also provide a 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 Fortran 90 code is more suitable for students and researchers who wish to become familiar with Edge Elements and prefer to make their own modifications. As our target application is exploration geophysics, our computational solution considers the lowest order linear Edge Elements defined on 2D triangles for $H(curl)$ problems only. There is a large bibliography dedicated to constructing nodal elements, but only few publications related to Edge Elements. Our implementation takes may practical ideas from literature [1, 6, 16, 17].

The paper is divided as follows: Section 2 shortly describes the theory associated to linear edge elements without going into details. In Section 3 we go through the particular constructions related to the implementation of these elements. Strong emphasis is placed on aspects not easily found in the literature, such as the choice of orientation of geometric entities. Section 4 describes the software stack, including the most important functions such as the interface that allows to use different kinds of solvers with little effort. In particular, we use the BLAS library [2]. Section 5 shows the performance of the code in terms of efficiency, accuracy and reliability. Finally, we make some concluding remarks in Section 6.

2 LINEAR NÉDÉLEC ELEMENTS FORMULATION

When using nodal elements (Lagrange elements) in electromagnetism, spurious solutions can occur [9, 10, 17]. In this section we summarize some basic facts about the Sobolev space $H(curl)$ and we discuss conforming finite elements associated with them. The reader is referred to [16] for a more thorough analysis of $H(curl)$.

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 rotation of a vector valued function $w : \Omega \to \mathbb{R}^d$ is defined as:

$$\nabla \times w = \begin{pmatrix} \partial_2 w_3 - \partial_3 w_2, \\ \partial_3 w_1 - \partial_1 w_3, \\ \partial_1 w_2 - \partial_2 w_1, \end{pmatrix} \tag{1}$$

Following the approach of [16, 17], we consider two types of rotation operators in 2D, the vector operator $\underline{\text{curl}}$ and the scalar operator curl:

$$\underline{curl} f = \begin{pmatrix} \partial_2 f \\ -\partial_1 f \end{pmatrix} \tag{2}$$

$$curl w = \partial_1 w_2 - \partial_2 w_1 \tag{3}$$

In literature, the operator $\underline{\text{curl}}$ is frequently called the co-gradient. The operators give rise to the standard Sobolev spaces:

$$H(\nabla \times \Omega) = \begin{cases} \{v \in L^2(\Omega\mathbb{R}^3) \mid \nabla \times \ v \ \in L^2(\Omega\mathbb{R}^3)\} & \text{if } d = 3 \\ \{v \in L^2(\Omega\mathbb{R}^2) \mid \nabla \times \ v \ \in L^2(\Omega)\} & \text{if } d = 2 \end{cases} \tag{4}$$

where L^2 denotes the space of the square Lebesgue integrable functions. Assuming that Ω is discretized by a triangular (2D) or a tetrahedral (3D) mesh $\mathbb{M}$, Nédélec Elements represent basis functions in $H(curl, \mathbb{M})$ spaces. Figure 1 shows the numbering of the degrees of freedom of the linear Edge elements.

On the other hand, we denote the global edge basis functions by n, and by $x = (x_1, x_2, x_3)^T$ the spatial variable in Ω. Similarly, we define the reference basis functions and the spatial variable simply by adding the hat $\hat{\cdot}$, for instance, $\hat{x}$ denotes the spatial variable in the reference element. The reference basis functions of the Nédélec Elements are as follows [1, 14, 16, 17]:

$$\textbf{2D case}: n_1(\hat{x}) = \begin{pmatrix} -\hat{x}_2 \\ \hat{x}_1 \end{pmatrix}, \ n_2(\hat{x}) = \begin{pmatrix} -\hat{x}_2 \\ \hat{x}_1 - 1 \end{pmatrix}, \ n_3(\hat{x}) = \begin{pmatrix} 1 - \hat{x}_2 \\ \hat{x}_1 \end{pmatrix} \tag{5}$$

$$\textbf{3D case}: n_1(\hat{x}) = \begin{pmatrix} 1 - \hat{x}_3 - \hat{x}_2 \\ \hat{x}_1 \\ \hat{x}_1 \end{pmatrix}, \ n_2(\hat{x}) = \begin{pmatrix} \hat{x}_2 \\ 1 - \hat{x}_3 - \hat{x}_1 \\ \hat{x}_2 \end{pmatrix} \tag{6}$$

$$n_3(\hat{x}) = \begin{pmatrix} \hat{x}_3 \\ \hat{x}_3 \\ 1 - \hat{x}_2 - \hat{x}_1 \end{pmatrix}, \ n_4(\hat{x}) = \begin{pmatrix} -\hat{x}_2 \\ \hat{x}_1 \\ 0 \end{pmatrix}$$

$$n_5(\hat{x}) = \begin{pmatrix} 0 \\ -\hat{x}_3 \\ \hat{x}_2 \end{pmatrix}, \qquad n_6(\hat{x}) = \begin{pmatrix} \hat{x}_3 \\ 0 \\ -\hat{x}_1 \end{pmatrix}$$

In the following, F_K denotes the affine element mapping $F_K := B_K \hat{x} + b_k$ from the reference element to an element K in the mesh. In order to preserve the tangential

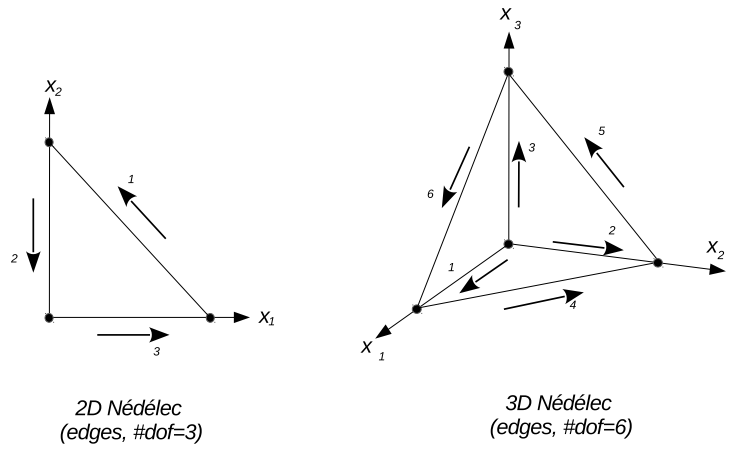

Figure 1: Degrees of freedom of linear Edge Elements (Reference configuration).

continuity of the reference basis functions, we need to use the so-called Piola mappings. We refer the reader to [4, 13] for a more thorough treatment. In our case, the values are mapped as follows [1, 14, 16, 17]:

$$n(x) = B_K^{-T} n(F_K^{-1}(x)) \tag{7}$$

Finally, the rotation is mapped differently depending on the dimension [1, 16, 17]:

$$\textbf{2D case} : n(x) = \frac{1}{detB_k}\ curl\ n(F_K^{-1}(x)) \tag{8}$$

$$\textbf{3D case} : n(x) = \frac{1}{detB_k} B_K\ curl\ n(F_K^{-1}(x))$$

3 PARTICULARITIES OF EDGE ELEMENTS

In this section we shortly describe three particularities to construct an Edge Elements solution: Piola mapping in order to guarantee global continuity of $H(curl)$ problems, numbering scheme and edges direction.

3.1 Piola mapping

First, it follows from Stokes theorem that in order for piecewise $H(curl)$ vector fields to belong to in $H(curl)$ globally, the traces of tangential components over patch interfaces must be continuous. In order to do that, one must consider the covariant Piola mapping which is defined by:

$$F^{curl}(\Phi) = J^{-T}\Phi \circ F^{-1} \tag{9}$$

Figure 2 depicts the vector field Φ between two triangles using the covariant Piola mapping which preserves tangential traces of vector fields [16].

We refer the reader to [4, 16, 17] for a more thorough treatment.

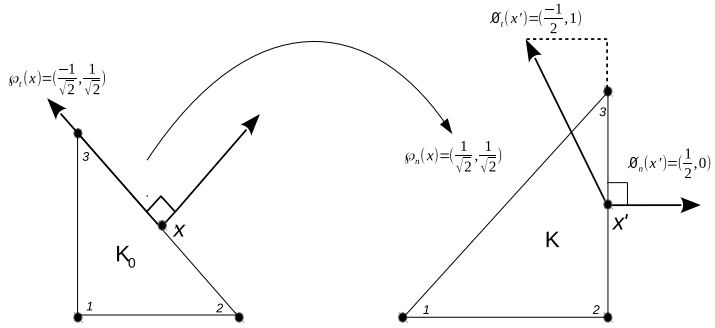

Figure 2: Covariant Piola mapping of vector field Φ in 2D case.

3.2 Numbering strategy

In order to guarantee global continuity with Piola mapping, special care has to be taken with regard to the numbering strategy. Our implementation is based on UFC (Unified Form-assembly Code), which is a unified framework for finite element assembly. More precisely, it defines a fixed interface to communicate low level routines (functions) for evaluating and assembling FE variational forms [12].

In short, the numbering strategy works as follows. A global index is assigned to each node of the mesh $\mathbb{M}$ (consisting of triangles or tetrahedra). If an edge adjoints two nodes n_i and n_j, we define the direction of the edge as going from node n_i to node n_j if $i < j$. This gives a unique orientation of each edge. The same philosophy is used locally to determine the directions of the local edges on each element. Thus, if an edge connects the second and the third node of a tetrahedron, then the direction is from the second to the third node.

A similar numbering scheme is employed for faces, the key point now is to require that the nodes of each element are always ordered based on the their global indices. More details may be found in [9, 13].

3.3 Edges direction

Since the degrees of freedom are integrals over edges or faces, we need to know how they are oriented. To do that, we need to include what is the positive orientation for an edge or a face in the mesh $\mathbb{M}$. Then, we proceed as follows. First, for every element we have an affine mapping from the reference element to the element in the mesh. Second, the reference element has a certain orientation for the edges and faces. Thus, if the orientation of an edge, or the orientation of a face is mapped in the same direction as positive direction we had agreed upon, we assign $+1$ for this edge/face. Otherwise we assign -1. These directions are depicted in Figure 3 [1, 16, 17].

In 3D, we need to again know which edge unit tangential vectors to use, hence, the calculation of the orientations is identical to the 2D case.

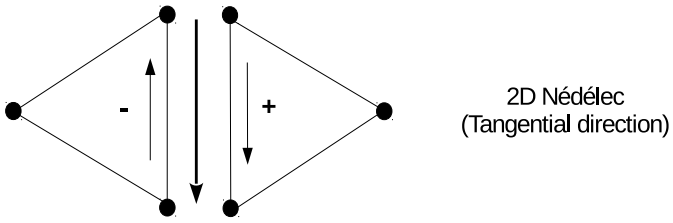

Figure 3: Orientation of 2D edge elements sharing an edge (the thick line denotes the agreed positive direction).

4 SOFTWARE STACK

In this section we describe the software stack of our solution including the most important data structures. An outline on the primary groups of functions in our implementation is given in Figure 4. A more detailed explanation is the following:

1. Mesh. This module stores geometric and topological properties of a mesh: how are the elements connected and where are their nodes. The properties and data of meshes are almost always queried through loops over all elements, possibly querying all faces of each element as well (3D). Our implementation is able to read as input nodal-based meshes.

2. Edge-FE. Describe the properties of a Edge-based Finite Element space as defined on the reference element. This includes, for example, individual shape functions at edges on the reference element.

3. Quadrature. As with FE, quadrature objects are defined on the reference element. Includes the location of quadrature points on the unit element, and the weights of quadrature points thereon.

4. Initializer. This module is the confluence of meshes and Edge-FE, in other words, Edge-FE describes how many degrees of freedom it needs per node or edge, and Initializer module allocates this space so that node or edge of the mesh has the correct number of them. It also gives them a global numbering.

5. Mapping. Computes matrix and right hand side (RHS) entries or other quantities on each element in the mesh, using the shape functions of a Edge-FE shape functions and quadrature points defined by a quadrature rule. To do that, it is necessary to map (Affine mapping) the shape functions, quadrature points and quadrature

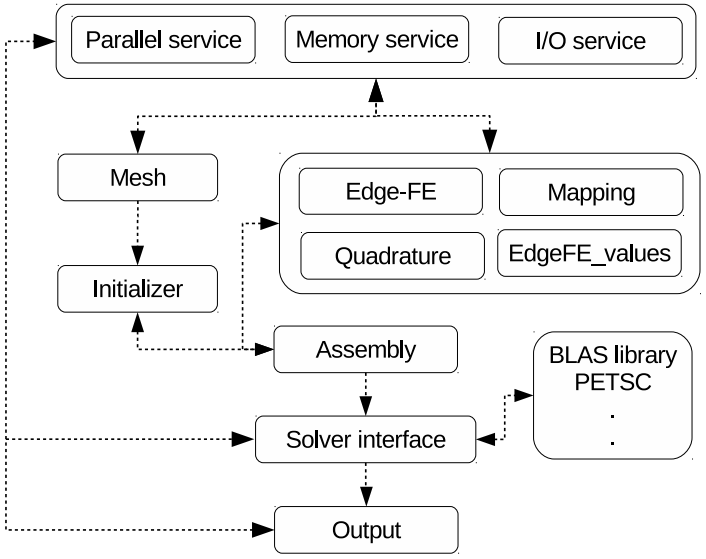

Figure 4: Upper view of software stack

weights from the unit element to each element in the mesh. This is directly done by Mapping module. It also provides support to compute signs for each edge in 2D or faces in 3D.

6. EdgeFE_values. This module takes the Edge-FE and evaluate its shape functions and their values at the points defined by a quadrature rule when mapped to the real element.

7. Assembly. This module assembles the system matrix and RHS of the linear system. We will determine the solution of our problem from this linear system. To this end, we have subroutines that store and manage in an efficient way the entries of matrices and vectors.

8. Solver interface. In order to find the solution of a linear system, one needs linear solvers. In FE applications, they are frequently iterative, but sometimes one may also want to use direct solvers. Since our implementation takes advantages of external solvers such as BLAS Library or PETSC to name a few, we need the Linear solver module, which allows communication with the external solvers.

9. Output. Once one has obtained a solution of a Edge-FE problem on a given mesh, one will to postprocess it using a visualization program. Our software doesn't do the visualization by itself, but rather generates output files with the final results [12]. It also gives timing values in order to evaluate the performance.

10. Common modules. Finally, our implementation has three fundamental services: parallel service, memory service and I/O service. In other words, common modules is a toolbox that provide a variety of independent procedures to be called by other modules or subroutines.

About data structures, our implementation is based on ideas of [1, 5, 9]. We consider triangles in 2D and tetrahedra in 3D and denote by $\#\varpi$ the number of elements in the set of ϖ, and by $\mathcal{N}, \mathcal{E}, \mathcal{F}$ and $\mathcal{T}$ the sets of Nodes, Edges, Faces and Elements, respectively. Table 1 describes the elemental matrices that are needed in order to implement Edge Elements. All structures are based on Column-Major Order approach.

Table 1: Elemental matrices to represent Edge Elements.

Name	Dimensions	Description
nodes2coord	2/3 x $\#\mathcal{N}$	Nodes defined by their 2/3 coordinates in 2D/3D
edges2nodes	2 x $\#\mathcal{E}$	Edges defined by their 2 nodes in 2D/3D
faces2nodes	3 x $\#\mathcal{F}$	Faces defined by their 3 nodes in 3D

With the previous matrices available, we can express every element by the list of its nodes, edges or faces as states Table 2, note that faces $\mathcal{F}$ exist only in 3D case.

Table 2: Element lists by their nodes, edges or faces

Name	Dimensions	Description
elems2nodes	3/4 x $\#\mathcal{T}$	Elements by their 3/4 nodes in 2D/3D
elems2edges	3/6 x $\#\mathcal{T}$	Elements by their 3/6 edges in 2D/3D
elems2faces	4 x $\#\mathcal{T}$	Elements by their 4 faces in 3D

We recall that we use first-order elements. Hence, in 2D, if one uses the unit triangle as the reference configuration, Nédélec Elements have a degree of freedom related to each of the three edges. In 3D case, if one uses the unit tetrahedron as reference configuration, Nédélec Elements will have a degree of freedom related to each of the six edges. Hence, the structures elems2nodes, elems2edges and elems2faces define the numbering of global degrees of freedom for a given mesh. Figure 5 provides a simple example of the mesh data in 2D.

5 PERFORMANCE ISSUES

In order to test the performance of our solution, we focus in assembly time, solving time, and convergence order. Our tests are based on ideas from [1], where the authors solve

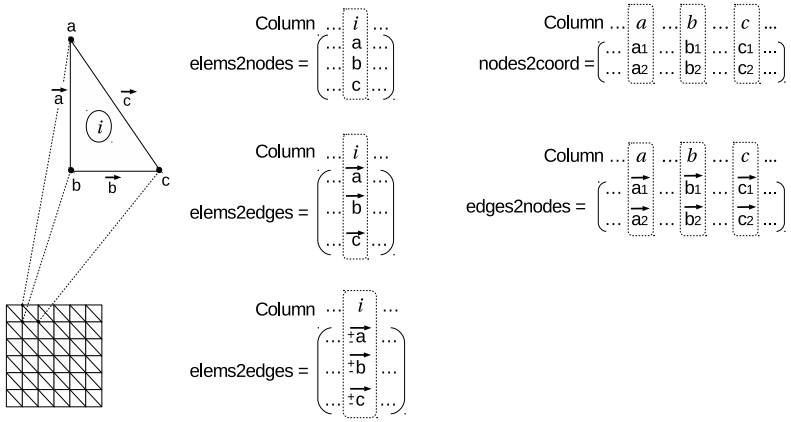

Figure 5: Elements by their nodes, edges and signs in 2D case. The element number is denoted by i.

the eddy-current problem using Nédélec Elements. The experiments were performed on a simple node of the Marenostrum supercomputer with two $8-core$ Intel Xeon processors $E5-2670$ at 2.6 GHz.

Uniform refinement in 2D results in 4 times more elements. In this work, the BLAS library is compiled and linked with our code in order to use the SGESV subroutine as solver [8]. Table 3 depicts the performance of our code with 16 OpenMP threads in terms of assembly time and solver time. Table 3 also includes the mesh spacing (h), L^2 error and the convergence order which is plotted also in Figure 6.

Table 3: Summary of results for the 2D case. Number of elements ($\mathcal{T}$), number of edges ($\mathcal{E}$), assembly time (seconds), solver time (seconds), mesh spacing (h), L^2 error and convergence order.

$\#\mathcal{T}$	$\#\mathcal{E}$	**Assembly time**	**Solver time**	h	L^2	$\mathcal{O}_{L^2}$
128	208	$7.66\cdot 10^{-3}$	$6.76\cdot 10^{-3}$	$3.3\cdot 10^{-1}$	$3.969\cdot 10^{-1}$	–
512	800	$6.75\cdot 10^{-3}$	$7.72\cdot 10^{-3}$	$2.0\cdot 10^{-1}$	$9.831\cdot 10^{-2}$	2.016
2,048	3,136	$2.22\cdot 10^{-2}$	$4.47\cdot 10^{-2}$	$1.1\cdot 10^{-1}$	$2.430\cdot 10^{-2}$	2.017
8,192	12,416	$1.82\cdot 10^{-1}$	$2.08\cdot 10^{-1}$	$5.8\cdot 10^{-2}$	$6.001\cdot 10^{-3}$	1.999
32,768	49,408	$4.27\cdot 10^{-1}$	$6.68\cdot 10^{-1}$	$3.0\cdot 10^{-2}$	$1.501\cdot 10^{-3}$	2.004
131,072	197,120	$1.51\cdot 10^{0}$	$2.27\cdot 10^{0}$	$1.5\cdot 10^{-2}$	$3.741\cdot 10^{-4}$	2.002
524,288	787,456	$5.83\cdot 10^{0}$	$6.71\cdot 10^{0}$	$7.8\cdot 10^{-3}$	$9.340\cdot 10^{-5}$	2.002
2,097,152	3,147,776	$21.32\cdot 10^{0}$	$24.7\cdot 10^{0}$	$3.9\cdot 10^{-3}$	$2.333\cdot 10^{-5}$	2.001

Figure 7 depicts the discrete solution of eddy-current problem in a mesh with $\mathcal{T} = 32,768$ and $\mathcal{E} = 49,408$.

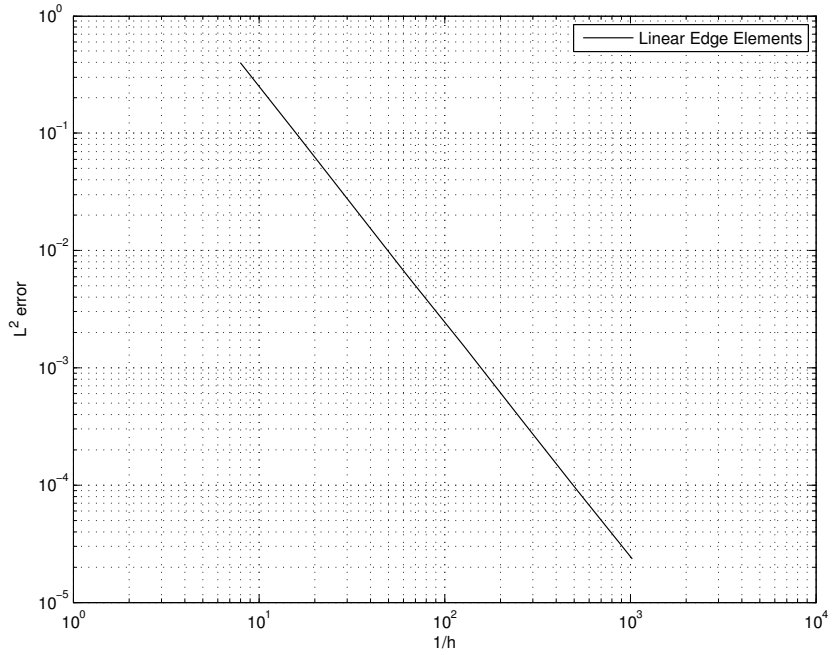

Figure 6: Visualization of the convergence results of the linear Edge Elements in 2D. The L^2 error is plotted versus the mesh spacing h.

6 CONCLUSIONS

The relative scarcity of Edge-based Finite Elements in practical use may be attributed to their higher theoretical and implementational cost. Indeed, more care and effort are required to implement them: basis functions, Piola mapping, edge directions and numbering strategy. However, as demostrated in this work, the implementation of Nédélec Elements for $H(curl)$ conforming may be automated. Particularly, the additional challenges in the assembly process can be viewed as not essentially different from those encountered in other approaches such as higher order Lagrange elements.

The first version of our framework, based on a simple mathematical approach [1], is able to resolve $H(curl)$-conforming problems in 2D. The described software stack relies on a flexible implementation which allows a general point of view. The efficiency and accuracy of the code is evaluated through a convergence test, assembly time and solver time, with a particular emphasis on the performance when the number of elements and degrees of freedom grows.

In our future work we will implement the 3D case, and we will also address issues that are related to the optimization of the code, i.e. solution techniques and in particular efficient assembly algorithms and preconditioning techniques that reduce the global system size and allow for a significant speed-up of the linear system solver. In this context, we will also investigate on domain decomposition techniques and parallelization in the spirit of [10, 11, 15]. Another natural extension of the presented framework is to address

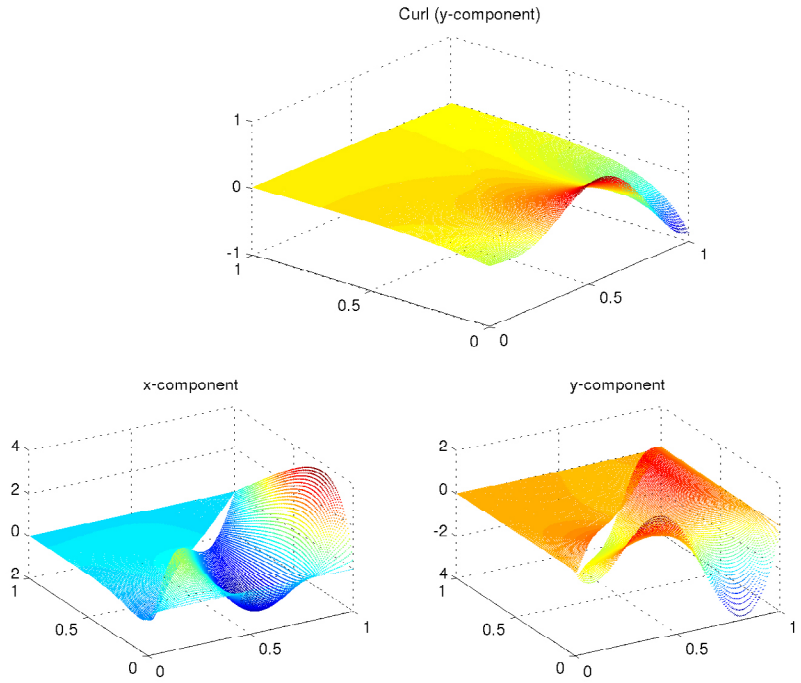

Figure 7: Discrete solution of eddy-current problem: curl(above), x-component(bottom-left) and y-Component(bottom-right).

inhomogeneous domains and adding dipole sources.

We believe that our Fortran 90 code is suitable for students and researchers who wish to become familiar with Edge Elements and prefer to have their own implementation.

REFERENCES

[1] Anjam, I., Valdman, J. (2014). *Fast MATLAB assembly of FEM matrices in 2D and 3D: Edge Elements.* Cornell University Library. <http://arxiv.org/abs/1409.4618> (01.08.2015).

[2] Blackford, L. S., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., Heroux, M., Kaufman, L., Lumsdaine, A., Petitet, A., Pozo, R., Remington, K., Whaley, R. C. (2002). *An updated set of basic linear algebra subprograms - BLAS.* ACM Trans. Math. Soft., 28-2. 135-151.

[3] Boffi, D., Kikuchi, F., Schöberl, J. (2006). *Edge element computation of Maxwell's eigenvalues on general quadrilateral meshes.*

[4] Brezzi, F., Fortin, M. (1991). *Mixed and hybrid Finite Element Methods.* Springer Ser. Comput. Math. 15.

[5] Burnett, D. S. (1987). *Finite element analysis: from concepts to applications.* Addison Wesley.

[6] Chen, L. (2009). *iFEM: an integrated finite element method package in MATLAB.* Technical report, University of California at Irvine. <http://users.tkk.fi/ mojuntun/preprints/matvecSISC.pdf> (01.08.2015).

[7] Chleboun, J., Solin, P. (2013). *On optimal node and polynomial degree distribution in one-dimensional hp-FEM.* Computing, Volume 95, Issue 1, 75-88

[8] Hogben, L. (2006). *Handbook of linear algebra - Discrete mathematics and its applications.* CRC Press.

[9] Jin, J. (2002). *The finite element method in electromagnetics.* John Wiley and Sons.

[10] Koldan, J. (2013). *Numerical solution of 3-D electromagnetic problems in exploration geophysics and its implementation on massively parallel computers.* UPC Thesis.

[11] Koldan, J., Puzyrev, V., de la Puente, J., Houzeaux, G., Cela, J.M. (2014). *Algebraic multigrid preconditioning within parallel finite-element solvers for 3-D electromagnetic modelling problems in geophysics.* Geophysical Journal International, 197(3), 1442-1458.

[12] Logg, A., Mardal, K., Wells, G. (2011). *Automated solution of differential equations by the finite element method: The FEniCS.* Springer Science & Business Media.

[13] Monk, P. (2003). *Finite Element Methods for Maxwell's Equations.* Oxford University Press.

[14] Nédélec, J.C. (1980). *Mixed finite elements in* $\mathbb{R}^3$. Numerische Mathematik, 35(3), 315-341. Mathematical models and methods in applied sciences. <http://http://www.hpfem.jku.at/publications/bks.pdf> (01.15.2015).

[15] Puzyrev, V., Koldan, J., de la Puente, J., Houzeaux, G., Vázquez, M., Cela, J.M. (2013). *A parallel finite-element method for three-dimensional controlled-source electromagnetic forward modelling.* Geophysical Journal International, ggt027.

[16] Rognes, M. E., Kirby, R. C., Logg, A. (2009). *Efficient assembly of H(div) and H(curl) conforming finite elements.* SIAM Journal on Scientific Computing, 31(6), 4130-4151.

[17] Schneebeli, A. (2003). *An H(curl)-conforming FEM: Nédélec's elements of first type.* Technical report. <http://www.dealii.org/8.0.0/reports/nedelec/nedelec.pdf> (01.15.2015).