# IMPLEMENTATION AND EXPERIMENTS WITH THE DISCONTINUOUS GALERKIN METHOD FOR MAXWELL'S EQUATIONS

BY

HEITOR DAVID PINTO

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2009

Urbana, Illinois

Adviser:

Professor Jianming Jin

## ABSTRACT

This thesis presents the mathematical derivation and implementation of, and improvements to, the discontinuous Galerkin method (DGM) for solving Maxwell's equations. Each step leading to the development of a computer code for this method is explained in detail, and samples codes are included in the Appendix. This work also shows numerical results of several experiments with the method, namely: simulation of simple electromagnetic problems with a known analytical solution for comparison and error analysis; comparison of different time discretization schemes, which are not strictly part of DGM; reduction of computation time with the use of adaptive time steps; and analysis of accuracy of absorbing boundaries in scattering problems. A discussion listing advantages and limitations of DGM concludes this work.

## ACKNOWLEDGMENTS

The author would like to thank his adviser, Professor Jianming Jin, for all his guidance, patience and understanding during the development of the research leading to this thesis. This research would not have been possible without his initial suggestion of the topic, useful insight, and valuable suggestions and comments. The author would also like to acknowledge the help of his fellow students Rui Wang, Xiaolei Li, Wang Yao, Mingfeng Xue, Huan-Ting Meng, and Su Yan, who provided many comments and helped solve some technical difficulties during the validation of the computer code.

## TABLE OF CONTENTS

## 1. INTRODUCTION

The solution of electromagnetic problems is an essential part of the development of many new technologies, such as stealth airplanes, medical imaging devices, and a wide range of components used in telecommunications. The increasing demand for these technologies creates more complex problems, in which Maxwell's equations cannot be solved analytically and so require numerical methods.

Some of the current numerical methods for solving Maxwell's equations are based on finite difference schemes, which approximate derivatives. Standard finite difference schemes require a rectangular grid, imposing a strong limitation on the geometry of the problem. Nevertheless, these schemes are still often used because they enable very fast simulations and low storage. Other numerical methods are based on finite elements and basis functions, into which the main functions are decomposed, and the derivatives are calculated exactly. Finite elements provide a greater geometrical flexibility and thus better accuracy than finite difference schemes, but require more memory and longer simulations, because the use of basis functions involves matrix calculations.

A method that has recently been introduced to solve electromagnetic problems is the discontinuous Galerkin method (DGM). It was first proposed by Reed and Hill in 1973 to solve the neutron transport equation [1], and since then, the method has been analyzed further regarding convergence and stability, and more recently it has been applied to various areas, such as fluid dynamics, acoustics, and electromagnetism [2]. DGM is also based on finite elements, but it can achieve fast simulations by greatly reducing the order of matrices. The values are calculated separately in each element, and any differences in adjacent elements are considered through numerical fluxes across the boundary between elements. The inclusion of numerical fluxes also facilitates the treatment of external boundaries. DGM only concerns the spatial derivatives in Maxwell's equations, and the time derivatives are usually dealt with using advanced finite difference schemes, such as the Runge-Kutta method.

Some challenges exist in the application of DGM to Maxwell's equations. For example, to achieve stability, a certain relation between the element sizes and time steps must be satisfied, and this requirement often results in longer simulations if there is considerable variation in element sizes. One successful proposal to overcome this limitation is to use adaptive time steps [3]. In scattering problems, another challenge is the need for a fictitious absorber, such as the

perfectly matched layer (PML), which has also been applied to DGM [4]. The PML is very efficient in absorbing incoming waves, but it requires many additional variables. These challenges are not necessarily disadvantages of DGM, since they are present in other numerical methods as well. However, some methods may be more efficient depending on specific problems. For instance, the method of moments is often preferred for scattering problems, since it does not need an absorber [5].

This work presents the theoretical basis of DGM, suggestions for its implementation, examples of results, and improvements. Chapter 2 gives an overview of Maxwell's equations and of a finite difference scheme, defines basis functions, and discusses the finite element method. This discussion is necessary for a better understanding of DGM. Chapter 3 presents a detailed derivation of DGM applied to Maxwell's equations, including the derivation of the numerical flux, for use in two-dimensional problems. Chapters 4–6 concern different time discretization schemes, implementation of variables, sources and boundaries, and validation of the method through some problems. Chapter 7 suggests some improvements, such as adaptive time steps, and shows the formulation of the PML, with examples. A discussion of the advantages and limitations of DGM concludes this work.

## 2. MAXWELL'S EQUATIONS AND NUMERICAL METHODS

Electromagnetic phenomena are described by Maxwell's equations. In integral form, they are

$$\oint_C \mathbf{E}\cdot d\mathbf{l} = -\frac{\partial}{\partial t}\iint_S \mathbf{B}\cdot d\mathbf{s} \tag{2.1}$$

$$\oint_C \mathbf{H}\cdot d\mathbf{l} = \frac{\partial}{\partial t}\iint_S \mathbf{D}\cdot d\mathbf{s} + \iint_S \mathbf{J}\cdot d\mathbf{s} \tag{2.2}$$

$$\oiint_S \mathbf{D}\cdot d\mathbf{s} = \iiint_V \rho\, dv \tag{2.3}$$

$$\oiint_S \mathbf{B}\cdot d\mathbf{s} = 0 \tag{2.4}$$

where **E** is the electric field intensity, **H** is the magnetic field intensity, **D** is the electric flux density, **B** is the magnetic flux density, **J** is the electric current density, and $\rho$ is the electric charge density. When solving electromagnetic problems, it is often desirable to use the divergence theorem to transform these equations into differential form:

$$\nabla\times\mathbf{E} = -\frac{\partial\mathbf{B}}{\partial t} \tag{2.5}$$

$$\nabla\times\mathbf{H} = \frac{\partial\mathbf{D}}{\partial t} + \mathbf{J} \tag{2.6}$$

$$\nabla\cdot\mathbf{D} = \rho \tag{2.7}$$

$$\nabla\cdot\mathbf{B} = 0. \tag{2.8}$$

In this form, the equations must be accompanied by a set of boundary conditions, which concern field discontinuities on an interface between different materials:

$$\hat{n}\times(\mathbf{E}_1 - \mathbf{E}_2) = 0 \tag{2.9}$$

$$\hat{n}\times(\mathbf{H}_1 - \mathbf{H}_2) = \mathbf{J}_s \tag{2.10}$$

$$\hat{n}\cdot(\mathbf{D}_1 - \mathbf{D}_2) = \rho_s \tag{2.11}$$

$$\hat{n}\cdot(\mathbf{B}_1 - \mathbf{B}_2) = 0 \tag{2.12}$$

where the subscripts 1 and 2 refer to the media on each side of the interface, $\hat{n}$ denotes the normal to the interface, $\mathbf{J}_s$ is the surface current density, and $\rho_s$ is the surface charge density, existing on the boundary.

The solution of a set of differential equations is generally not simple, especially in the case of Maxwell's equations, since they involve two functions of four variables: **E** and **H** both depend on the three space coordinates and on time. Therefore, several numerical methods exist to

solve such equations. One of the simplest methods is the finite difference method in time domain (FDTD), which approximates all derivatives with divisions:

$$\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}. \tag{2.13}$$

This method enables very fast simulations when compared to other methods. However, it requires a constant $\Delta$ for each variable, which poses a strong limitation on the geometry of the problem. For example, in a two-dimensional problem with constant $\Delta x$ and $\Delta y$, a circle must be discretized with small rectangles, resulting in low accuracy. A better approximation could be obtained with triangles, as shown in Figure 2.1 below, but the standard FDTD does not support this kind of discretization.

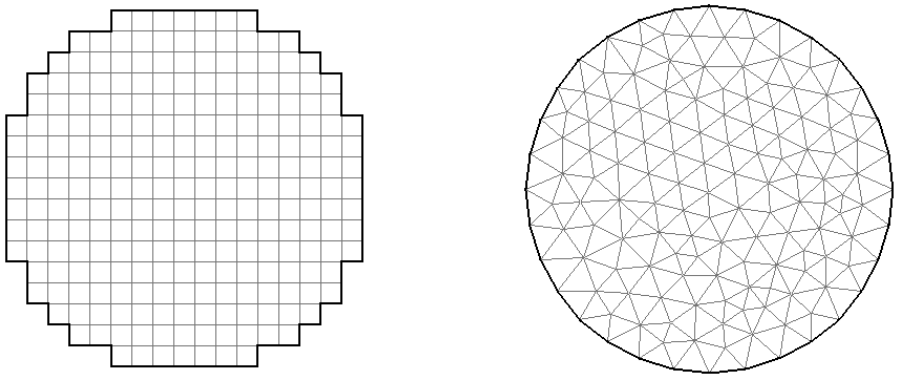


Figure 2.1: Circle discretized with rectangles and with triangles.

The finite element method (FEM) is more flexible than FDTD regarding the geometry of the problem, as FEM allows a discretization of the region of interest into any kind of elements, including triangles. Instead of approximating the derivatives, this method consists of approximating the fields in each element with a superposition of basis functions, whose derivatives can be calculated exactly. Before introducing FEM, an overview of basis functions is provided below.

## 2.1 Basis Functions

In a two-dimensional numerical problem, the region where the calculations are performed can be discretized into small elements, usually polygons with the same number of sides, generating a mesh. Triangles are most often used for this purpose for their ability to approximate

any planar shape with good accuracy, as shown in Figure 2.1 above. An example of a triangular element is presented in Figure 2.2 below.

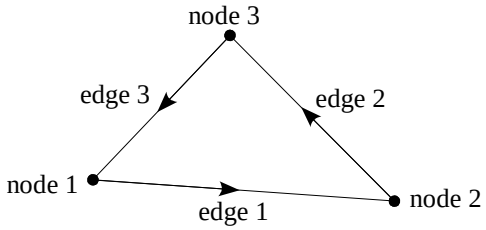



Figure 2.2: Triangular element.

In each element, a scalar field can be approximately expanded with basis functions: $E(x,y) \approx \sum_i E_i N_i(x,y)$, where $E_i$ is the value of the field at vertex, or node, $i$, and $N_i$ is the basis function associated with that node. Therefore, to satisfy $E = E_i$ at node $i$, the basis function must have the value $N_i = 1$ at node $i$ and $N_i = 0$ at the other nodes. In the rest of the element, the basis functions can be defined by interpolation polynomials. For a linear interpolation in a triangular element, the basis functions are [6]

$$N_i(x,y) = \frac{a_i + b_i x + c_i y}{2\Delta} \tag{2.14}$$

where

$$a_i = x_{i+1} y_{i+2} - x_{i+2} y_{i+1} \tag{2.15}$$

$$b_i = y_{i+1} - y_{i+2} \tag{2.16}$$

$$c_i = x_{i+2} - x_{i+1} \tag{2.17}$$

$i = 1, 2, 3$, and $\Delta$ is the area of the triangle, given by

$$\Delta = \frac{b_i c_{i+1} - b_{i+1} c_i}{2}. \tag{2.18}$$

In this notation, whenever the subscripts $i+1$ and $i+2$ result in a value higher than 3, the sum is decreased by 3. For example, if $i = 2$, $x_{i+1}$ represents $x_3$, and $x_{i+2}$ represents $x_1$.

It can be observed that Equation (2.14) satisfies the desired property of a basis function, as $N_i(x_i, y_i) = 1$ and $N_i(x_{i+1}, y_{i+1}) = N_i(x_{i+2}, y_{i+2}) = 0$. Also, Equation (2.18) results in the same value for any $i$.

A vector field can also be expanded with basis functions: $\mathbf{E}(x,y) \approx \sum_i E_i \mathbf{N}_i(x,y)$, where

$E_i$ is the magnitude of the tangential component of the field at side, or edge, $i$, and $\mathbf{N}_i$ is the basis function associated with that edge. Each vector basis function should have a constant tangential component along its associated edge, have only a normal component along the other edges, and vary linearly from $|\mathbf{N}_i| = 1$ at edge $i$ to $|\mathbf{N}_i| = 0$ at the opposite node. To ensure these properties [6],

$$\mathbf{N}_i(x_i, y_i) = l_i\left(N_i \nabla N_{i+1} - N_{i+1}\nabla N_i\right) \tag{2.19}$$

where $N_i$ is a scalar basis function, and $l_i$ is the length of edge $i$,

$$l_i = \sqrt{\left(x_{i+1} - x_i\right)^2 + \left(y_{i+1} - y_i\right)^2} = \sqrt{b_{i+2}^2 + c_{i+2}^2}. \tag{2.20}$$

For $N_i$ defined in (2.14), (2.19) reduces to

$$\mathbf{N}_i(x, y) = \frac{l_i}{2\Delta}\left[\hat{x}\left(y_{i+2} - y\right) + \hat{y}\left(x - x_{i+2}\right)\right]. \tag{2.21}$$

Along edge $i$, there is only one point where $|E| = E_i$. It is the point where the basis function is completely tangential to its edge. To be at edge $i$, the point must satisfy

$$\frac{y - y_i}{y_{i+1} - y_i} = \frac{x - x_i}{x_{i+1} - x_i} \tag{2.22}$$

and for $\mathbf{N}_i$ to be tangential to the edge, the point must also satisfy

$$\left(y_{i+2} - y\right)\left(y_{i+1} - y_i\right) = \left(x - x_{i+2}\right)\left(x_{i+1} - x_i\right). \tag{2.23}$$

The solution of these two equations is

$$x = x_{i+2} - \frac{2\Delta\, b_{i+2}}{l_i^2} \tag{2.24}$$

$$y = y_{i+2} - \frac{2\Delta\, c_{i+2}}{l_i^2}. \tag{2.25}$$

A line segment from this point to the opposite node is perpendicular to the edge, as shown in Figure 2.3 below.

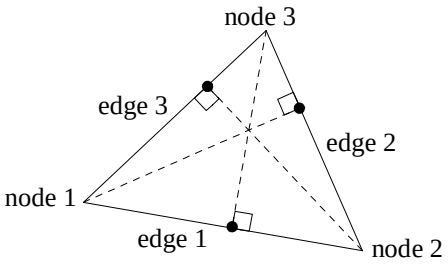



Figure 2.3: Points where the basis functions are tangential to their edges.

## 2.2 Finite Element Method

Many electromagnetic problems involve the solution of the first two Maxwell's equations. Using the constitutive relations,

$$\mathbf{D} = \varepsilon \mathbf{E} \tag{2.26}$$

$$\mathbf{B} = \mu \mathbf{H} \tag{2.27}$$

$$\mathbf{J} = \sigma \mathbf{E} \tag{2.28}$$

and adding another term **J** for a current source, first two Maxwell's equations can be written as

$$\varepsilon \frac{\partial \mathbf{E}}{\partial t} = \nabla \times \mathbf{H} - \sigma \mathbf{E} - \mathbf{J} \tag{2.29}$$

$$\mu \frac{\partial \mathbf{H}}{\partial t} = -\nabla \times \mathbf{E} \tag{2.30}$$

where $\varepsilon$ is the electric permittivity, $\mu$ is the magnetic permeability, and $\sigma$ is the electric conductivity. This set of equations has two unknown functions, the electric and magnetic fields. To discretize them, the fields and sources in each element can be expanded in terms of basis functions: $\mathbf{E} \approx \sum_j E_j \mathbf{N}_{Ej}$, $\mathbf{H} \approx \sum_j H_j \mathbf{N}_{Hj}$, $\mathbf{J} \approx \sum_j J_j \mathbf{N}_{Ej}$, where $\mathbf{N}_{Ej}$ and $\mathbf{N}_{Hj}$ are basis functions for the electric and magnetic fields, respectively. $\mathbf{N}_{Ej}$ and $\mathbf{N}_{Hj}$ may be different from each other; for example, in a problem where $\mathbf{E} = E_z \hat{z}$ and $\mathbf{H} = H_x \hat{x} + H_y \hat{y}$, the basis functions should be $\mathbf{N}_{Ej} = N_j \hat{z}$ and $\mathbf{N}_{Hj} = \mathbf{N}_j$, where $N_j$ is a scalar basis functions associated with a node, and $\mathbf{N}_j$ is a vector basis function associated with an edge, as explained in the previous section.

Applying the basis function expansion, the equations for the fields at each element become

$$\left(\varepsilon \frac{\partial}{\partial t} + \sigma\right) \sum_j E_j \mathbf{N}_{Ej} + \sum_j J_j \mathbf{N}_{Ej} - \nabla \times \left(\sum_j H_j \mathbf{N}_{Hj}\right) \approx \mathbf{r}_E \tag{2.31}$$

$$\mu \frac{\partial}{\partial t} \sum_j H_j \mathbf{N}_{Hj} + \nabla \times \left(\sum_j E_j \mathbf{N}_{Ej}\right) \approx \mathbf{r}_H \tag{2.32}$$

where $\mathbf{r}_E$ and $\mathbf{r}_H$ are the residuals due to the approximation. The Galerkin method in FEM consists of minimizing the weighted residual, which is the product of the residual by a weighting or test function, integrated over the entire element. The test functions are chosen to be the same as the basis functions. Thus, Equations (2.31) and (2.32) are multiplied by a basis function and integrated, and since vector functions are used in this case, the multiplication takes the form of a scalar product:

$$\int_{\Omega}\left[\left(\varepsilon\frac{\partial}{\partial t}+\sigma\right)\sum_j E_j\mathbf{N}_{Ej}+\sum_j J_j\mathbf{N}_{Ej}-\nabla\times\left(\sum_j H_j\mathbf{N}_{Hj}\right)\right]\cdot\mathbf{N}_{Ei}\,d\Omega=\int_{\Omega}\mathbf{r}_E\cdot\mathbf{N}_{Ei}\,d\Omega=R_{Ei} \tag{2.33}$$

$$\int_{\Omega}\left[\mu\frac{\partial}{\partial t}\sum_j H_j\mathbf{N}_{Hj}+\nabla\times\left(\sum_j E_j\mathbf{N}_{Ej}\right)\right]\cdot\mathbf{N}_{Hi}\,d\Omega=\int_{\Omega}\mathbf{r}_H\cdot\mathbf{N}_{Hi}\,d\Omega=R_{Hi} \tag{2.34}$$

where $R_{Ei}$ and $R_{Hi}$ are the weighted residuals associated with node or edge *i* of the element, and $\Omega$ represents the area of the element. With some substitutions, a set of matrix equations can be formed:

$$\left(\varepsilon\frac{\partial}{\partial t}+\sigma\right)[M_E]\{E\}+[M_E]\{J\}-[S_E]\{H\}=\{R_E\} \tag{2.35}$$

$$\mu\frac{\partial}{\partial t}[M_H]\{H\}+[S_H]\{H\}=\{R_H\} \tag{2.36}$$

where $\{E\},\{H\},\{J\},\{R_E\}$ and $\{R_H\}$ are vectors containing the values of the electric and magnetic fields, the electric current source and the weighted residuals, respectively, $[M_E]$ and $[M_H]$ are called mass matrices, and $[S_E]$ and $[S_H]$ are called stiffness matrices, whose elements are given by

$$M_{Eij}=\int_{\Omega}\mathbf{N}_{Ei}\cdot\mathbf{N}_{Ej}\;d\Omega \tag{2.37}$$

$$S_{Eij}=\int_{\Omega}\left(\nabla\times\mathbf{N}_{Hj}\right)\cdot\mathbf{N}_{Ei}\,d\Omega \tag{2.38}$$

$$M_{Hij}=\int_{\Omega}\mathbf{N}_{Hi}\cdot\mathbf{N}_{Hj}\;d\Omega \tag{2.39}$$

$$S_{Hij}=\int_{\Omega}\left(\nabla\times\mathbf{N}_{Ej}\right)\cdot\mathbf{N}_{Hi}\,d\Omega\;. \tag{2.40}$$

It is important to note that the pair of Equations (2.35) and (2.36) above refers to each element separately, so $\{R_E\}$ and $\{R_H\}$ are not exactly the residuals to be minimized. Since most nodes and edges belong to more than one element, the sum of all residuals referring to a node or edge should be added, and this sum is the one to be minimized. This addition can be accomplished by combining the pairs of matrix equations for each element into one pair of matrix equations for the entire region of interest. In this combined system, the weighted residuals can be set to zero:

$$\left(\varepsilon\frac{\partial}{\partial t}+\sigma\right)[M_E]\{E\}+[M_E]\{J\}-[S_E]\{H\}=0 \tag{2.41}$$

$$\mu\frac{\partial}{\partial t}[M_H]\{H\}+[S_H]\{H\}=0 \tag{2.42}$$

which can be solved with matrix inversion. With this combination of matrices, the whole system reduces to one pair of matrix equations, but the order of matrices and vectors becomes very large,

as this order is the number of nodes or edges in the entire region of interest, not in just one element. However, since each node or edge belongs to only a few elements, the resulting matrices are sparse, meaning that most values in the matrices are zero.

Hence, the FEM formulation usually generates very large and sparse matrices. Although many computational methods exist to deal efficiently with sparse matrices, their large orders usually result in large memory and time requirements. For example, for a matrix of order $n$, the memory required to store it is proportional to $n^2$, and standard algorithms to invert matrices take a computation time proportional to $n^3$.

The discontinuous Galerkin method (DGM) is essentially a modification of the Galerkin method in FEM. The main advantage of DGM over FEM is that it solves the matrix equations separately for each element, similarly to (2.35) and (2.36). Thus, DGM requires lower computation costs, as the matrices involved in the calculations have low orders. The details of this method are explained in the next chapter.

## 3. DISCONTINUOUS GALERKIN METHOD

To solve a system of differential equations with DGM, it must be first written in a conservation form, meaning that the sum of all changes equals zero:

$$\mathrm{Q}\frac{\partial \mathbf{q}}{\partial t}+\nabla\cdot\mathbf{F}(\mathbf{q})-\mathbf{S}=0. \tag{3.1}$$

Maxwell's equations fit this form with the following substitutions:

$$\mathrm{Q}=\begin{bmatrix}\varepsilon & 0\\ 0 & \mu\end{bmatrix},\ \mathbf{q}=\begin{bmatrix}\mathbf{E}\\ \mathbf{H}\end{bmatrix},\ \mathbf{S}=\begin{bmatrix}-\sigma\mathbf{E}-\mathbf{J}\\ 0\end{bmatrix} \tag{3.2}$$

$$\nabla\cdot\mathbf{F}(\mathbf{q})=\begin{bmatrix}-\nabla\times\mathbf{H}\\ \nabla\times\mathbf{E}\end{bmatrix}\ \therefore\ \mathbf{F}(\mathbf{q})=\begin{bmatrix}-\hat{x}\times\mathbf{H}\\ \hat{x}\times\mathbf{E}\end{bmatrix}\hat{x}+\begin{bmatrix}-\hat{y}\times\mathbf{H}\\ \hat{y}\times\mathbf{E}\end{bmatrix}\hat{y}+\begin{bmatrix}-\hat{z}\times\mathbf{H}\\ \hat{z}\times\mathbf{E}\end{bmatrix}\hat{z}. \tag{3.3}$$

The fields and sources in each element are approximated with a basis function expansion: $\mathbf{E}\approx\sum_j E_j\mathbf{N}_{Ej}$,$\mathbf{H}\approx\sum_j H_j\mathbf{N}_{Hj}$, $\mathbf{J}\approx\sum_j J_j\mathbf{N}_{Ej}$. As in FEM, this approximation creates a residual:

$$\mathrm{Q}\frac{\partial \mathbf{q}}{\partial t}+\nabla\cdot\mathbf{F}(\mathbf{q})-\mathbf{S}=\mathbf{r}. \tag{3.4}$$

Instead of minimizing the residuals, as in the Galerkin method in FEM, DGM calculates the fields separately in each element and uses the discontinuity of fields between adjacent elements:

$$\int_\Omega\left(\mathrm{Q}\frac{\partial \mathbf{q}}{\partial t}+\nabla\cdot\mathbf{F}(\mathbf{q})-\mathbf{S}\right)\cdot\mathbf{N}_i\,d\Omega=\oint_\Gamma\hat{n}\cdot\left(\mathbf{F}-\mathbf{F}^*\right)\cdot\mathbf{N}_i\,d\Gamma \tag{3.5}$$

where

$$\mathbf{N}_i=\begin{bmatrix}\mathbf{N}_{Ei}\\ \mathbf{N}_{Hi}\end{bmatrix} \tag{3.6}$$

$\Gamma$ is the contour that defines the area $\Omega$ of the element, $\hat{n}$ is a unit vector normal to $\Gamma$, and $\mathbf{F}-\mathbf{F}^*$ is called the *numerical flux*. Only its normal component $\hat{n}\cdot\left(\mathbf{F}-\mathbf{F}^*\right)$ is used in this method, so for simplicity the name *numerical flux* can also be used to refer to this component.

The specification of $\mathbf{F}^*$ starts with the Rankine-Hugoniot condition from the theory of Riemann solvers [2]:

$$-\Lambda_i\mathrm{Q}\left(\mathbf{q}^-\mathbf{-q}^+\right)+\left(\Pi\mathbf{q}\right)^-\mathbf{-}\left(\Pi\mathbf{q}\right)^+=0 \tag{3.7}$$

where $\Pi\mathbf{q}=\hat{n}\cdot\mathbf{F}$, and $\Lambda_i$ is an eigenvalue of $\mathrm{Q}^{-1}\Pi$ ($\Lambda_i$ is actually a matrix and not a scalar eigenvalue, since $\mathrm{Q}^{-1}\Pi$ is a block matrix). The superscripts indicate the two sides of the boundary $\Gamma$: $\mathbf{q}^-$ represents the fields at the boundary, within the element defined by $\Gamma$; $\mathbf{q}^+$

represents the fields at the same boundary, but within an adjacent element, as shown in Figure 3.1 below. The same notation can be used for $\varepsilon$ and $\mu$.

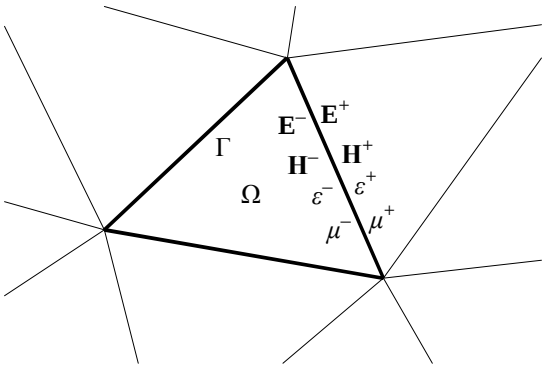



Figure 3.1: Fields at adjacent elements.

Assuming that $\Lambda_i$ may have three values, $-\Lambda$, $\Lambda$ or 0, Equation (3.7) can be generalized with intermediate states, represented by the superscripts * and **:

$$(\Lambda Q)^- (\mathbf{q}^* - \mathbf{q}^-) + (\Pi \mathbf{q})^* - (\Pi \mathbf{q})^- = 0 \tag{3.8}$$

$$(\Pi \mathbf{q})^* - (\Pi \mathbf{q})^{**} = 0 \tag{3.9}$$

$$(-\Lambda Q)^+ (\mathbf{q}^{**} - \mathbf{q}^+) + (\Pi \mathbf{q})^{**} - (\Pi \mathbf{q})^+ = 0 \tag{3.10}$$

which reduce to

$$(\Lambda Q)^- (\mathbf{q}^* - \mathbf{q}^-) + (\Pi \mathbf{q})^* - (\Pi \mathbf{q})^- = 0 \tag{3.11}$$

$$(-\Lambda Q)^+ (\mathbf{q}^* - \mathbf{q}^+) + (\Pi \mathbf{q})^* - (\Pi \mathbf{q})^+ = 0. \tag{3.12}$$

The numerical flux is obtained with the relation $\hat{n} \cdot \mathbf{F}^* = (\Pi \mathbf{q})^*$, from the definition of $\Pi$.

To find the numerical flux for Maxwell's equations, first the matrix $\Pi$ and the eigenvalues $\Lambda_i$ must be found using the variables **F**, Q and **q** defined in Equation (3.2), before the system (3.11–3.12) can be solved. Since the matrix $\Pi$ is defined by $\Pi \mathbf{q} = \hat{n} \cdot \mathbf{F}$, it can be expressed by

$$\Pi = \begin{bmatrix} 0 & -\mathrm{N} \\ \mathrm{N} & 0 \end{bmatrix} \tag{3.13}$$

where $\mathrm{N}\mathbf{E} = \hat{n} \times \mathbf{E}$, $\mathrm{N}\mathbf{H} = \hat{n} \times \mathbf{H}$. Using the notation $\mathbf{E} = \begin{bmatrix} E_x \\ E_y \\ E_z \end{bmatrix}$, $\mathbf{H} = \begin{bmatrix} H_x \\ H_y \\ H_z \end{bmatrix}$, N can be expressed by

$$\mathrm{N} = \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix} \tag{3.14}$$

where $n_x$, $n_y$ and $n_z$ are the components of $\hat{n}$. It is important to observe that N is singular, antisymmetric, and that $\mathrm{N}^3 = -\mathrm{N}$. The matrix $\mathrm{Q}^{-1}\Pi$ is

$$\mathrm{Q}^{-1}\Pi = \frac{1}{\mu\varepsilon}\begin{bmatrix} \mu & 0 \\ 0 & \varepsilon \end{bmatrix}\begin{bmatrix} 0 & -\mathrm{N} \\ \mathrm{N} & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\mathrm{N}/\varepsilon \\ \mathrm{N}/\mu & 0 \end{bmatrix} \tag{3.15}$$

and its eigenvalues can thus be found with the following equation:

$$\det\left(\Lambda_i^2 + \frac{\mathrm{N}^2}{\mu\varepsilon}\right) = 0. \tag{3.16}$$

One solution is $\Lambda_i = 0$. Other solutions can be found by

$$\Lambda_i^2 = -\frac{\mathrm{N}^2}{\mu\varepsilon} = \frac{\mathrm{N}^4}{\mu\varepsilon} \;\therefore\; \Lambda_i = \pm c\mathrm{N}^2 \tag{3.17}$$

where $c = \dfrac{1}{\sqrt{\mu\varepsilon}}$. Therefore, in Equations (3.8) to (3.12), $\Lambda = c\mathrm{N}^2$. Solving (3.11) and (3.12), we have

$$\left(c^+\mathrm{Q}^+ + c^-\mathrm{Q}^-\right)\mathrm{N}^2\left(\Pi\mathbf{q}\right)^* = c^+\mathrm{Q}^+\mathrm{N}^2\left(\Pi\mathbf{q}\right)^- + c^-\mathrm{Q}^-\mathrm{N}^2\left(\Pi\mathbf{q}\right)^+ + c^+\mathrm{Q}^+c^-\mathrm{Q}^-\mathrm{N}^4\left(\mathbf{q}^- - \mathbf{q}^+\right). \tag{3.18}$$

In the first three terms of (3.18), the vectors $\mathbf{q}$ are multiplied by $\Pi$ and $\mathrm{N}^2$, so in effect $\mathrm{N}^3$ multiplies $\mathbf{E}$ and $\mathbf{H}$, while in the fourth term the fields are multiplied by $\mathrm{N}^4$. Although N has no inverse, the factor $\mathrm{N}^2$ can be removed from all four terms in the equation because N is antisymmetric. Defining $Z = \sqrt{\mu/\varepsilon}$ and $Y = \sqrt{\varepsilon/\mu}$, Equation (3.18) can be written as

$$\begin{bmatrix} Y^+ + Y^- & 0 \\ 0 & Z^+ + Z^- \end{bmatrix}\hat{n}\cdot\mathbf{F}^* = \begin{bmatrix} Y^+ & 0 \\ 0 & Z^+ \end{bmatrix}\begin{bmatrix} -\hat{n}\times\mathbf{H}^- \\ \hat{n}\times\mathbf{E}^- \end{bmatrix} + \begin{bmatrix} Y^- & 0 \\ 0 & Z^- \end{bmatrix}\begin{bmatrix} -\hat{n}\times\mathbf{H}^+ \\ \hat{n}\times\mathbf{E}^+ \end{bmatrix} + \begin{bmatrix} Y^+Y^- & 0 \\ 0 & Z^+Z^- \end{bmatrix}\begin{bmatrix} \hat{n}\times\hat{n}\times\left(\mathbf{E}^- - \mathbf{E}^+\right) \\ \hat{n}\times\hat{n}\times\left(\mathbf{H}^- - \mathbf{H}^+\right) \end{bmatrix}. \tag{3.19}$$

Solving for $\hat{n}\cdot\mathbf{F}^*$ yields

$$\hat{n}\cdot\mathbf{F}^* = \begin{bmatrix} -\hat{n}\times\dfrac{Z^+\mathbf{H}^+ + Z^-\mathbf{H}^- - \hat{n}\times\left(\mathbf{E}^- - \mathbf{E}^+\right)}{Z^+ + Z^-} \\ \hat{n}\times\dfrac{Y^+\mathbf{E}^+ + Y^-\mathbf{E}^- + \hat{n}\times\left(\mathbf{H}^- - \mathbf{H}^+\right)}{Y^+ + Y^-} \end{bmatrix} \tag{3.20}$$

which can be used to find the term $\hat{n}\cdot\left(\mathbf{F} - \mathbf{F}^*\right)$ in (3.5):

$$\hat{n}\cdot\left(\mathbf{F}-\mathbf{F}^{*}\right)=\begin{bmatrix}-\hat{n}\times\mathbf{H}^{-}\\ \hat{n}\times\mathbf{E}^{-}\end{bmatrix}-\hat{n}\cdot\mathbf{F}^{*}=\begin{bmatrix}\hat{n}\times\dfrac{Z^{+}\left(\mathbf{H}^{+}-\mathbf{H}^{-}\right)-\hat{n}\times\left(\mathbf{E}^{+}-\mathbf{E}^{-}\right)}{Z^{+}+Z^{-}}\\ -\hat{n}\times\dfrac{Y^{+}\left(\mathbf{E}^{+}-\mathbf{E}^{-}\right)+\hat{n}\times\left(\mathbf{H}^{+}-\mathbf{H}^{-}\right)}{Y^{+}+Y^{-}}\end{bmatrix}=\begin{bmatrix}\mathbf{G}_{E}\\ \mathbf{G}_{H}\end{bmatrix}. \tag{3.21}$$

With the numerical flux defined, the system in (3.5) can be separated into two equations:

$$\int_{\Omega}\left[\left(\varepsilon\frac{\partial}{\partial t}+\sigma\right)\mathbf{E}+\mathbf{J}-\nabla\times\mathbf{H}\right]\cdot\mathbf{N}_{Ei}\,d\Omega=\oint_{\Gamma}\mathbf{G}_{E}\cdot\mathbf{N}_{Ei}\,d\Gamma \tag{3.22}$$

$$\int_{\Omega}\left[\mu\frac{\partial\mathbf{H}}{\partial t}+\nabla\times\mathbf{E}\right]\cdot\mathbf{N}_{Hi}\,d\Omega=\oint_{\Gamma}\mathbf{G}_{H}\cdot\mathbf{N}_{Hi}\,d\Gamma \tag{3.23}$$

with $\mathbf{G}_{E}$ and $\mathbf{G}_{H}$ defined in (3.21). Applying the basis function expansion to (3.22), we have

$$\int_{\Omega}\left[\left(\varepsilon\frac{\partial}{\partial t}+\sigma\right)\sum_{j}E_{j}\mathbf{N}_{Ej}+\sum_{j}J_{j}\mathbf{N}_{Ej}-\nabla\times\left(\sum_{j}H_{j}\mathbf{N}_{Hj}\right)\right]\cdot\mathbf{N}_{Ei}\,d\Omega=\oint_{\Gamma}\mathbf{G}_{E}\cdot\mathbf{N}_{Ei}\,d\Gamma$$

$$\sum_{j}\left[\left(\varepsilon\frac{\partial}{\partial t}+\sigma\right)E_{j}+J_{j}\right]\int_{\Omega}\mathbf{N}_{Ej}\cdot\mathbf{N}_{Ei}\,d\Omega=\sum_{j}H_{j}\int_{\Omega}\left(\nabla\times\mathbf{N}_{Hj}\right)\cdot\mathbf{N}_{Ei}\,d\Omega+\oint_{\Gamma}\mathbf{G}_{E}\cdot\mathbf{N}_{Ei}\,d\Gamma$$

$$\left(\varepsilon\frac{\partial}{\partial t}+\sigma\right)\sum_{j}M_{Eij}E_{j}+\sum_{j}M_{Eij}J_{j}=\sum_{j}S_{Eij}H_{j}+F_{Ei} \tag{3.24}$$

where

$$M_{Eij}=\int_{\Omega}\mathbf{N}_{Ei}\cdot\mathbf{N}_{Ej}\,d\Omega \tag{3.25}$$

$$S_{Eij}=\int_{\Omega}\left(\nabla\times\mathbf{N}_{Hj}\right)\cdot\mathbf{N}_{Ei}\,d\Omega \tag{3.26}$$

$$F_{Ei}=\oint_{\Gamma}\mathbf{G}_{E}\cdot\mathbf{N}_{Ei}\,d\Gamma\ . \tag{3.27}$$

Equation (3.24) can also be written in matrix form:

$$\left(\varepsilon\frac{\partial}{\partial t}+\sigma\right)\left[M_{E}\right]\{E\}+\left[M_{E}\right]\{J\}=\left[S_{E}\right]\{H\}+\{F_{E}\}$$

$$\frac{\partial\{E\}}{\partial t}=\frac{\left[M_{E}\right]^{-1}\left(\left[S_{E}\right]\{H\}+\{F_{E}\}\right)-\sigma\{E\}-\{J\}}{\varepsilon} \tag{3.28}$$

where $\left[M_{E}\right]$ is called the mass matrix, $\left[S_{E}\right]$ is called the stiffness matrix, and $\{F_{E}\}$ is the numerical flux vector. Similarly, (3.23) becomes

$$\mu\frac{\partial}{\partial t}\left[M_{H}\right]\{H\}=-\left[S_{H}\right]\{E\}+\{F_{H}\}$$

$$\frac{\partial\{H\}}{\partial t}=\frac{\left[M_{H}\right]^{-1}\left(-\left[S_{H}\right]\{E\}+\{F_{H}\}\right)}{\mu} \tag{3.29}$$

where the elements of $\left[M_{H}\right]$, $\left[S_{H}\right]$ and $\{F_{H}\}$ are similar to (3.25–3.27), but with the subscripts *E*

and *H* switched.

The evaluation of the mass and stiffness matrices and the numerical flux vector depends on the choice of basis functions. Two cases are considered at the end of this chapter: transverse magnetic (TM) and transverse electric (TE), in two dimensions.

Equations (3.28) and (3.29) calculate the fields in each element separately, and the numerical flux accounts for the differences in the fields at adjacent elements. Thus, in DGM, a set of matrix equations must be solved for each element, as opposed to one set for the entire system in FEM. However, the order of matrices and vectors used in DGM is much smaller, since this order is the number of nodes or edges in one element, not in the entire region of interest. This result is the main advantage of DGM: both the memory required to store all values and the time to perform all computations vary linearly with the number of elements.

Another important observation is that, since the fields are calculated separately in each element, more than one value is obtained for the fields at the boundaries between elements. In other words, the calculated fields are discontinuous, and this aspect is reflected in the name of the method.

### 3.1 Transverse Magnetic

In a two-dimensional transverse magnetic (TM) problem, the basis functions are $\mathbf{N}_{Ei} = N_i \hat{z}$ and $\mathbf{N}_{Hi} = \mathbf{N}_i$, where $N_i$ is a scalar basis function and $\mathbf{N}_i$ is a vector basis function. For triangular elements with linear interpolation, these basis functions are defined in (2.14–2.21). Using this discretization, the elements of the mass and stiffness matrices and of the numerical flux vector are evaluated as

$$M_{Eij} = \int_\Omega N_i N_j \, d\Omega = \frac{\Delta}{12}\left(1+\delta_{i,j}\right) \tag{3.30}$$

$$S_{Eij} = \int_\Omega \left(\nabla \times \mathbf{N}_j\right) \cdot N_i \hat{z} \, d\Omega = \int_\Omega N_i \left(\frac{\partial N_{jy}}{\partial x} - \frac{\partial N_{jx}}{\partial y}\right) d\Omega = \int_\Omega N_i \left(\frac{l_j}{2\Delta} + \frac{l_j}{2\Delta}\right) d\Omega = \frac{l_j}{3} \tag{3.31}$$

$$F_{Ei} = \oint_\Gamma \mathbf{G}_E \cdot N_i \hat{z} \, d\Gamma = \oint_\Gamma \hat{n} \times \frac{Z^+\left(\mathbf{H}^+ - \mathbf{H}^-\right) - \hat{n} \times \left(\mathbf{E}^+ - \mathbf{E}^-\right)}{Z^+ + Z^-} \cdot N_i \hat{z} \, d\Gamma$$

$$= \sum\nolimits_j \left[ \frac{Z^+\left(H_{op,j} - H_j\right)}{Z^+ + Z^-} \int_{\Gamma_j} N_i \, d\Gamma + \frac{E_{op,j} - E_j}{Z^+ + Z^-} \int_{\Gamma_j} N_i N_j \, d\Gamma + \frac{E_{op,j+1} - E_{j+1}}{Z^+ + Z^-} \int_{\Gamma_j} N_i N_{j+1} \, d\Gamma \right]$$

$$= \sum_j \left[ 3Z^+ \left( H_{op,j} - H_j \right) + \left( E_{op,j} - E_j \right) \left( 1 + \delta_{i,j} \right) + \left( E_{op,j+1} - E_{j+1} \right) \left( 1 + \delta_{i,j+1} \right) \right] \frac{l_j \left( 1 - \delta_{i,j+2} \right)}{6 \left( Z^+ + Z^- \right)} \tag{3.32}$$

$$M_{Hij} = \int_\Omega \mathbf{N}_i \cdot \mathbf{N}_j \, d\Omega = \int_\Omega \left[ l_i \left( \frac{c_{i+1}}{2\Delta} N_i - \frac{c_i}{2\Delta} N_{i+1} \right) l_j \left( \frac{c_{j+1}}{2\Delta} N_j - \frac{c_j}{2\Delta} N_{j+1} \right) + l_i \left( \frac{b_{i+1}}{2\Delta} N_i - \frac{b_i}{2\Delta} N_{i+1} \right) l_j \left( \frac{b_{j+1}}{2\Delta} N_j - \frac{b_j}{2\Delta} N_{j+1} \right) \right] d\Omega$$

$$= \frac{l_i l_j}{48\Delta} \left[ \left( f_{i,j} + f_{i+1,j+1} \right) \left( 1 + \delta_{i,j} \right) - f_{i+1,j} \left( 1 + \delta_{i,j+1} \right) - f_{i,j+1} \left( 1 + \delta_{i,j+2} \right) \right] \tag{3.33}$$

$$S_{Hij} = \int_\Omega \left( \nabla \times N_j \, \hat{z} \right) \cdot \mathbf{N}_i \, d\Omega = \int_\Omega \left( N_{ix} \frac{\partial N_j}{\partial y} - N_{iy} \frac{\partial N_j}{\partial x} \right) d\Omega$$

$$= \int_\Omega \left[ l_i \left( \frac{b_{i+1}}{2\Delta} N_i - \frac{b_i}{2\Delta} N_{i+1} \right) \frac{c_j}{2\Delta} - l_i \left( \frac{c_{i+1}}{2\Delta} N_i - \frac{c_i}{2\Delta} N_{i+1} \right) \frac{b_j}{2\Delta} \right] d\Omega = \frac{l_i}{6} \left( 3\delta_{i,j+1} - 1 \right) \tag{3.34}$$

$$F_{Hi} = \oint_\Gamma \mathbf{G}_H \cdot \mathbf{N}_i \, d\Gamma = \oint_\Gamma -\hat{n} \times \frac{Y^+ \left( \mathbf{E}^+ - \mathbf{E}^- \right) + \hat{n} \times \left( \mathbf{H}^+ - \mathbf{H}^- \right)}{Y^+ + Y^-} \cdot \mathbf{N}_i \, d\Gamma$$

$$= \sum_j \left[ \frac{H_{op,j} - H_j}{Y^+ + Y^-} \int_{\Gamma_j} d\Gamma + \frac{Y^+ \left( E_{op,j} - E_j \right)}{Y^+ + Y^-} \int_{\Gamma_j} N_j \, d\Gamma + \frac{Y^+ \left( E_{op,j+1} - E_{j+1} \right)}{Y^+ + Y^-} \int_{\Gamma_j} N_{j+1} \, d\Gamma \right] \delta_{i,j}$$

$$= \sum_j \left[ 2 \left( H_{op,j} - H_j \right) + Y^+ \left( E_{op,j} - E_j + E_{op,j+1} - E_{j+1} \right) \right] \frac{l_j \delta_{i,j}}{2 \left( Y^+ + Y^- \right)} \, . \tag{3.35}$$

In the expressions above, $i, j \in \{1, 2, 3\}$, and whenever a sum involving one of these indices results in a value higher than 3, the result is decreased by 3. The subscript *op,j* refers to the field at the element opposite to edge *j*. The binary function $\delta_{i,j}$ is defined by

$$\delta_{i,j} = \begin{cases} 0 \text{ , if } i \neq j \\ 1 \text{ , if } i = j \end{cases} \tag{3.36}$$

and the function $f_{i,j}$ is defined by

$$f_{i,j} = b_i b_j + c_i c_j. \tag{3.37}$$

### 3.2 Transverse Electric

In a two-dimensional transverse electric (TE) problem, the basis functions are $\mathbf{N}_{Ei} = \mathbf{N}_i$

and $\mathbf{N}_{Hi} = N_i\,\hat{z}$. The mass and stiffness matrices in this case are very similar to the ones in the TM case, but with the subscripts *E* and *H* switched. The numerical flux vector is also similar to the previous case, as shown below.

$$M_{Eij} = \int_\Omega \mathbf{N}_i \cdot \mathbf{N}_j \, d\Omega = \frac{l_i l_j}{48\Delta}\Big[\big(f_{i,j} + f_{i+1,j+1}\big)\big(1+\delta_{i,j}\big) - f_{i+1,j}\big(1+\delta_{i,j+1}\big) - f_{i,j+1}\big(1+\delta_{i,j+2}\big)\Big] \tag{3.38}$$

$$S_{Eij} = \int_\Omega \big(\nabla \times N_j\,\hat{z}\big)\cdot \mathbf{N}_i \, d\Omega = \frac{l_i}{6}\big(3\delta_{i,j+1} - 1\big) \tag{3.39}$$

$$\begin{aligned} F_{Ei} &= \oint_\Gamma \mathbf{G}_E \cdot \mathbf{N}_i \; d\Gamma \; = \int_\Gamma \hat{n}\times \frac{Z^+\big(\mathbf{H}^+ - \mathbf{H}^-\big) - \hat{n}\times\big(\mathbf{E}^+ - \mathbf{E}^-\big)}{Z^+ + Z^-}\cdot \mathbf{N}_i \; d\Gamma \\ &= \sum\nolimits_j \left[ \frac{E_{op,j} - E_j}{Z^+ + Z^-} \int_{\Gamma_j} d\Gamma - \frac{Z^+\big(H_{op,j} - H_j\big)}{Z^+ + Z^-} \int_{\Gamma_j} N_j \, d\Gamma - \frac{Z^+\big(H_{op,j+1} - H_{j+1}\big)}{Z^+ + Z^-} \int_{\Gamma_j} N_{j+1} d\Gamma \right] \delta_{i,j} \\ &= \sum\nolimits_j \Big[ 2\big(E_{op,j} - E_j\big) - Z^+\big(H_{op,j} - H_j + H_{op,j+1} - H_{j+1}\big)\Big] \frac{l_j \delta_{i,j}}{2\big(Z^+ + Z^-\big)} \end{aligned} \tag{3.40}$$

$$M_{Hij} = \int_\Omega N_i N_j \, d\Omega \; = \frac{\Delta}{12}\big(1+\delta_{i,j}\big) \tag{3.41}$$

$$S_{Hij} = \int_\Omega \big(\nabla \times \mathbf{N}_j\big)\cdot N_i\,\hat{z} \; d\Omega = \frac{l_j}{3} \tag{3.42}$$

$$\begin{aligned} F_{Hi} &= \oint_\Gamma \mathbf{G}_H \cdot N_i\,\hat{z} \; d\Gamma = \int_\Gamma -\hat{n}\times \frac{Y^+\big(\mathbf{E}^+ - \mathbf{E}^-\big) + \hat{n}\times\big(\mathbf{H}^+ - \mathbf{H}^-\big)}{Y^+ + Y^-}\cdot N_i\,\hat{z} \; d\Gamma \\ &= \sum\nolimits_j \left[ -\frac{Y^+\big(E_{op,j} - E_j\big)}{Y^+ + Y^-} \int_{\Gamma_j} N_i \, d\Gamma + \frac{H_{op,j} - H_j}{Y^+ + Y^-} \int_{\Gamma_j} N_i N_j \, d\Gamma + \frac{H_{op,j+1} - H_{j+1}}{Y^+ + Y^-} \int_{\Gamma_j} N_i N_{j+1} d\Gamma \right] \\ &= \sum\nolimits_j \Big[ -3Y^+\big(E_{op,j} - E_j\big) + \big(H_{op,j} - H_j\big)\big(1+\delta_{i,j}\big) + \big(H_{op,j+1} - H_{j+1}\big)\big(1+\delta_{i,j+1}\big)\Big] \frac{l_j\big(1-\delta_{i,j+2}\big)}{6\big(Y^+ + Y^-\big)} \; . \end{aligned} \tag{3.43}$$

## 4. TIME DISCRETIZATION

The DGM formulation in the previous chapter only concerns spatial derivatives. The time derivatives in Maxwell's equations can be approximated, for instance, with finite difference schemes, as in FDTD. One of these schemes is called forward difference:

$$\frac{\partial \{E\}_n}{\partial t} \approx \frac{\{E\}_{n+1} - \{E\}_n}{\Delta t} \tag{4.1}$$

where $\{E\}_n$ and $\{E\}_{n+1}$ are the values of the field at times $n$ and $n$+1, respectively, and $\Delta t$ is the difference between these two times, also called the time step. A similar expression can be used for $\{H\}$. Substituting these expressions in Equations (3.28) and (3.29), we obtain

$$\{E\}_{n+1} = \{E\}_n + \frac{\Delta t}{\varepsilon}\left[[M_E]^{-1}\left([S_E]\{H\}_n + \{F_E\}_n\right) - \sigma\{E\}_n - \{J\}_n\right] \tag{4.2}$$

$$\{H\}_{n+1} = \{H\}_n + \frac{\Delta t}{\mu}[M_H]^{-1}\left(-[S_H]\{E\}_n + \{F_H\}_n\right). \tag{4.3}$$

The system above yields the solution of Maxwell's equations through DGM and forward difference. Given initial conditions and sources, the fields at all points in the region of interest and at any later time can be calculated.

### 4.1 Stability

One problem that arises from the discretization of both space and time derivatives is stability. If the relation between $\Delta x$, the distance between nodes, and $\Delta t$, the interval between two time points, is not enough to approximate the real variation of the fields, the calculated values may increase indefinitely with time. Hence, it is necessary to find a stability condition that involves $\Delta x$ and $\Delta t$.

Combining Equations (4.2) and (4.3), and ignoring sources and numerical fluxes, we have

$$\{E\}_{n+2} = \{E\}_{n+1} + \frac{\Delta t}{\varepsilon}[M_E]^{-1}[S_E]\{H\}_n - \frac{(\Delta t)^2}{\mu\varepsilon}[M_E]^{-1}[M_H]^{-1}[S_E][S_H]\{E\}_n. \tag{4.4}$$

The calculated value $\{E\}_{n+2}$ relates to one of its previous values, $\{E\}_n$, through the factor

$\frac{(\Delta t)^2}{\mu\varepsilon}[M_E]^{-1}[M_H]^{-1}[S_E][S_H]$. Using either (3.30–3.35) or (3.38–3.43), and considering the most common element of each matrix, this factor becomes

$$\frac{(\Delta t)^2}{\mu\varepsilon}\frac{12}{\Delta}\frac{48\Delta}{(\Delta x)^4}\frac{\Delta x}{3}\frac{\Delta x}{6}=\left(\frac{c\Delta t}{\Delta x}4\sqrt{2}\right)^2 \tag{4.5}$$

where $c=\frac{1}{\sqrt{\mu\varepsilon}}$, the speed of an electromagnetic wave in the element, and $\Delta x$ represents the length of one of the edges of the element. To ensure stability, the factor cannot be greater than 1:

$$\left(\frac{c\Delta t}{\Delta x}4\sqrt{2}\right)^2 \le 1$$

$$\frac{\Delta x}{\Delta t}\ge 4\sqrt{2}\,c. \tag{4.6}$$

In this derivation, the numerical fluxes were ignored, the edges were considered to have the same length, and the matrices were substituted with their most common elements, so the stability condition in Equation (4.6) is approximate. Therefore, the value of $4\sqrt{2}$ may not be enough to achieve stability, and usually a slightly higher value is used.

### 4.2 Runge-Kutta Method

The simple approximation of time derivatives with forward difference can be easily implemented in a computer program with a condition loop. However, the accuracy obtained with this method is limited because it considers that the fields vary linearly between two consecutive points in time. Therefore, to obtain high accuracy in fields with large time variations, many points in time are required, which results in a large memory needed to store all values as well as a long computation time.

The Runge-Kutta method is also widely used to approximate time derivatives because of it can often achieve the same accuracy as FDTD schemes but with fewer points in time, thus allowing faster computations. The classical version of this method involves calculating four additional values in each approximation, so it is also referred to as the fourth-order Runge-Kutta method (RK4). Considering the partial differential equation

$$\frac{\partial x}{\partial t}=f(x) \tag{4.7}$$

where $x$ is a function of $t$, and discretizing this variable with

$$t_{n+1}=t_n+\Delta t \tag{4.8}$$

RK4 gives a solution of $x$ by

$$x_{n+1} = x_n + \frac{\Delta t}{6}\left(a_n + 2b_n + 2c_n + d_n\right) \tag{4.9}$$

where

$$a_n = f\left(x_n\right) \tag{4.10}$$

$$b_n = f\left(x_n + a_n\, \Delta t/2\right) \tag{4.11}$$

$$c_n = f\left(x_n + b_n \Delta t/2\right) \tag{4.12}$$

$$d_n = f\left(x_n + c_n \Delta t\right). \tag{4.13}$$

A simpler version, which still has a comparable accuracy, is the second-order Runge-Kutta method (RK2):

$$x_{n+1} = x_n + b_n \Delta t \tag{4.14}$$

where

$$a_n = f\left(x_n\right) \tag{4.15}$$

$$b_n = f\left(x_n + a_n \Delta t/2\right). \tag{4.16}$$

To apply RK4 or RK2 to Equations (3.28) and (3.29), the function $x$ can be considered as a vector containing both $\{E\}$ and $\{H\}$.

## 5. IMPLEMENTATION

Before Equations (3.28) and (3.29) can be implemented in a computer code using the finite difference or the Runge-Kutta method for time derivatives, a large number of variables and parameters must be defined.

### 5.1 Discretized Region, Matrices and Vectors

First of all, the region of interest should contain all objects and spaces relevant to the electromagnetic problem to be solved, including sources, scatterers, dielectrics and free space. Once defined, the region is discretized into small elements, usually triangles, such that the boundaries of all objects are approximated with the edges of the elements. Figure 5.1 below shows an example of a discretized region.

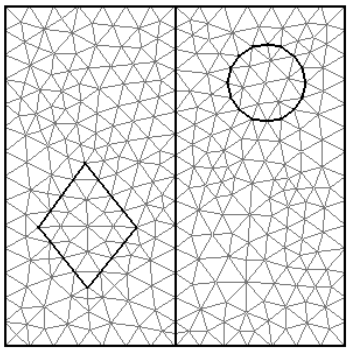
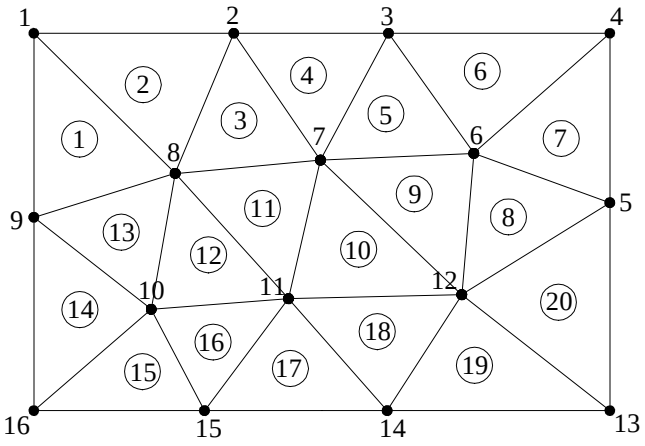


Figure 5.1: Discretized region with objects.

To deal with the discretization efficiently, it is useful to number all elements and nodes uniquely, and to generate two matrices with this information: a *matrix of elements*, listing all elements and their nodes, and a *matrix of nodes*, listing all nodes individually and their coordinates. While the numbering of elements and nodes is arbitrary, in the matrix of elements it is important that the nodes be listed in the same rotation order for every element, clockwise or counterclockwise. In other words, the specific numbers assigned to each element and node are irrelevant, but the order in which the nodes are listed for each element must be consistent. Figure 5.2 below gives an example of how the nodes should be listed.




| element | node 1 | node 2 | node 3 |
|---|---|---|---|
| 1 | 1 | 9 | 8 |
| 2 | 1 | 8 | 2 |
| 3 | 2 | 8 | 7 |
| 4 | 2 | 7 | 3 |
| 5 | 3 | 7 | 6 |
| 6 | 3 | 6 | 4 |
| 7 | 4 | 6 | 5 |
| 8 | 5 | 6 | 12 |
| 9 | 6 | 7 | 12 |
| 10 | 7 | 11 | 12 |
| 11 | 7 | 8 | 11 |
| 12 | 8 | 10 | 11 |
| 13 | 8 | 9 | 10 |
| 14 | 9 | 16 | 10 |
| 15 | 10 | 16 | 15 |
| 16 | 10 | 15 | 11 |
| 17 | 11 | 15 | 14 |
| 18 | 11 | 14 | 12 |
| 19 | 12 | 14 | 13 |
| 20 | 12 | 13 | 2 |

Figure 5.2: Listing of nodes for each element, counterclockwise.

The fields *E* and *H*, as well as the source *J*, can be stored as vectors having three indices: element number, node number in the element, and time point. For example, the electric field at node *i* of element *e*, at time *n*, can be written as $E_{i,n}^{(e)}$.

The material parameters $\varepsilon$, $\mu$ and $\sigma$ are usually considered to be invariant within each element, so they can be stored as vectors with only one index, the element number. Moreover, if the problem only includes one kind of dielectric, or only free space, and any conductors are considered perfect, the material parameters can be stored as constants.

The mass and stiffness matrices can also be stored with three indices: element number *e*, and the row and column numbers *i* and *j*, for instance $M_{E,i,j}^{(e)}$ (the subscript *E* is not an index). Their values can be calculated according to the expressions in Chapter 2, and with the help of the matrix of elements and matrix of nodes, to find the correct indices and coordinate values easily. The mass and stiffness matrices only need to be calculated once for each element, as they do not change with time, so they should be stored for use at every time point.

The calculation of the numerical flux vectors includes the values of the fields and material parameters at adjacent elements, and edge lengths. Since the fields change with time, the numerical flux vectors must also be calculated for each element and at each time point. They do not need to be stored, since they are only used in one time point. However, just for the purpose of notation, they can be written with three indices: element number *e*, row number *i*, and time point

*n*, such as $F_{E,i,n}^{(e)}$. In addition, to avoid recalculating the edge lengths at every time point, it is useful to store them as vectors with two indices: element number *e*, and edge number *i*, as $l_i^{(e)}$.

The fields at adjacent elements must be carefully selected when calculating the numerical flux. Using the notation that includes all indices, as explained above, the expressions in (3.32) and (3.35), for the TM case, are written as

$$F_{E,i,n}^{(e)} = \sum_j \Big[ 3Z^{(op)} \left( -H_{k,n}^{(op)} - H_{j,n}^{(e)} \right) + \left( E_{k+1,n}^{(op)} - E_{j,n}^{(e)} \right) \left( 1 + \delta_{i,j} \right) + \left( E_{k,n}^{(op)} - E_{j+1,n}^{(e)} \right) \left( 1 + \delta_{i,j+1} \right) \Big] \frac{l_j^{(e)} \left( 1 - \delta_{i,j+2} \right)}{6 \left( Z^{(op)} + Z^{(e)} \right)} \tag{5.1}$$

$$F_{H,i,n}^{(e)} = \sum_j \left[ 2 \left( -H_{k,n}^{(op)} - H_{j,n}^{(e)} \right) + Y^{(op)} \left( E_{k+1,n}^{(op)} - E_{j,n}^{(e)} + E_{k,n}^{(op)} - E_{j+1,n}^{(e)} \right) \right] \frac{l_j^{(e)} \delta_{i,j}}{2 \left( Y^{(op)} + Y^{(e)} \right)} \tag{5.2}$$

and those in (3.40) and (3.43), for the TE case, as

$$F_{E,i,n}^{(e)} = \sum_j \left[ 2 \left( -E_{k,n}^{(op)} - E_{j,n}^{(e)} \right) - Z^{(op)} \left( H_{k+1,n}^{(op)} - H_{j,n}^{(e)} + H_{k,n}^{(op)} - H_{j+1,n}^{(e)} \right) \right] \frac{l_j^{(e)} \delta_{i,j}}{2 \left( Z^{(op)} + Z^{(e)} \right)} \tag{5.3}$$

$$F_{H,i,n}^{(e)} = \sum_j \Big[ -3Y^{(op)} \left( -E_{k,n}^{(op)} - E_{j,n}^{(e)} \right) + \left( H_{k+1,n}^{(op)} - H_{j,n}^{(e)} \right) \left( 1 + \delta_{i,j} \right) + \left( H_{k,n}^{(op)} - H_{j+1,n}^{(e)} \right) \left( 1 + \delta_{i,j+1} \right) \Big] \frac{l_j^{(e)} \left( 1 - \delta_{i,j+2} \right)}{6 \left( Y^{(op)} + Y^{(e)} \right)} \ . \tag{5.4}$$

In the expressions above, the index *op* means the element adjacent to element *e*, opposite to edge *j* of element *e*. Edge *k* of element *op* and edge *j* of element *e* are the same edge; node *k*+1 of *op* and node *j* of *e* are the same node; and node *k* of *op* and node *j*+1 of *e* are the same node as well. The negative signs in $-H_{k,n}^{(op)}$ for the TM case and in $-E_{k,n}^{(op)}$ for the TE case are necessary because these fields along the edges are oriented at opposite directions in adjacent elements. Figure 5.3 gives an example of two adjacent elements with all nodes, edges, and fields indexed, for the TM case (for simplicity, the index *n* is omitted in all fields).

To facilitate finding the element opposite to a certain edge, it is useful to create a matrix whose row and column numbers are the unique numbers of the nodes, and the matrix elements are the numbers of the elements that share that edge. For instance, denoting this matrix by A, if the edge between nodes *p* and *q* is shared between elements *e* and *op*, then $\mathrm{A}_{p,q} = \mathrm{A}_{q,p} = (e, op)$. If this edge is at the end of the region of interest, and only belongs to element *e*, a zero can be used to represent the second element. If two nodes do not form an edge, as is the case for most pairs of

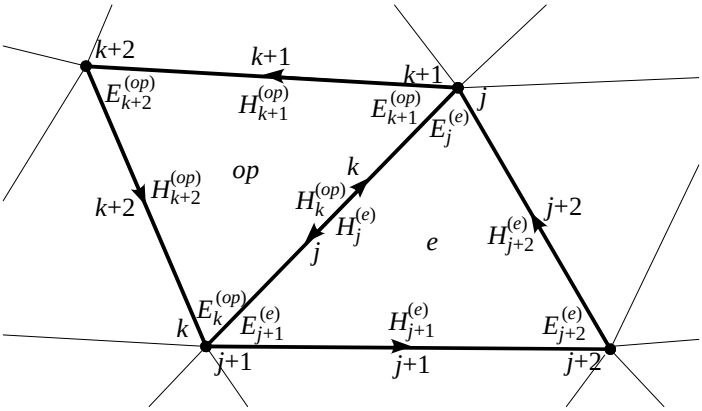


Figure 5.3: Adjacent elements and fields in a TM problem.

nodes, zeros can be used for both elements. When calculating the numerical flux, this matrix can be accessed using the numbers of the two nodes that define a certain edge of the element, returning two values: one is the number of the element itself, the other is the number of the adjacent element opposite to the edge.

## 5.2 Initial Conditions and Sources

Any scheme that approximates time derivatives requires known values of the functions at the first time point. Therefore, the values of $E_{i,0}^{(e)}$ and $H_{i,0}^{(e)}$ for all $e$ and $i$ must be defined, and these initial conditions depend on the type of problem being simulated. In scattering problems, for instance, where an electromagnetic wave is incident on an object, usually the initial values of the fields are set to zero at all points. In a resonance problem with no sources, the initial values may represent the desired modes to be analyzed.

In case the initial field values are not zero, and dependent on position, they must be calculated for specific points in the elements, where the basis functions are maximum. For example, in a TM problem using triangles as elements, $E_z$ should be calculated at the nodes, while $H_x$ and $H_y$ should be calculated at the points given by (2.24) and (2.25). Then the initial fields $E_{i,0}^{(e)}$ and $H_{i,0}^{(e)}$ are implemented with

$$E_{i,0}^{(e)} = E_z \tag{5.5}$$

$$H_{i,0}^{(e)} = \frac{H_x c_{i+2} - H_y b_{i+2}}{l_i}. \tag{5.6}$$

In a TE problem, where the initially set fields are $E_x$, $E_y$ and $H_z$,

$$E_{i,0}^{(e)} = \frac{E_x c_{i+2} - E_y b_{i+2}}{l_i} \tag{5.7}$$

$$H_{i,0}^{(e)} = H_z. \tag{5.8}$$

The values of any sources, $J_{i,n}^{(e)}$, must be defined for all elements and time points.

### 5.3 External Boundaries

The calculation of the fields in DGM is done separately for each element, but the field values at adjacent elements are needed to compute the numerical flux. However, some elements have an edge at an external boundary of the region of interest, so there is no defined element opposite to these edges, and no adjacent field value is available for them. Thus, the adjacent field values at the external boundaries must be chosen or estimated. For instance, in Figure 5.4 below, the elements having an edge at the external boundary are shaded. The element *op* is not defined, and in the TM case, the values $E_k^{(op)}$, $E_{k+1}^{(op)}$ and $H_k^{(op)}$ are unknown (the index *n* is omitted here).

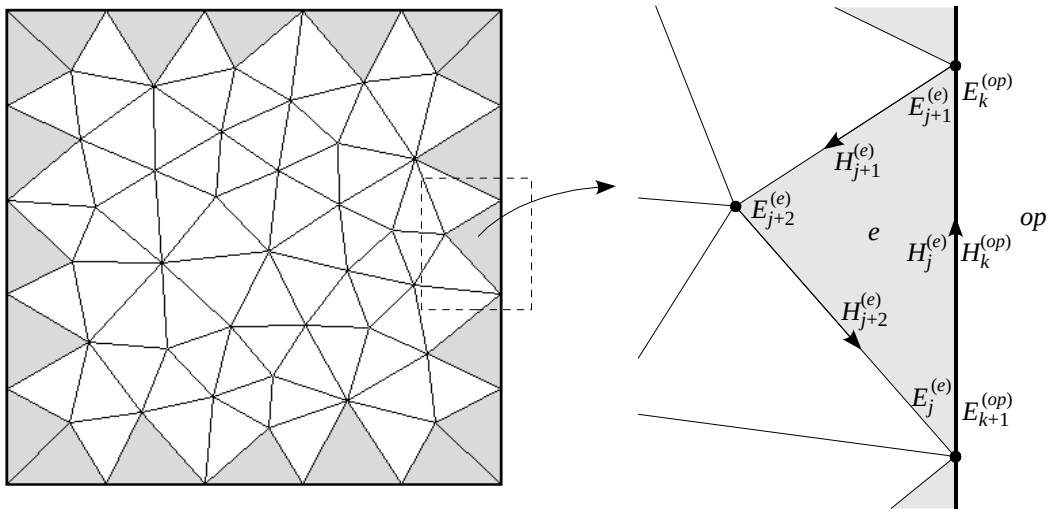



Figure 5.4: Elements at the external boundary.

#### 5.3.1 Conductors

There are several ways to treat the fields at the external boundaries, depending on the problem. Often an external boundary is considered as a perfect electric conductor (PEC) or a

perfect magnetic conductor (PMC). The former models metals with high conductivity, and the latter, although not representing any real material, is useful to study some types of propagating waves. Both kinds of boundaries are used in cavity and waveguide problems.

Inside a PEC, the impedance and the electric field are zero. These properties can be applied to (3.21) with $Z^+ = 0$ and $E^+ = 0$ [7]:

$$\begin{bmatrix} \mathbf{G}_E \\ \mathbf{G}_H \end{bmatrix} = \begin{bmatrix} \dfrac{\hat{n} \times \hat{n} \times \mathbf{E}^-}{Z^-} \\ \hat{n} \times \mathbf{E}^- \end{bmatrix}. \tag{5.9}$$

Alternatively, this expression can be achieved with $Z^+ = Z^-$, $\mathbf{H}^+ = \mathbf{H}^-$, and $\mathbf{E}^+ = -\mathbf{E}^-$. Therefore, in Equations (5.1) and (5.2), the adjacent fields can be set as

$$H_{k,n}^{(op)} = -H_{j,n}^{(e)},\ E_{k+1,n}^{(op)} = -E_{j,n}^{(e)},\ E_{k,n}^{(op)} = -E_{j+1,n}^{(e)} \tag{5.10}$$

and in (5.3) and (5.4) as

$$E_{k,n}^{(op)} = E_{j,n}^{(e)},\ H_{k+1,n}^{(op)} = H_{j,n}^{(e)},\ H_{k,n}^{(op)} = H_{j+1,n}^{(e)} \tag{5.11}$$

both with $Z^{(op)} = Z^{(e)}$ and $Y^{(op)} = Y^{(e)}$.

Inside a PMC, the admittance and the magnetic field are zero. Applying $Y^+ = 0$ and $H^+ = 0$ to (3.21), we have

$$\begin{bmatrix} \mathbf{G}_E \\ \mathbf{G}_H \end{bmatrix} = \begin{bmatrix} -\hat{n} \times \mathbf{H}^- \\ \dfrac{\hat{n} \times \hat{n} \times \mathbf{H}^-}{Y^-} \end{bmatrix} \tag{5.12}$$

which is equivalent to setting $Y^+ = Y^-$, $\mathbf{E}^+ = \mathbf{E}^-$, and $\mathbf{H}^+ = -\mathbf{H}^-$. In Equations (5.1) and (5.2), the adjacent fields can be chosen as

$$H_{k,n}^{(op)} = H_{j,n}^{(e)},\ E_{k+1,n}^{(op)} = E_{j,n}^{(e)},\ E_{k,n}^{(op)} = E_{j+1,n}^{(e)} \tag{5.13}$$

and in (5.3) and (5.4), as

$$E_{k,n}^{(op)} = -E_{j,n}^{(e)},\ H_{k+1,n}^{(op)} = -H_{j,n}^{(e)},\ H_{k,n}^{(op)} = -H_{j+1,n}^{(e)} \tag{5.14}$$

both with $Z^{(op)} = Z^{(e)}$ and $Y^{(op)} = Y^{(e)}$.

### 5.3.2 Absorbing boundary condition

If the external boundary is not a conductor, but free space or a dielectric, the values of the adjacent fields at the boundary are not known. One idea is to place the external boundary very far from any sources and scatterers, such that the waves propagating from these objects will be attenuated to small values at the boundary, and the adjacent fields there can be set to zero. This

approach is not practical, of course, because it requires the calculation of the fields at many additional elements, and such values are not of interest to the problem being simulated. And more importantly, the waves, although attenuated, would be reflected back to the region and corrupt the simulation. Therefore, a better solution is to use a smaller region and estimate the field values at the boundary, based on assumptions about the incident waves. As explained below, these assumptions result in an *absorbing boundary condition* (ABC), so called because it minimizes reflections of incident waves on the boundary.

In a plane wave propagating in the direction $\hat{a}$, the electric and magnetic fields are related by

$$\hat{a}\times\mathbf{E} = Z\mathbf{H},\ \hat{a}\times\mathbf{H} = -Y\mathbf{E}. \tag{5.15}$$

To create the ABC, $\hat{a}$ must be specified. One option is to assume that the wave is normally incident on the boundary, so $\hat{a} = \hat{n}$. In this case, the ABC is

$$\hat{n}\times\mathbf{E} = Z\mathbf{H},\ \hat{n}\times\mathbf{H} = -Y\mathbf{E}. \tag{5.16}$$

Applying these relations to $\mathbf{E}^+$ and $\mathbf{H}^+$ in Equation (3.21), and considering no change in the material parameters across the boundary, the numerical flux becomes

$$\begin{bmatrix}\mathbf{G}_E\\ \mathbf{G}_H\end{bmatrix} = \begin{bmatrix}-\hat{n}\times\dfrac{Z\mathbf{H}^- - \hat{n}\times\mathbf{E}^-}{2Z}\\ \hat{n}\times\dfrac{Y\mathbf{E}^- + \hat{n}\times\mathbf{H}^-}{2Y}\end{bmatrix}. \tag{5.17}$$

If the wave is indeed normally incident, $\mathbf{E}^-$ and $\mathbf{H}^-$ also satisfy (5.16), and this numerical flux becomes zero. Otherwise, a small reflection will occur from the boundary. To implement the ABC, the expression in (5.17) can be achieved by simply setting the adjacent fields to zero. In (5.1) and (5.2):

$$H_{k,n}^{(op)} = 0,\ E_{k+1,n}^{(op)} = 0,\ E_{k,n}^{(op)} = 0 \tag{5.18}$$

and in (5.3) and (5.4):

$$E_{k,n}^{(op)} = 0,\ H_{k+1,n}^{(op)} = 0,\ H_{k,n}^{(op)} = 0 \tag{5.19}$$

both with $Z^{(op)} = Z^{(e)}$ and $Y^{(op)} = Y^{(e)}$.

The ABC presented above results in small reflections only for plane waves at small incident angles with the boundary. For other kinds of waves or higher angles, there are additional methods that can be used to limit reflections. A very common method in scattering problems is the *perfectly matched layer* (PML), which basically consists of a layer of variable conductivity. This method is covered in Chapter 7.

## 6. VALIDATION

The method presented in the previous chapters was implemented in a computer code, which can be found in the Appendix. To validate the code and evaluate the method's accuracy, simple electromagnetic problems with known solutions were tested.

### 6.1 Cavity

The first problem represents a two-dimensional rectangular cavity containing free space, surrounded by a PEC. The discretized region is shown in Figure 6.1 below.

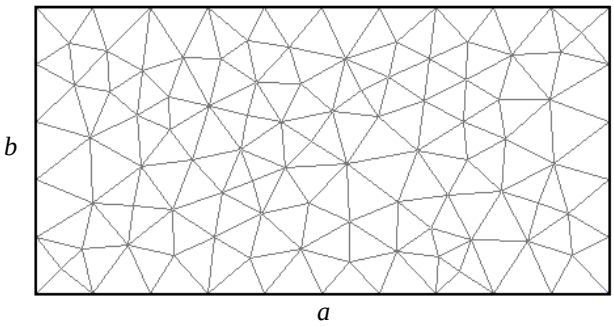



Figure 6.1: Rectangular cavity.

The cavity in this problem has finite dimensions in the *xy* plane, and is considered infinite in the $\hat{z}$ direction. Therefore, the TM fields inside the cavity are given by [8]:

$$\mathbf{E} = E_0 \sin(k_x x)\sin(k_y y)\sin(\omega t)\,\hat{z} \tag{6.1}$$

$$\mathbf{H} = \frac{E_0 k_y}{\eta k}\sin(k_x x)\cos(k_y y)\cos(\omega t)\,\hat{x} - \frac{E_0 k_x}{\eta k}\cos(k_x x)\sin(k_y y)\cos(\omega t)\,\hat{y} \tag{6.2}$$

where $E_0$ is the amplitude of the electric field, $\eta = \sqrt{\mu/\varepsilon}$, $k = \sqrt{k_x^2 + k_y^2}$, $\omega = kc$, and $k_x$ and $k_y$ are wave numbers given by:

$$k_x = \frac{m\pi}{a},\ k_y = \frac{n\pi}{b} \tag{6.3}$$

where *a* and *b* are the horizontal and vertical dimensions of the cavity, respectively, and *m* and *n* are integers greater than zero that define the modes.

In a completely closed cavity without any sources inside, the fields can only oscillate or decay, so it is necessary to define an initial state for the fields, according to (6.1) and (6.2). As

explained in the previous chapter, the coordinates used to calculate **E** and **H** are the points where the respective basis functions are maximum; thus, they are not the same for both fields. In this TM case, the nodes are used for the electric field, and the points given by (2.24) and (2.25), for the magnetic field, then $E_{i,0}^{(e)}$ and $H_{i,0}^{(e)}$ are obtained with (5.5) and (5.6). The adjacent fields at the external boundaries must be chosen appropriately for the PEC, as in (5.10).

Figure 6.2 below presents the results of the cavity simulation for several levels of space discretization and different time discretization schemes. In this example, $m = n = 1$, $a = 1$ m, and $b = 0.5$ m. To satisfy the stability condition, the time step was chosen with $\Delta x/\Delta t = 6c$, where $\Delta x$ is the smallest edge in the entire region. The results show that the accuracy strongly improves with the increase in the number of elements per wavelength, as expected. The Runge-Kutta method provides a much lower error than FDTD, while increasing the computation time only slightly. In this simulation, RK2 and RK4 give the same accuracy, but RK4 uses a longer computation time than RK2, so RK4 does not provide any advantage over RK2 in this simulation. This result occurs probably because linear basis functions are used for space discretization, so a fourth-order would only reduce the error if the basis functions also have a higher polynomial order.

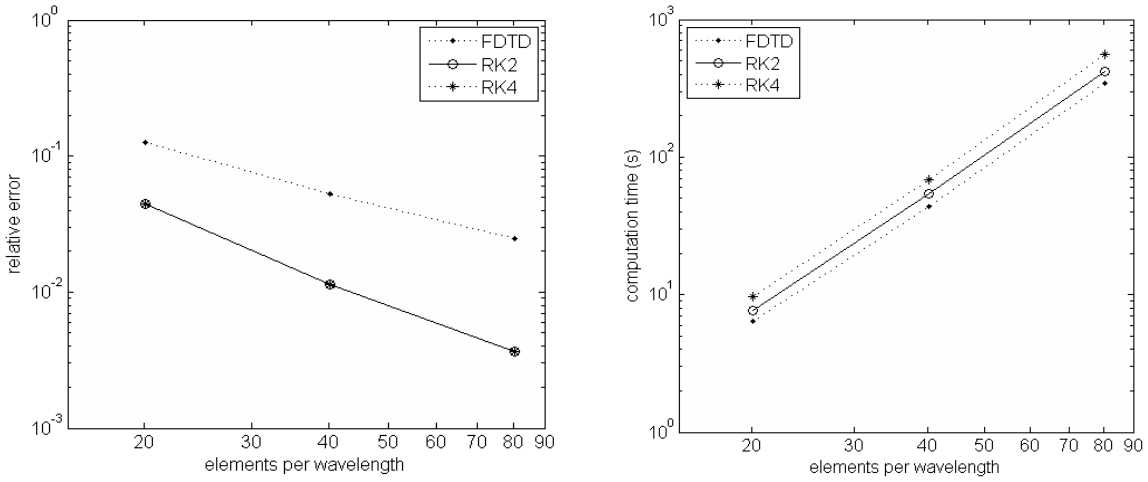



Figure 6.2: Results of the cavity simulation with various element sizes.

Figure 6.3 shows the results of the simulation with several time steps, again for different time discretization schemes, and about 20 elements per wavelength. Although a strong reduction in the error is observed with the decrease in element sizes, the interval between time points does

not seem to interfere with the accuracy significantly, as long as it is small enough to ensure stability.

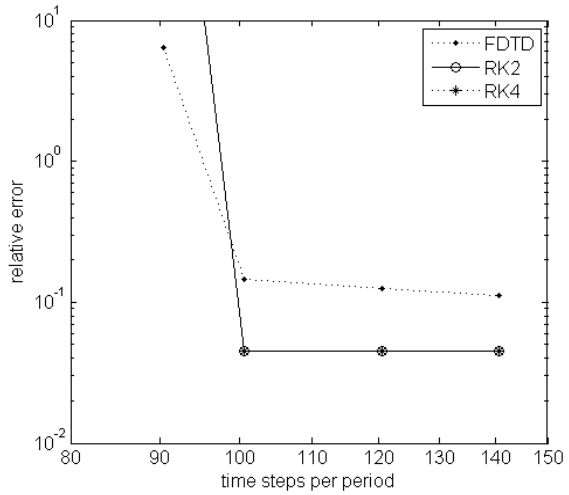



Figure 6.3: Results of the cavity simulation with various time steps.

To verify the stability condition with Figure 6.3, Equation (4.6) can be rewritten in terms of elements per wavelength, $\lambda/\Delta x$, and time steps per period, $T/\Delta t$:

$$\frac{\Delta x}{c\Delta t} = \frac{T/\Delta t}{\lambda/\Delta x} \geq 4\sqrt{2}. \tag{6.4}$$

Therefore, in this problem, the results should become stable if $T/\Delta t \geq 4\sqrt{2}\lambda/\Delta x \approx 113$, which is observed in Figure 6.3.

The error in the above figures was computed with:

$$\text{relative error} = \frac{\sum_{e,i,n} \left| E_{e,i,n} - E_{analytical} \right|}{\sum_{e,i,n} \left| E_{analytical} \right|} \tag{6.5}$$

where $E_{analytical}$ refers to the electric field calculated by (6.1) and (6.2) for the same position and time as $E_{e,i,n}$. A similar expression can be used to find the error for the magnetic field.

## 6.2 Waveguide

Another simple two-dimensional problem with a known analytical solution is the parallel-plate waveguide, consisting of two PEC plates separated by a fixed distance. The plates are

considered infinite, so the cross-section used in the simulation represents in fact only a part of the waveguide. A rectangular region can be used; however, only two of the sides of the rectangle are closed by the PEC, while the others are open, as shown in Figure 6.4 below.

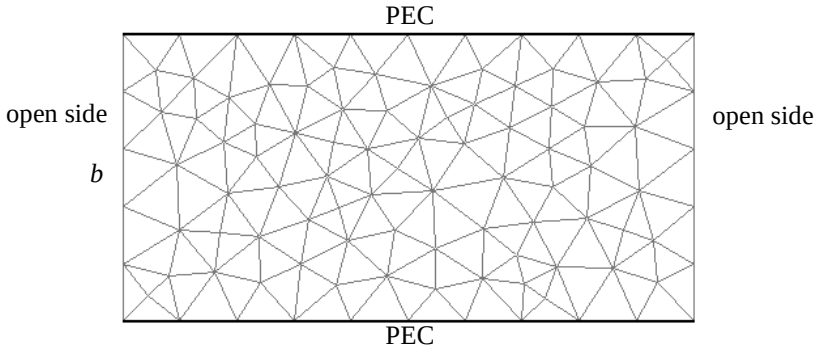


Figure 6.4: Section of parallel-plate waveguide.

Considering a wave that travels inside the waveguide in the $\hat{x}$ direction, the fields at the two open sides, for all time points, as well as the initial state in the entire region, can be set according to the analytical solution for this problem [8]:

$$\mathbf{E} = E_0 \sin\left(k_y y\right) \sin\left(\omega t - k_x x\right) \hat{z} \tag{6.6}$$

$$\mathbf{H} = \frac{E_0 k_y}{k\eta} \cos\left(k_y y\right) \cos\left(\omega t - k_x x\right) \hat{x} - \frac{E_0 k_x}{k\eta} \sin\left(k_y y\right) \sin\left(\omega t - k_x x\right) \hat{y} \tag{6.7}$$

where $E_0$ is the amplitude of the electric field, $\eta = \sqrt{\mu/\varepsilon}$, $\omega$ is the angular frequency of the wave, $k = \omega/c$, $k_x = \sqrt{k^2 - k_y^2}$, and $k_y = n\pi/b$, where $b$ is the vertical dimension of the cavity, and $n$ is an integer greater than zero that defines the mode in the $\hat{y}$ direction. The fields are calculated at the points where the basis functions are maximum, similarly to the previous section.

Figure 6.5 presents the results of the waveguide simulation, again for several levels of space discretization and different time discretization schemes. In this example, $n = 1$, $b = 0.5\,\text{m}$, $f = 500\,\text{MHz}$, where $\omega = 2\pi f$, and $\Delta x/\Delta t = 6c$. Similarly to the cavity simulation, the increase in the number of elements per wavelength enhances the accuracy, and the Runge-Kutta method further reduces the error, with a slight increase in the computation time, but RK2 and RK4 result in the same accuracy. Figure 6.6 shows the results of the simulation with several time steps, for about 13 elements per wavelength. As in the cavity simulation, the interval between time points is only relevant for stability. Again, the results agree with the stability condition, $T/\Delta t \geq 4\sqrt{2}\lambda/\Delta x \approx 74$.

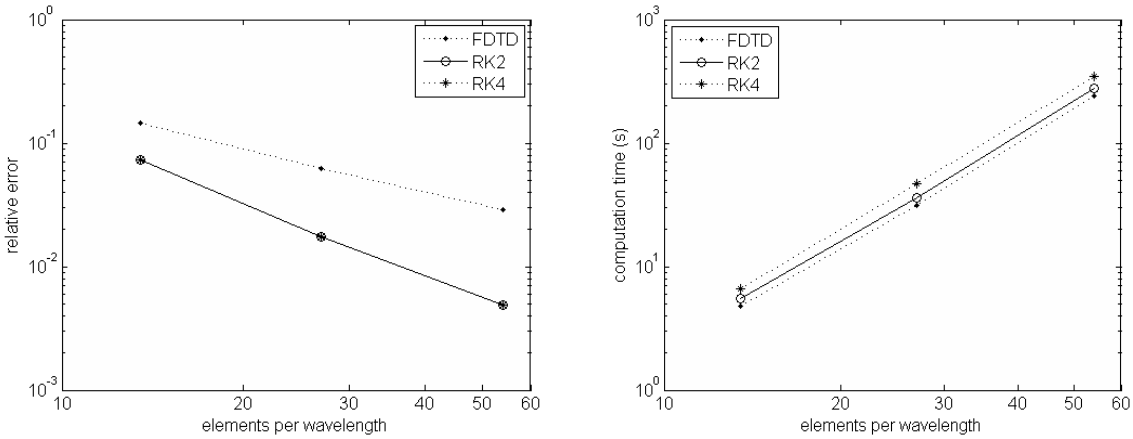



Figure 6.5: Results of the waveguide simulation with various element sizes.

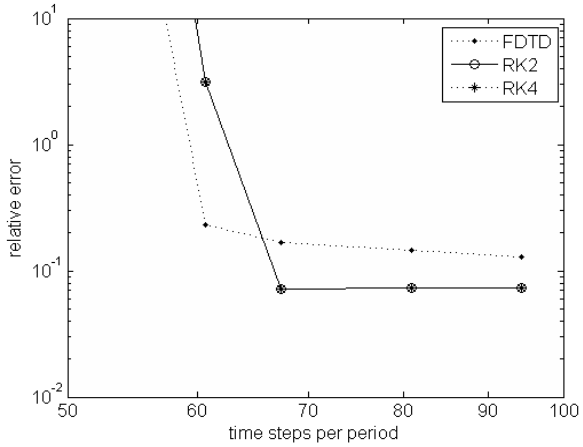



Figure 6.6: Results of the waveguide simulation with various time steps.

Another way to implement the fields in the waveguide is to specify them with Equations (6.6) and (6.7) only at one open side and use an ABC at the other open side. This implementation is more useful for real applications because it does not require a prior knowledge of the field values at all external boundaries. The results of the simulation with the ABC can be seen in Figure 6.7. The error with the ABC is still acceptable, but higher than the error due only to the discretization in Figure 6.5. A more detailed analysis of the ABC and other absorbers is given in the next chapter.

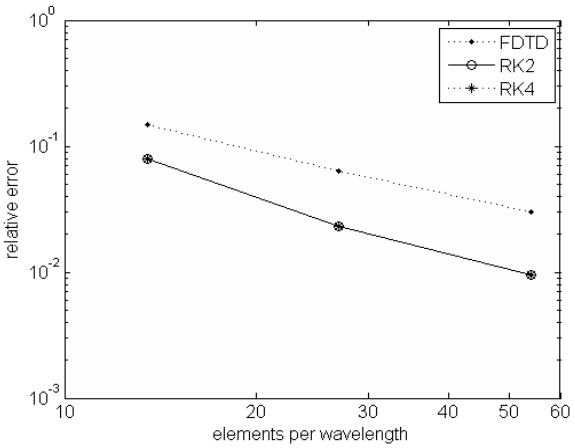



Figure 6.7: Results of the waveguide simulation with ABC.

Based on the results of this chapter, all the remaining simulations in this work use RK2 and $\Delta x/\Delta t = 6c$.

# 7. IMPROVEMENTS

The previous chapters explain the derivation and implementation of DGM, and show numerical results that validate the method and the code used in the simulations. Yet, there are some modifications that can be employed to improve the efficiency of the method. Two improvements are discussed in this chapter: the adaptive time steps, used to reduce the computation time in meshes with a large variation in element sizes, and the perfectly matched layer, which reduces reflections from the external boundaries and thus improves the accuracy of the results. The computer code used to generate the results in this chapter can be found in the Appendix.

## 7.1 Adaptive Time Steps

As explained in Chapter 4, the stability condition depends on the element sizes and the time step. The element sizes are usually chosen to provide a desired level of accuracy; for example, they can be specified as a fraction of the wavelengths involved. After that, the time step to be used in the calculation can be obtained with Equation (4.6). As shown in the previous chapter, reducing the time step below the value necessary for stability does not improve the accuracy significantly, but it increases the computation time because more steps are needed to cover the same total time. Thus, it is desirable to choose the largest time step that ensures stability.

The stability condition must be satisfied in all elements, so if the same time step is used for all elements, the length of the smallest edge in the entire region should be used to calculate the time step. This restriction is not a problem in meshes having uniform element sizes, such as the examples in Figures 6.1 and 6.4 in the previous chapter. However, if the region includes objects that are small compared to the wavelengths considered, or objects that have complex shapes with small details, the elements that discretize these objects or details need to be small as well. These small elements restrict the time step, and if other elements in the region are considerably larger, the time spent to calculate the fields in the larger elements will be unnecessarily long, without much gain in accuracy. Hence, a region with a large variation in element sizes cannot be handled efficiently if the same time step is used for all elements.

One solution is to use adaptive time steps. The ideal approach would be to apply the

stability condition to every element and find a time step for each one. However, this approach is not practical because the fields would be calculated for different time points in each element, and the numerical flux requires known values of the fields at the same positions and times. Thus, an efficient approach is to define classes of elements based on their size, and the time steps used for the classes of larger elements are multiples of those used for the classes of smaller elements [3]. Since the larger time steps are multiples of the smaller ones, the time points coincide.

The following procedure explains how the classes are defined. First, the smallest edge in the whole region, $\Delta x_{\min}$, is identified and used to calculate the time step $\Delta t_{\min}$ with the stability condition. All elements whose smallest edge is smaller than, for instance, $2\Delta x_{\min}$ belong to the first class and their fields are calculated using the time step $\Delta t_{\min}$. The elements whose smallest edge is larger than $2\Delta x_{\min}$ but smaller than $4\Delta x_{\min}$ belong to the second class and use time step $2\Delta t_{\min}$, those with edges larger than $4\Delta x_{\min}$ belong to the third class and use $4\Delta t_{\min}$, and so on. In this example the classes were divided according to powers of two, but any integer may be used. In general, the class number of an element can be calculated with the following expression:

$$C(e) = \text{floor}\left[\log_d\left(\frac{l_{\min}^{(e)}}{\Delta x_{\min}}\right)\right] \tag{7.1}$$

where $C(e)$ is the class of element $e$, $l_{e,\min}$ is the smallest edge of element $e$, and $d$ is the integer used to divide the classes (in the example above, $d = 2$). The function $\text{floor}(a)$ returns the largest integer that is smaller than $a$.

Since higher classes use longer time steps, their fields are not calculated at every time point. For example, in elements of a class using $4\Delta t_{\min}$, the fields are calculated at time $n+4$ from the values at time $n$, and the values at intermediate times $n+1$, $n+2$ and $n+3$ can be obtained through interpolation [3]. Also, the numerical flux uses field values from adjacent elements, which may belong to different classes, so the fields in lower classes can only be calculated after the interpolated values from higher classes are available. Therefore, it is important to observe the order in which the fields are calculated, as illustrated in Figure 7.1, for three classes and $d = 2$. In the figure, this order is denoted by the number on the top left corner of each rectangle. The fields in each rectangle can only be computed after those in rectangles with lower numbers are available. The calculation starts with known $E_n$ and $H_n$ in all elements.

| Class 3: $\Delta x > 4\Delta x_{\min}$ | Class 2: $2\Delta x_{\min} < \Delta x \le 4\Delta x_{\min}$ | Class 1: $\Delta x \le 2\Delta x_{\min}$ |
|---|---|---|
| 1) $E_{n+4}$, $H_{n+4}$ with $4\Delta t_{\min}$ by interpolation: $E_{n+1}$, $H_{n+1}$, $E_{n+2}$, $H_{n+2}$, $E_{n+3}$, $H_{n+3}$ | 1) $E_{n+2}$, $H_{n+2}$ with $2\Delta t_{\min}$ by interpolation: $E_{n+1}$, $H_{n+1}$ | 1) $E_{n+1}$, $H_{n+1}$ with $\Delta t_{\min}$ |
| | | 2) $E_{n+2}$, $H_{n+2}$ with $\Delta t_{\min}$ |
| | 3) $E_{n+4}$, $H_{n+4}$ with $2\Delta t_{\min}$ by interpolation: $E_{n+3}$, $H_{n+3}$ | 3) $E_{n+3}$, $H_{n+3}$ with $\Delta t_{\min}$ |
| | | 4) $E_{n+4}$, $H_{n+4}$ with $\Delta t_{\min}$ |
| 5) $E_{n+8}$, $H_{n+8}$ with $4\Delta t_{\min}$ by interpolation: $E_{n+5}$, $H_{n+5}$, $E_{n+6}$, $H_{n+6}$, $E_{n+7}$, $H_{n+7}$ | 5) $E_{n+6}$, $H_{n+6}$ with $2\Delta t_{\min}$ by interpolation: $E_{n+5}$, $H_{n+5}$ | 5) $E_{n+5}$, $H_{n+5}$ with $\Delta t_{\min}$ |
| | | 6) $E_{n+6}$, $H_{n+6}$ with $\Delta t_{\min}$ |
| | 7) $E_{n+8}$, $H_{n+8}$ with $2\Delta t_{\min}$ by interpolation: $E_{n+7}$, $H_{n+7}$ | 7) $E_{n+7}$, $H_{n+7}$ with $\Delta t_{\min}$ |
| | | 8) $E_{n+8}$, $H_{n+8}$ with $\Delta t_{\min}$ |

Figure 7.1: Order of calculation in elements of different classes.

In the general case, starting from time $n = 0$, the fields in element $e$ should be calculated if $n$ is a multiple of $d^{C(e)}$. If so, the fields are calculated at $n + d^{C(e)}$ from the values at time $n$ and using time step $d^{C(e)}\Delta t_{\min}$, then at any intermediate time points between $n$ and $n + d^{C(e)}$ through interpolation. This procedure conforms to the order of calculation described above.

Figures 7.2 and 7.3 present a comparison between the computation time using a single time step and using adaptive time steps. The two problems analyzed were the same as in the previous chapter, cavity and waveguide, but using different ranges of element sizes. As expected, the reduction in computation time increases with the ratio between largest and smallest element sizes. The error level is not significantly changed with adaptive time steps, only slightly lower.

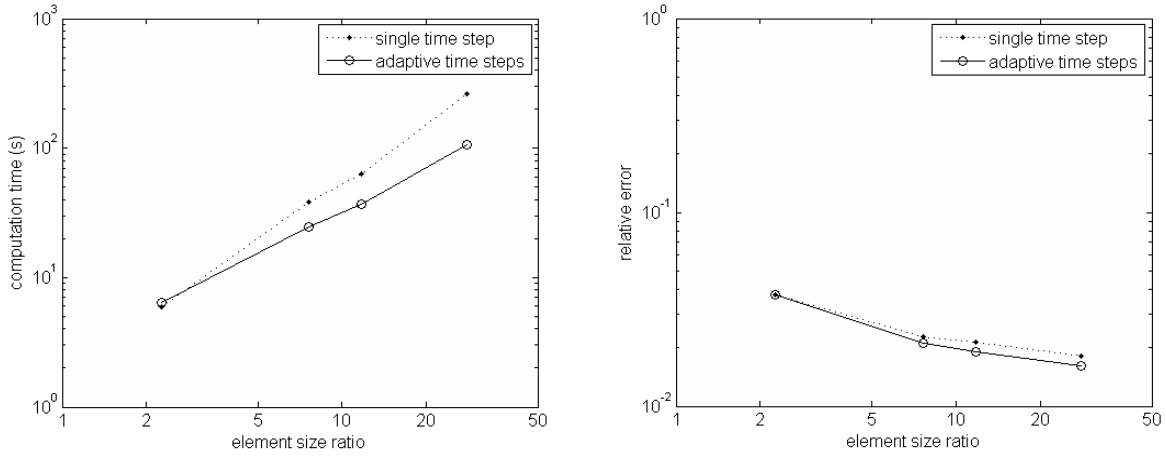



Figure 7.2: Results of the cavity simulation with adaptive time steps.

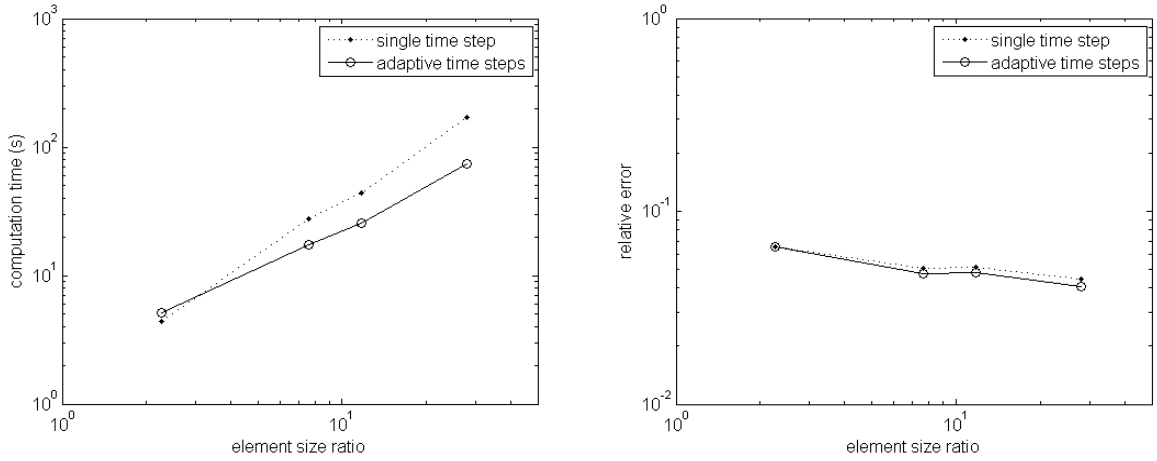



Figure 7.3: Results of the waveguide simulation with adaptive time steps.

## 7.2 Perfectly Matched Layer

As mentioned in Chapter 5, in problems where the external boundary is not a conductor, the field values at the boundary are not known. The ABC estimates these values assuming that the waves at the boundary are plane waves with normal incidence. Thus, the ABC is only accurate if the incident angles on the boundary are small. If the region of interest is kept small, with boundaries near the sources and scatterers, high incident angles may occur, resulting in undesired reflections from the boundary back to the region. Therefore, accurate and efficient simulations of scattering problems require a boundary that creates no reflections, regardless of the type of waves or incident angles.

One type of boundary that possesses such properties, at least ideally, is the *perfectly matched layer* (PML), first proposed by Berenger in 1994 and applied in FDTD [9]. It has since become one of the most popular methods to treat boundaries in scattering problems. The PML is not a simple boundary condition as the ABC, but a layer of varying unidirectional conductivity that attenuates the incident waves without causing reflections.

### 7.2.1 Split-field formulation

The first form of the PML, developed by Berenger, is called a split-field formulation, because one of the fields in Maxwell's equations is split into two components. This formulation is described below for the TM case [9]:

$$\varepsilon \frac{\partial E_{zx}}{\partial t} + \sigma_x E_{zx} = \frac{\partial H_y}{\partial x} \tag{7.2}$$

$$\varepsilon \frac{\partial E_{zy}}{\partial t} + \sigma_y E_{zy} = -\frac{\partial H_x}{\partial y} \tag{7.3}$$

$$\mu \frac{\partial H_x}{\partial t} + \sigma_y^* H_x = -\frac{\partial E_z}{\partial y} \tag{7.4}$$

$$\mu \frac{\partial H_y}{\partial t} + \sigma_x^* H_y = \frac{\partial E_z}{\partial x} \tag{7.5}$$

where $E_z = E_{zx} + E_{zy}$, $\sigma_x$ and $\sigma_y$ are the electric conductivities in the $\hat{x}$ and $\hat{y}$ directions, respectively, and $\sigma_x^*$ and $\sigma_y^*$ are the magnetic conductivities, also in each direction. The magnetic conductivities are not physical, but they can be implemented in the fictitious PML material. The conductivities are zero except in layers near the external boundaries of the region, such that $\sigma_x = \sigma_x^* = 0$ in horizontal layers, and $\sigma_y = \sigma_y^* = 0$ in vertical layers, as shown in Figure 7.4 below. These absorbing layers form the PML region. In most of the region of interest, the conductivities are all zero, and Equations (7.2–7.5) reduce to the regular Maxwell's equations for the TM case.

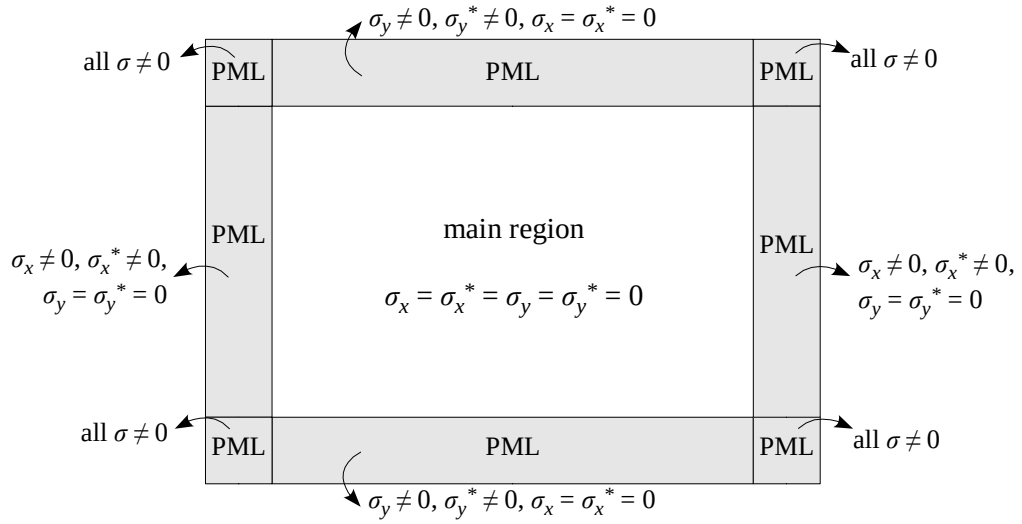



Figure 7.4: Conductivities in the PML regions.

The reflection factor of a plane wave incident on an absorbing layer is:

$$R(\theta) = e^{-2\delta\sigma\eta\cos\theta} \tag{7.6}$$

where $\sigma$ is $\sigma_x$ or $\sigma_y$, and $\delta$ is the thickness of the layer. According to this expression, the layer could be made extremely thin, as long as a high $\sigma$ is used, and the reflection factor can still be

arbitrarily small. However, Equation (7.6) assumes that the conductivity varies continuously inside the layer, which is not true in a discretized region where each element has constant material parameters. In this case, the discontinuities between elements cause numerical reflections, so in practice the absorbing layer should include several elements to minimize these discontinuities. Inside the layer, the conductivities vary from zero, at the edge near the main region of interest, to $\sigma_{\max}$, at the external boundary. A polynomial profile is commonly used to express the variation of the conductivities:

$$\sigma(\rho) = \sigma_{\max}\left(\frac{\rho}{\delta}\right)^n,\ \rho < \delta \tag{7.7}$$

where $\rho$ is the distance to the external boundary, and $n$ is the polynomial order. Usually, $n = 2$ is enough to approximate a smooth variation [5].

The split-field formulation is useful to understand the PML idea, for its simplicity, but it can only be easily implemented in methods using grids, such as FDTD, because the field components are calculated separately. A different formulation is needed for methods based in elements, as explained below.

### 7.2.2 Coordinate-stretching formulation

Another form of PML is derived with coordinate stretching [10]. First, Maxwell's equations are written in frequency domain:

$$-i\omega\varepsilon\mathbf{E} = \nabla\times\mathbf{H} - \sigma\mathbf{E} \tag{7.8}$$

$$i\omega\mu\mathbf{H} = \nabla\times\mathbf{E} \tag{7.9}$$

where $\omega$ is the angular frequency of the fields. Next, the curl is redefined with the following expression:

$$\nabla\times\mathbf{A} = \frac{1}{1+i\omega_x/\omega}\frac{\partial(\hat{x}\times\mathbf{A})}{\partial x} + \frac{1}{1+i\omega_y/\omega}\frac{\partial(\hat{y}\times\mathbf{A})}{\partial y} + \frac{1}{1+i\omega_z/\omega}\frac{\partial(\hat{z}\times\mathbf{A})}{\partial z} \tag{7.10}$$

where $\omega_x$, $\omega_y$ and $\omega_z$ are coordinate stretching variables that attenuate waves, similarly to $\sigma_x$ and $\sigma_y$ in the previous formulation. Applying the redefined curl to the Equation in (7.8),

$$-i\omega\varepsilon\left(E_x\hat{x}+E_y\hat{y}+E_z\hat{z}\right)=\frac{1}{1+i\omega_x/\omega}\frac{\partial\left(H_y\hat{z}-H_z\hat{y}\right)}{\partial x}+\frac{1}{1+i\omega_y/\omega}\frac{\partial\left(H_z\hat{x}-H_x\hat{z}\right)}{\partial y}$$
$$+\frac{1}{1+i\omega_z/\omega}\frac{\partial\left(H_x\hat{y}-H_y\hat{x}\right)}{\partial z}-\sigma\left(E_x\hat{x}+E_y\hat{y}+E_z\hat{z}\right). \qquad (7.11)$$

The $\hat{x}$ component of (7.11) is:

$$-i\omega\varepsilon\left(1+\frac{i\omega_y}{\omega}\right)\left(1+\frac{i\omega_z}{\omega}\right)E_x=\left(1+\frac{i\omega_z}{\omega}\right)\frac{\partial H_z}{\partial y}-\left(1+\frac{i\omega_y}{\omega}\right)\frac{\partial H_y}{\partial z}-\sigma\left(1+\frac{i\omega_y}{\omega}\right)\left(1+\frac{i\omega_z}{\omega}\right)E_x$$
$$-i\omega\varepsilon E_x+\left(\omega_y+\omega_z\right)\varepsilon E_x+\omega_y\omega_z\varepsilon\frac{E_x}{-i\omega}=\frac{\partial}{\partial y}\left(H_z+\omega_z\frac{H_z}{-i\omega}\right)-\frac{\partial}{\partial z}\left(H_y+\omega_y\frac{H_y}{-i\omega}\right)$$
$$-\sigma\left[E_x+\left(\omega_y+\omega_z\right)\frac{E_x}{-i\omega}+\omega_y\omega_z\frac{E_x}{\left(-i\omega\right)^2}\right]. \qquad (7.12)$$

Then the equation can be returned to time domain:

$$\varepsilon\frac{\partial E_x}{\partial t}+\left(\omega_y+\omega_z\right)\varepsilon E_x+\omega_y\omega_z\varepsilon E_x^{(1)}=\frac{\partial}{\partial y}\left(H_z+\omega_z H_z^{(1)}\right)-\frac{\partial}{\partial z}\left(H_y+\omega_y H_y^{(1)}\right)$$
$$-\sigma E_x-\sigma\left(\omega_y+\omega_z\right)E_x^{(1)}-\sigma\omega_y\omega_z E_x^{(2)} \qquad (7.13)$$

where

$$\frac{\partial E_x^{(1)}}{\partial t}=E_x,\ \frac{\partial E_x^{(2)}}{\partial t}=E_x^{(1)},\ \frac{\partial H_y^{(1)}}{\partial t}=H_y,\ \frac{\partial H_z^{(1)}}{\partial t}=H_z. \qquad (7.14)$$

Defining $\tilde{E}_x=E_x+\omega_x E_x^{(1)}$, (7.13) can be written as:

$$\left[\varepsilon\frac{\partial}{\partial t}+\sigma+\varepsilon\left(\omega_y+\omega_z-\omega_x\right)\right]\tilde{E}_x+\left[\sigma\left(\omega_y+\omega_z-\omega_x\right)+\varepsilon\left(\omega_x-\omega_y\right)\left(\omega_x-\omega_z\right)\right]E_x^{(1)}$$
$$+\sigma\omega_y\omega_z E_x^{(2)}=\frac{\partial\tilde{H}_z}{\partial y}-\frac{\partial\tilde{H}_y}{\partial z}\ . \qquad (7.15)$$

Finally, the procedure can be repeated to the $\hat{y}$ and $\hat{z}$ components, and similarly to Equation (7.9). The result is [4]:

$$\left(\varepsilon\frac{\partial}{\partial t}+\sigma+\varepsilon\mathrm{P}\right)\tilde{\mathbf{E}}+\left(\sigma\mathrm{P}+\varepsilon\mathrm{Q}\right)\tilde{\mathbf{E}}^{(1)}+\left(\sigma\mathrm{R}\right)\tilde{\mathbf{E}}^{(2)}+\mathbf{J}=\nabla\times\tilde{\mathbf{H}} \qquad (7.16)$$

$$\left(\mu\frac{\partial}{\partial t}+\mu\mathrm{P}\right)\tilde{\mathbf{H}}+\left(\mu\mathrm{Q}\right)\tilde{\mathbf{H}}^{(1)}=-\nabla\times\tilde{\mathbf{E}} \qquad (7.17)$$

where the auxiliary variables $\tilde{\mathbf{E}}$, $\tilde{\mathbf{H}}$, $\mathbf{E}^{(1)}$, $\mathbf{E}^{(2)}$ and $\mathbf{H}^{(1)}$ are defined by:

$$\tilde{\mathbf{E}}=\mathbf{E}+\mathrm{W}\mathbf{E}^{(1)},\ \tilde{\mathbf{H}}=\mathbf{H}+\mathrm{W}\mathbf{H}^{(1)},\ \frac{\partial\mathbf{E}^{(1)}}{\partial t}=\mathbf{E},\ \frac{\partial\mathbf{E}^{(2)}}{\partial t}=\mathbf{E}^{(1)},\ \frac{\partial\mathbf{H}^{(1)}}{\partial t}=\mathbf{H} \qquad (7.18)$$

and the matrices P, Q and R are:

$$\mathrm{P} = \mathrm{diag}\left(\omega_y + \omega_z - \omega_x, \omega_x + \omega_z - \omega_y, \omega_x + \omega_y - \omega_z\right) \tag{7.19}$$

$$\mathrm{Q} = \mathrm{diag}\left[\left(\omega_x - \omega_y\right)\left(\omega_x - \omega_z\right), \left(\omega_y - \omega_x\right)\left(\omega_y - \omega_z\right), \left(\omega_z - \omega_x\right)\left(\omega_z - \omega_y\right)\right] \tag{7.20}$$

$$\mathrm{R} = \mathrm{diag}\left(\omega_y\omega_z, \omega_x\omega_z, \omega_x\omega_y\right) \tag{7.21}$$

where $\mathrm{diag}\left(a, b, c\right)$ means a 3×3 matrix whose diagonal elements are *a*, *b* and *c*, and whose off-diagonal elements are zero.

In the derivation above, the source **J** was omitted. Equations (7.16) and (7.17) reduce to the original Maxwell's equations if $\omega_x = \omega_y = \omega_z = 0$, and there should not be any sources inside the PML region, so **J** was included only in the final result to give general equations applicable to the whole region of interest.

The DGM procedure in Chapter 3 can be applied to (7.16) and (7.17) to transform it into a pair of matrix equations. Equation (7.16) becomes:

$$\frac{\partial}{\partial t}\left\{\tilde{E}\right\} = \left[M_E\right]^{-1}\left[\frac{\left[S_E\right]}{\varepsilon}\left\{\tilde{H}\right\} + \frac{\left\{F_E\right\}}{\varepsilon} - \left[M_{EP}\right]\left\{\tilde{E}\right\} - \left(\frac{\sigma}{\varepsilon}\left[M_{EP}\right] + \left[M_{EQ}\right]\right)\left\{E^{(1)}\right\}\right.$$
$$\left. - \frac{\sigma}{\varepsilon}\left[M_{ER}\right]\left\{E^{(2)}\right\}\right] - \frac{\sigma}{\varepsilon}\left\{\tilde{E}\right\} - \frac{\left\{J\right\}}{\varepsilon} \tag{7.22}$$

where

$$M_{EPij} = \int_\Omega \mathrm{P}\mathbf{N}_{Ej} \cdot \mathbf{N}_{Ei}\, d\Omega \tag{7.23}$$

$$M_{EQij} = \int_\Omega \mathrm{Q}\mathbf{N}_{Ej} \cdot \mathbf{N}_{Ei}\, d\Omega \tag{7.24}$$

$$M_{ERij} = \int_\Omega \mathrm{R}\mathbf{N}_{Ej} \cdot \mathbf{N}_{Ei}\, d\Omega\,. \tag{7.25}$$

Similarly, Equation (7.17) becomes:

$$\frac{\partial}{\partial t}\left\{\tilde{H}\right\} = \left[M_H\right]^{-1}\left[-\frac{\left[S_H\right]}{\mu}\left\{\tilde{E}\right\} + \frac{\left\{F_H\right\}}{\mu} - \left[M_{HP}\right]\left\{\tilde{H}\right\} - \left[M_{HQ}\right]\left\{H^{(1)}\right\}\right] \tag{7.26}$$

where

$$M_{HPij} = \int_\Omega \mathrm{P}\mathbf{N}_{Hj} \cdot \mathbf{N}_{Hi}\, d\Omega \tag{7.27}$$

$$M_{HQij} = \int_\Omega \mathrm{Q}\mathbf{N}_{Hj} \cdot \mathbf{N}_{Hi}\, d\Omega\,. \tag{7.28}$$

The auxiliary equations in (7.18) can be written as:

$$\frac{\partial}{\partial t}\left\{E^{(1)}\right\} = \left\{\tilde{E}\right\} - \left[M_E\right]^{-1}\left[M_{EW}\right]\left\{E^{(1)}\right\} = \left\{E\right\} \tag{7.29}$$

$$\frac{\partial}{\partial t}\left\{E^{(2)}\right\}=\left\{E^{(1)}\right\} \tag{7.30}$$

$$\frac{\partial}{\partial t}\left\{H^{(1)}\right\}=\left\{\tilde{H}\right\}-\left[M_H\right]^{-1}\left[M_{HW}\right]\left\{H^{(1)}\right\}=\left\{H\right\} \tag{7.31}$$

where

$$M_{EWij}=\int_\Omega \mathrm{W}\mathbf{N}_{Ej}\cdot\mathbf{N}_{Ei}\,d\Omega \tag{7.32}$$

$$M_{HWij}=\int_\Omega \mathrm{W}\mathbf{N}_{Hj}\cdot\mathbf{N}_{Hi}\,d\Omega\,. \tag{7.33}$$

The matrices and vectors $[M_E]$, $[S_E]$, $\{F_E\}$, $[M_H]$, $[S_H]$ and $\{F_H\}$ are the same as in the general DGM derivation.

The matrices defined in (7.23–7.25), (7.27–7.28) and (7.32–7.33) are further evaluated below for the TM case. The basis functions are $\mathbf{N}_{Ei}=N_i\,\hat{z}$ and $\mathbf{N}_{Hi}=\mathbf{N}_i$, and $\omega_z=0$, so:

$$\mathrm{W}\mathbf{N}_{Ei}=(0,0,0),\ \mathrm{W}\mathbf{N}_{Hi}=\left(\omega_x N_{ix},\omega_y N_{iy},0\right)$$

$$M_{EWij}=0 \tag{7.34}$$

$$M_{HWij}=\frac{l_i l_j}{48\Delta}\left[\left(f_{Wi,j}+f_{Wi+1,j+1}\right)\left(1+\delta_{i,j}\right)-f_{Wi+1,j}\left(1+\delta_{i,j+1}\right)-f_{Wi,j+1}\left(1+\delta_{i,j+2}\right)\right] \tag{7.35}$$

where

$$f_{Wi,j}=\omega_x b_i b_j+\omega_y c_i c_j \tag{7.36}$$

$$\mathrm{P}\mathbf{N}_{Ei}=\left[0,0,\left(\omega_x+\omega_y\right)N_i\right],\ \mathrm{Q}\mathbf{N}_{Ei}=\left(0,0,\omega_x\omega_y N_i\right),\ \mathrm{R}\mathbf{N}_{Ei}=\left(0,0,\omega_x\omega_y N_i\right)$$

$$M_{EPij}=\left(\omega_x+\omega_y\right)M_{Eij},\ M_{EQij}=\omega_x\omega_y M_{Eij},\ M_{ERij}=\omega_x\omega_y M_{Eij} \tag{7.37}$$

$$\mathrm{P}\mathbf{N}_{Hi}=\left[\left(\omega_y-\omega_x\right)N_{ix},\left(\omega_x-\omega_y\right)N_{iy},0\right],\ \mathrm{Q}\mathbf{N}_{Hi}=\left[\omega_x\left(\omega_x-\omega_y\right)N_{ix},\omega_y\left(\omega_y-\omega_x\right)N_{iy},0\right]$$

$$M_{HPij}=\frac{l_i l_j}{48\Delta}\left[\left(f_{Pi,j}+f_{Pi+1,j+1}\right)\left(1+\delta_{i,j}\right)-f_{Pi+1,j}\left(1+\delta_{i,j+1}\right)-f_{Pi,j+1}\left(1+\delta_{i,j+2}\right)\right] \tag{7.38}$$

where

$$f_{Pi,j}=\left(\omega_y-\omega_x\right)b_i b_j+\left(\omega_x-\omega_y\right)c_i c_j \tag{7.39}$$

$$M_{HQij}=\frac{l_i l_j}{48\Delta}\left[\left(f_{Qi,j}+f_{Qi+1,j+1}\right)\left(1+\delta_{i,j}\right)-f_{Qi+1,j}\left(1+\delta_{i,j+1}\right)-f_{Qi,j+1}\left(1+\delta_{i,j+2}\right)\right] \tag{7.40}$$

where

$$f_{Qi,j}=\omega_x\left(\omega_x-\omega_y\right)b_i b_j+\omega_y\left(\omega_y-\omega_x\right)c_i c_j. \tag{7.41}$$

The variables $\left\{\tilde{E}\right\}$, $\left\{\tilde{H}\right\}$, $\left\{E^{(1)}\right\}$, $\left\{E^{(2)}\right\}$ and $\left\{H^{(1)}\right\}$ should be calculated first, with Equations (7.22), (7.26) and (7.29–7.31), then their values can be used to calculate $\{E\}$ and $\{H\}$, with Equations

(7.29) and (7.31).

### 7.2.3 Numerical results

The coordinate-stretching formulation for the PML in DGM was implemented in a computer program, which can be found in the Appendix. To compare the accuracy of the PML and ABC, the first problem simulates a plane wave incident on an external boundary, for several incident angles. The coordinate-stretching variables $\omega_x$ and $\omega_y$ vary quadratically inside the absorbing layer, from zero near the main region of interest to $\omega_{\max}$ at the external boundary. To create the plane wave, the adjacent field values at three external boundaries of a rectangular region are set to the field values of the incident wave, and the PML covers the fourth external boundary, as seen in Figure 7.5 below. The ABC is also applied at the fourth external boundary.

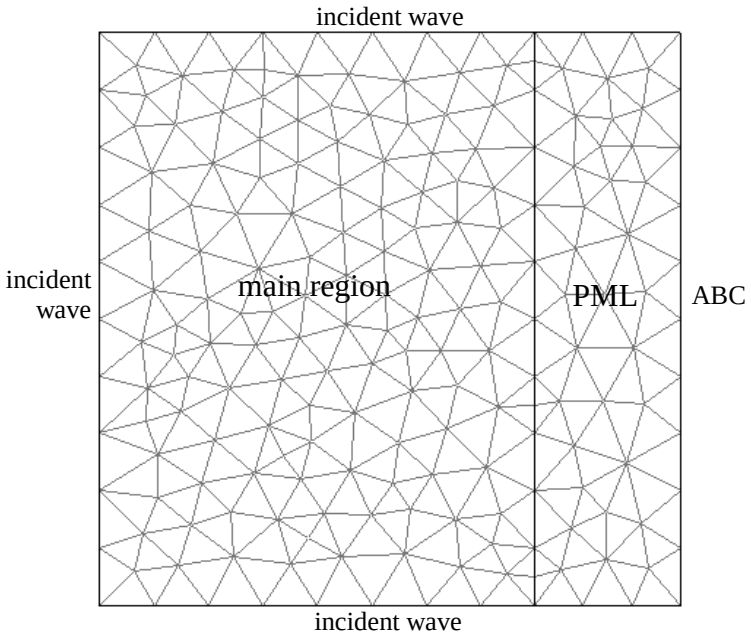



Figure 7.5: Region for simulation of a plane wave with PML.

The relative error for several values of $s = \omega_{\max} / \omega$ is presented in the first graph of Figure 7.6. The line for $s = 0$ represents the simulation with only the ABC. As $s$ increases, the reflection at high angles is reduced, showing the efficiency of the PML, but the reflection at low angles increases because the discontinuities between elements are more pronounced when $s$ is high. On average, the lowest error for this problem is found when $s = 0.2$, represented by the solid line in

the graph. Moreover, the error can be further reduced, especially for high angles of incidence, if the PML includes more elements and is thicker compared to the wavelength, as shown in the second graph of Figure 7.6. These results shows that the PML combined with the ABC provides a better accuracy than if only the ABC is used.

It is important to notice that this relative error is obtained by comparing the values from the simulation with the theoretical field values of a plane wave, so the error is due to the discretization and reflection combined.

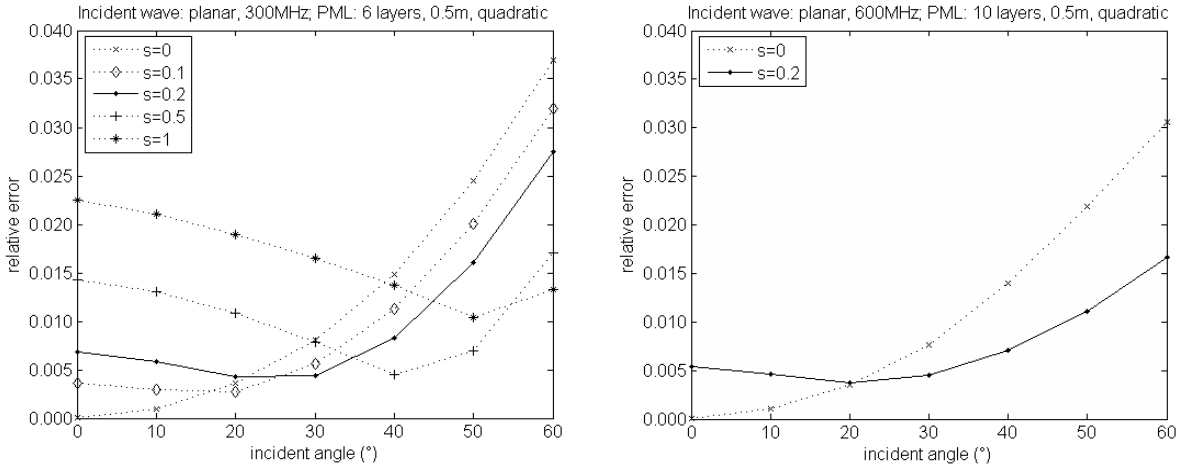



Figure 7.6: Relative error of the PML for a plane wave.

The second problem simulates a current source at the center of the region, surrounded by the PML. Since the method used is in the time domain, the source is represented by a causal function, so there is no simple analytical solution for this problem. However, there is a way to evaluate the accuracy of the PML by comparing two simulations. First, the current source radiates in a very large region, such that during the period considered by the simulation the waves propagate to the external boundary but there is not enough time for them to be reflected. The field values are stored and considered as the reference values. Next, the region is truncated and the PML covers the external boundary. The source radiates, the waves are reflected, and this reflection is calculated by subtracting the reference values of the first simulation from the field values of the second simulation. This procedure is illustrated in Figure 7.7.

Table 7.1 presents the results of the simulation of the radiation from a current source, using several absorbers at the external boundary. The values in the table do not include the error

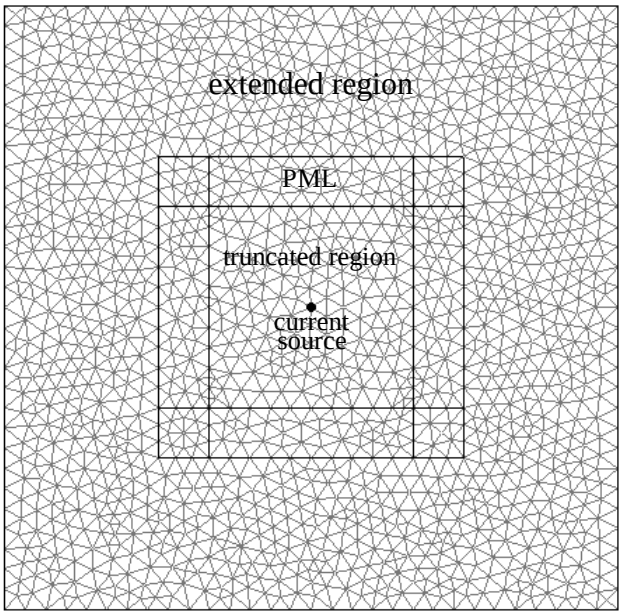



Figure 7.7: Extended and truncated regions.

Table 7.1: Results of the simulation of the radiation from a current source.

a) $f$ = 300 MHz, PML thickness 0.5 m, low $s$ = 0.15, high $s$ = 2, quadratic

| Absorber | Relative error | | Reflection coefficient |
|---|---|---|---|
| | average | near PML | |
| PML with PEC, low $s$ | 36.29% | 55.96% | 30.37% |
| PML with PEC, high $s$ | 6.61% | 8.45% | 14.69% |
| ABC only | 1.48% | 2.55% | 7.58% |
| PML with ABC, low $s$ | **1.06%** | **1.42%** | **2.72%** |
| PML(low $s$) × ABC | 0.54% | 1.43% | **2.30%** |

b) $f$ = 300 MHz, PML thickness 1 m, low $s$ = 0.07, high $s$ = 1.5, quadratic

| Absorber | Relative error | | Reflection coefficient |
|---|---|---|---|
| | average | near PML | |
| PML with PEC, low $s$ | 14.60% | 31.44% | 22.34% |
| PML with PEC, high $s$ | 1.61% | 2.32% | 4.15% |
| ABC only | 0.58% | 1.05% | 4.36% |
| PML with ABC, low $s$ | **0.55%** | **0.72%** | **1.91%** |
| PML(low $s$) × ABC | 0.09% | 0.33% | **0.97%** |

due to the discretization, only the reflection, since the reference and calculated values are both obtained with simulations, as explained above. The reflection varies in time, as does the incident wave, so relative error shown is the time average, while the reflection coefficient is obtained from the reflection peaks. It is expected that the reflection coefficient of the PML combined with the ABC will approximately be the product of two reflection coefficients: of the PML with a PEC at the external boundary, and of only the ABC, as long as the PML with PEC and the PML

with ABC use the same $s = \omega_{max}/\omega$. This expectation is verified in the table. As in the previous example, the best accuracy occurs when the PML is combined with the ABC.

The value of *s* used in the PML with ABC is optimized to provide the lowest reflection. The optimal *s* for the PML with PEC is higher, which explains the high reflection when the same *s* as for the PML with ABC is used. The reflection coefficient of the PML with PEC using the higher optimal *s* is also shown in the table for comparison.

A more complex problem is the scattering of a plane wave due to a conducting cylinder. To simulate this problem, the region around the cylinder should be surrounded by the PML, but this arrangement prevents the use of the external boundaries to create the incident plane wave. Therefore, a fictitious boundary is created between the PML and the cylinder, and the plane wave is created there [11]. This fictitious boundary is called a Huygens boundary because it invokes Huygens's principle to create the wave inside it. Figure 7.8 below shows the region with the scatterer, the PML and the Huygens boundary.

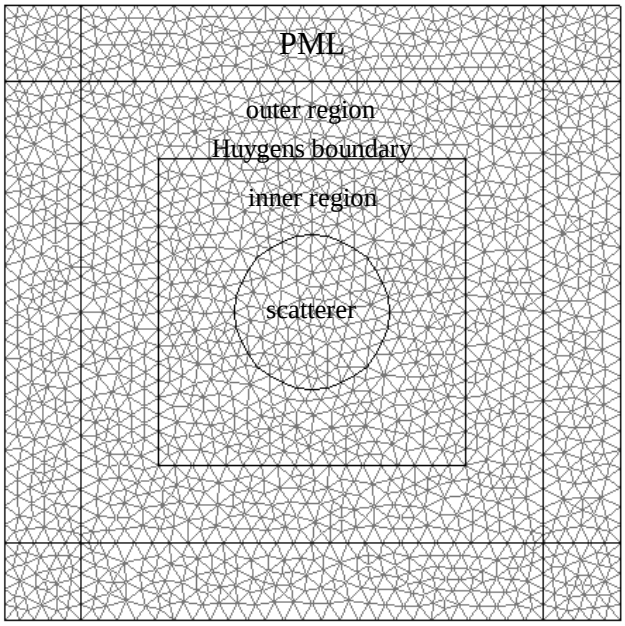



Figure 7.8: Huygens boundary and regions in a scattering problem.

The plane wave should be confined in the inner region, the region surrounded by the Huygens boundary, and only the scattered waves should pass into the outer region. To implement this restriction in DGM, the adjacent fields of elements with an edge at the Huygens boundary

must be modified when calculating the numerical flux. For an element in the inner region, the field values of the plane wave are added to adjacent fields from the outer region. Conversely, for an element in the outer region, the field values of the plane wave are subtracted from the adjacent fields from the inner region.

The figures below show the simulation of this scattering problem. Figure 7.9 presents the electric field as calculated, and the Huygens boundary is clearly visible. Figures 7.10 and 7.11 show the scattered and total fields, respectively; these two figures were generated by adding or subtracting the field values of the plane wave in the outer or inner regions.

Figure 7.9: Electric field as calculated in scattering by a conducting cylinder.

Figure 7.10: Electric field scattered by a conducting cylinder.

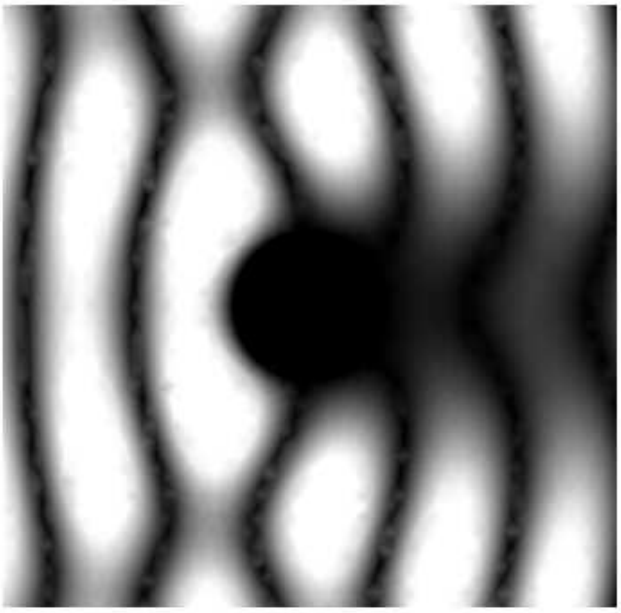


Figure 7.11: Total electric field around a conducting cylinder.

To evaluate the accuracy, the same procedure of the previous simulation can be applied, namely using a large region and then truncating it. Table 7.2 presents the relative error of the numerical results, again showing that the PML combined with the ABC has a better accuracy

than the ABC alone. In addition, the error decreases with a higher level of discretization, as expected.

Table 7.2: Relative error of the scattering simulation.

| Elements per wavelength | Relative error | |
|---|---|---|
| | ABC only | PML with ABC |
| 16.8 | 5.68% | 5.24% |
| 22.5 | 3.88% | 3.42% |
| 28.3 | 3.02% | 2.47% |

## 8. CONCLUSION

Several numerical methods exist for the solution of differential or integral equations in electromagnetic problems. Finite difference in time domain (FDTD) approximates derivatives and usually enables very fast simulations, but the method is not appropriate for an arbitrary geometry if a high accuracy is desired. The finite element metod (FEM), on the other hand, uses elements and basis functions, is more flexible regarding the geometry of the problem and gives more accurate results, but it generally involves the inversion of large matrices and therefore consumes a great amount of time and memory.

The discontinuous Galerkin method (DGM) presented in this work attempts to reduce the time and memory requirements of FEM by calculating the fields in each element separately, thus employing much smaller matrices, and accounting for the discontinuities across element boundaries through numerical fluxes. The complete derivation given in Chapters 3 and 4, along with the detailed implementation procedures in Chapter 5, allow the generation of a computer code to solve any electromagnetic problem with DGM. Chapter 6 shows that the derivation and implementation are valid and produce acceptable results. For the linear basis functions used in the examples, it is found that the best time discretization scheme among those considered is the second-order Runge-Kutta method, for its success in reducing the error, and because the fourth-order method does not provide any further reduction. The stability condition is also analyzed and verified.

Nevertheless, the are some limitations of DGM. The stability condition forces a dependence of the time step on the size of smallest element, so using the same time step for every element is very inefficient when the problem requires a large variation in element sizes. One solution that reduces this inefficiency is the adaptive time steps, detailed in Chapter 7. The solution is successful, but the ideal case of using the best time step for each element is not practical, so the stability condition limits the efficiency of DGM.

Another limitation is the requirement of an absorbing boundary in scattering problems. Two main kinds of absorbers are analyzed—the absorbing boundary condition (ABC) and the perfectly matched layer (PML)—as well as their combination. The original PML formulation is not suitable for methods based on elements, so another formulation is derived. It is also shown, through several examples, that the PML combined with the ABC can achieve better accuracy than the PML or ABC alone. However, the PML extends the computational domain and thus the

decrease in the error is compensated by an increase in the time and memory used.

All simulations done in this work used linear basis functions. Although the error is shown to decrease indefinitely with element sizes, the convergence rate may not be fast enough for simulations requiring a high level of accuracy. Based on these results, it appears that the greatest advantage of DGM is indeed its reduction of matrix orders. Therefore, DGM seems more suitable for simulations requiring fast results, where accuracy is not as important.

Other works have been successful in obtaining very low error levels with DGM by applying higher-order basis functions [12–14], but this approach increases the matrix orders and thus requires more computation time and memory. Future work may concentrate on the comparison of DGM and other numerical methods, regarding accuracy and efficiency.

## REFERENCES


[1] W. H. Reed and T. R. Hill, "Triangular mesh methods for the neutron transport equation," Technical Report LA-UR-73-479, Los Alamos National Laboratory, 1973.

[2] J. S. Hesthaven and T. Warburton, *Nodal Discontinuous Galerkin Methods*. New York, NY: Springer, 2008.

[3] E. Montseny, S. Pernet, X. Ferrières, and G. Cohen, "Dissipative terms and local time-stepping improvements in a spatial high order discontinuous Galerkin scheme for the time-domain Maxwell's equations," *Journal of Computational Physics*, vol. 227, no. 14, pp. 6795-6820, July 2008.

[4] T. Xiao and Q. H. Liu, "Three-dimensional unstructured-grid discontinuous Galerkin method for Maxwell's equations with well-posed perfectly matched layer," *Microwave and Optical Technology Letters*, vol. 46, no. 5, Sep. 2005.

[5] J.-M. Jin, *Electromagnetic Field Computation*, ECE 540 Course Notes, University of Illinois at Urbana-Champaign, 2008.

[6] J.-M. Jin, *The Finite Element Method in Electromagnetics*, 2nd ed. New York, NY: Wiley-Interscience, 2002.

[7] N. Gödel, S. Lange, and M. Clemens, "Time domain discontinuous Galerkin method with efficient modelling of boundary conditions for simulations of electromagnetic wave propagation," in *Asia-Pacific Symposium on Electromagnetic Compatibility and 19th International Zurich Symposium on Electromagnetic Compatibility*, May 2008, pp. 594-597.

[8] C. A. Balanis, *Advanced Engineering Electromagnetics*. New York, NY: Wiley, 1989.

[9] J.-P. Berenger, “A perfectly matched layer for the absorption of electromagnetic waves,” *Journal of Computational Physics*, vol. 114, no. 2, pp. 185-200, Oct. 1994.

[10] G.-X. Fan and Q. H. Liu, “A well-posed PML absorbing boundary condition for lossy media,” in *IEEE Antennas and Propagation Society International Symposium*, July 2001, vol. 3, pp. 2-5.

[11] W.-J. Yao and Y.-X. Wang, “An equivalence principle-based plane wave excitation in time/envelope-domain finite-element analysis,” *IEEE Antennas and Wireless Propagation Letters*, vol. 2, pp. 337-340, 2003.

[12] J. S. Hesthaven and T. Warburton, “Nodal high-order methods on unstructured grids,” *Journal of Computational Physics*, vol. 181, no. 1, pp. 186-221, Sep. 2002.

[13] J. S. Hesthaven and T. Warburton, “High-order accurate methods for time-domain electromagnetics,” *Computer Modeling in Engineering and Sciences*, vol. 5, no. 5, pp. 395-407, 2004.

[14] S. D. Gedney, C. Luo, J. A. Roden, R. D. Crawford, B. Guernsey, J. A. Miller, and E. W. Lucas, “A discontinuous Galerkin finite element time domain method with PML,” in *IEEE Antennas and Propagation Society International Symposium*, July 2008, pp. 1-4.

## APPENDIX: MATLAB CODE

This appendix provides some of the Matlab codes used to generate the numerical results of the previous chapters. The codes listed below are for cavity and waveguide problems, cavity with adaptive time steps, plane wave, current source, and scattering by a conducting cylinder. The last three codes include an implementation of the PML.

Cavity problem

```
%p: matrix of points (nodes)
%t: matrix of triangles (elements)
%e: matrix of points at the boundaries
% tic;
%2D, TM
ne=size(t,2); %number of elements
Me=zeros(ne,3,3); Se=Me; Mh=Me; Sh=Me;
b=zeros(ne,3); c=b; l=b;
A=zeros(ne,1); f=zeros(3,3);
method=1; scale=1;

np=size(p,2); %number of nodes
opposite=zeros(np,np,2);
for el=1:ne
      for i=1:3
            for j=1:3
                  if i~=j
                  if opposite(t(i,el),t(j,el),1)==0
                        opposite(t(i,el),t(j,el),1)=el;
                  else
                        opposite(t(i,el),t(j,el),2)=el;
                  end;
                  end;
            end;
      end;
end;

for el=1:ne
    b(el,1)=p(2,t(2,el))-p(2,t(3,el));
    b(el,2)=p(2,t(3,el))-p(2,t(1,el));
    b(el,3)=p(2,t(1,el))-p(2,t(2,el));
    c(el,1)=p(1,t(3,el))-p(1,t(2,el));
    c(el,2)=p(1,t(1,el))-p(1,t(3,el));
    c(el,3)=p(1,t(2,el))-p(1,t(1,el));
    l(el,1)=sqrt(c(el,3)^2+b(el,3)^2);
    l(el,2)=sqrt(c(el,1)^2+b(el,1)^2);
    l(el,3)=sqrt(c(el,2)^2+b(el,2)^2);
    A(el)=(b(el,1)*c(el,2)-b(el,2)*c(el,1))/2;
    for i=1:3
        for j=1:3
            Me(el,i,j)=A(el)/12*(1+(i==j));
            Se(el,i,j)=l(el,j)/3;
            f(i,j)=b(el,i)*b(el,j)+c(el,i)*c(el,j);
        end;
    end;
```

```
    for i=1:3
        for j=1:3
            temp=(f(i+1-3*(i+1>3),j+1-3*(j+1>3))+f(i,j))*(1+(i==j));
            temp=temp-f(i+1-3*(i+1>3),j)*(1+(i==j+1-3*(j+1>3)))-f(i,j+1-3*
(j+1>3))*(1+(i==j+2-3*(j+2>3)));
            Mh(el,i,j)=l(el,i)*l(el,j)/(48*A(el))*temp;
            Sh(el,i,j)=l(el,i)/6*(3*(i==j+1-3*(j+1>3))-1);
        end;
    end;
end;
lmin=min(min(l));

vc=299792458;
mu=pi*4e-7;
eps=1/(mu*vc^2);%free space
eta=sqrt(mu/eps);
ky=1*pi/(max(p(2,:))-min(p(2,:)));
kx=1*pi/(max(p(1,:))-min(p(1,:)));
k=sqrt(kx^2+ky^2);
freq=vc*k/(2*pi);
snumber=1;
dt=lmin/vc/4.5;
Nt=floor(1/freq/dt);         %number of points of t
time=0:dt:dt*Nt; time=time(:);
xd=p(1,:); yd=p(2,:); Ed=zeros(np,size(time,1));
xd=xd(:);yd=yd(:);
tri=delaunay(xd,yd);
clear scenes;

%initial E and H
E=zeros(ne,3,size(time,1));
H=zeros(ne,3,size(time,1));
for el=1:ne
      for i=1:3
            x=p(1,t(i,el))-min(p(1,:));
            y=p(2,t(i,el))-min(p(2,:));
            nt=1;
            %for nt=1:size(time,1)
        E(el,i,nt)=1*sin(2*pi*freq*time(nt))*sin(kx*x)*sin(ky*y);
            %end;
            x=p(1,t(i+2-3*(i+2>3),el))-2*A(el)*b(el,i+2-3*(i+2>3))/(l(el,i)
^2)-min(p(1,:));
        y=p(2,t(i+2-3*(i+2>3),el))-2*A(el)*c(el,i+2-3*(i+2>3))/(l(el,i)^2)-
min(p(2,:));
            nt=1;
            %for nt=1:size(time,1)
        Hy=-1*kx/(k*eta)*sin(ky*y)*cos(2*pi*freq*time(nt))*cos(kx*x);
        Hx=1*(k^2-kx^2)/(ky*k*eta)*cos(ky*y)*cos(2*pi*freq*time(nt))*sin
(kx*x);
        H(el,i,nt)=Hx*c(el,i+2-3*(i+2>3))/l(el,i)-Hy*b(el,i+2-3*(i+2>3))/l
(el,i);
            %end;
      end;
end;

for nt=1:size(time,1)-1
      %plot
      nt
      Edpoints=zeros(np,1);
      for el=1:ne
            for i=1:3
```

```
                Ed(t(i,el),nt)=Ed(t(i,el),nt)+E(el,i,nt);
                Edpoints(t(i,el))=Edpoints(t(i,el))+1;
            end;
        end;
        for el=1:np
            Ed(el,nt)=Ed(el,nt)/Edpoints(el);
        end;
%       figure(2);
%       trisurf(tri,xd,yd,Ed(:,nt));%-Ead(:,nt));
%       xlim([min(p(1,:)) max(p(1,:))]); ylim([min(p(2,:)) max(p(2,:))]);
%       zlim([-2 2]); caxis([-2 2]); %shading interp;
%       axis square; view([0 0]);
%       scenes(nt)=getframe;
        %calculate
        for el=1:ne
          Met=zeros(3,3); Mht=Met; Set=Met; Sht=Met;
          Et=zeros(3,1); Ht=Et; Fe=Et; Fh=Et;
          for i=1:3
              Et(i)=E(el,i,nt);
              Ht(i)=H(el,i,nt);
              for j=1:3
                  Met(i,j)=Me(el,i,j);
                  Mht(i,j)=Mh(el,i,j);
                  Set(i,j)=Se(el,i,j);
                  Sht(i,j)=Sh(el,i,j);
              end;
          end;
          Met=inv(Met); Mht=inv(Mht);
          for i=1:3
              for j=1:3
                    op=opposite(t(j,el),t(j+1-3*(j+1>3),el),1);
                    if op==el
                          op=opposite(t(j,el),t(j+1-3*(j+1>3),el),2);
                    end;
                    if op==0
                          Eop0=-Et(j);
                      Eop1=-Et(j+1-3*(j+1>3));
                          Hop=-Ht(j);
                  else
                          opj=0;
                          if opposite(t(1,op),t(2,op),1)==el || opposite(t
(1,op),t(2,op),2)==el
                                opj=1; end;
                          if opposite(t(2,op),t(3,op),1)==el || opposite(t
(2,op),t(3,op),2)==el
                                opj=2; end;
                          if opposite(t(3,op),t(1,op),1)==el || opposite(t
(3,op),t(1,op),2)==el
                                opj=3; end;
                          Hop=H(op,opj,nt);
                          Eop0=E(op,opj+1-3*(opj+1>3),nt);
                      Eop1=E(op,opj,nt);
                    end;
                  temp=eta*(-Hop-Ht(j))*3+(Eop0-Et(j))*(1+(i==j));
                  temp=temp+(Eop1-Et(j+1-3*(j+1>3)))*(1+(i==j+1-3*(j+1>3)));
                  temp=temp*l(el,j)/6*(1-(i==j+2-3*(j+2>3)))/(eta+eta);
                  Fe(i)=Fe(i)+temp;
                  temp=(-Hop-Ht(j))*2;
                  temp=temp+(1/eta)*(Eop0-Et(j)+Eop1-Et(j+1-3*(j+1>3)));
                  Fh(i)=Fh(i)+temp*l(el,j)/2*(i==j)/(1/eta+1/eta);
              end;
```

```
        end;
            %FDTD
            Erka=1/eps*Met*(Set*Ht+Fe);
            Hrka=1/mu*Mht*(-Sht*Et+Fh);
            E(el,:,nt+1)=(Et+Erka*dt)';
            H(el,:,nt+1)=(Ht+Hrka*dt)';
        %second-order Runge-Kutta method
%           Erkb=1/eps*Met*(Set*(Ht+Hrka*dt/2)+Fe);
%           Hrkb=1/mu*Mht*(-Sht*(Et+Erka*dt/2)+Fh);
%           E(el,:,nt+1)=(Et+Erkb*dt)';
%           H(el,:,nt+1)=(Ht+Hrkb*dt)';
        %fourth order
%           Erkc=1/eps*Met*(Set*(Ht+Hrkb*dt/2)+Fe);
%           Hrkc=1/mu*Mht*(-Sht*(Et+Erkb*dt/2)+Fh);
%           Erkd=1/eps*Met*(Set*(Ht+Hrkc*dt)+Fe);
%           Hrkd=1/mu*Mht*(-Sht*(Et+Erkc*dt)+Fh);
%           E(el,:,nt+1)=(Et+(Erka+2*Erkb+2*Erkc+Erkd)*dt/6)';
%           H(el,:,nt+1)=(Ht+(Hrka+2*Hrkb+2*Hrkc+Hrkd)*dt/6)';
      end;
end;
% comptime(method,scale)=toc;
```

Waveguide problem

```
%p: matrix of points (nodes)
%t: matrix of triangles (elements)
%e: matrix of points at the boundaries
tic;
%2D, TM
ne=size(t,2); %number of elements
Me=zeros(ne,3,3); Se=Me; Mh=Me; Sh=Me;
b=zeros(ne,3); c=b; l=b;
A=zeros(ne,1); f=zeros(3,3);
method=2; scale=3;

np=size(p,2); %number of nodes
opposite=zeros(np,np,2);
for el=1:ne
      for i=1:3
            for j=1:3
                  if i~=j
                  if opposite(t(i,el),t(j,el),1)==0
                        opposite(t(i,el),t(j,el),1)=el;
                  else
                        opposite(t(i,el),t(j,el),2)=el;
                  end;
                  end;
            end;
      end;
end;

xmin=min(p(1,:));
ymin=min(p(2,:));

%boundary
bpointsh=[];
ecorrection=(e(5,size(e,2))>4);
for el=1:size(e,2)
      if e(5,el)==4-ecorrection
            bpointsh=[bpointsh [e(1,el);e(2,el)]];
      end;
end;
for el=1:ne
      for i=1:3
            for j=1:size(bpointsh,2)
                  if t(i,el)==bpointsh(2,j) && t(i+1-3*(i+1>3),el)==bpointsh
(1,j)
                        opposite(t(i,el),t(i+1-3*(i+1>3),el),2)=-1;
                        opposite(t(i+1-3*(i+1>3),el),t(i,el),2)=-1;
                  end;
            end;
      end;
end;
bpointsh=[];
ecorrection=(e(5,size(e,2))>4);
for el=1:size(e,2)
      if e(5,el)==2-ecorrection
            bpointsh=[bpointsh [e(1,el);e(2,el)]];
      end;
end;
for el=1:ne
      for i=1:3
            for j=1:size(bpointsh,2)
```

```
                if t(i,el)==bpointsh(2,j) && t(i+1-3*(i+1>3),el)==bpointsh
(1,j)
                    opposite(t(i,el),t(i+1-3*(i+1>3),el),2)=-2;
                    opposite(t(i+1-3*(i+1>3),el),t(i,el),2)=-2;
                end;
            end;
      end;
end;

for el=1:ne
    b(el,1)=p(2,t(2,el))-p(2,t(3,el));
    b(el,2)=p(2,t(3,el))-p(2,t(1,el));
    b(el,3)=p(2,t(1,el))-p(2,t(2,el));
    c(el,1)=p(1,t(3,el))-p(1,t(2,el));
    c(el,2)=p(1,t(1,el))-p(1,t(3,el));
    c(el,3)=p(1,t(2,el))-p(1,t(1,el));
    l(el,1)=sqrt(c(el,3)^2+b(el,3)^2);
    l(el,2)=sqrt(c(el,1)^2+b(el,1)^2);
    l(el,3)=sqrt(c(el,2)^2+b(el,2)^2);
    A(el)=(b(el,1)*c(el,2)-b(el,2)*c(el,1))/2;
    for i=1:3
        for j=1:3
            Me(el,i,j)=A(el)/12*(1+(i==j));
            Se(el,i,j)=l(el,j)/3;
            f(i,j)=b(el,i)*b(el,j)+c(el,i)*c(el,j);
        end;
    end;
    for i=1:3
        for j=1:3
            temp=(f(i+1-3*(i+1>3),j+1-3*(j+1>3))+f(i,j))*(1+(i==j));
            temp=temp-f(i+1-3*(i+1>3),j)*(1+(i==j+1-3*(j+1>3)))-f(i,j+1-3*
(j+1>3))*(1+(i==j+2-3*(j+2>3)));
            Mh(el,i,j)=l(el,i)*l(el,j)/(48*A(el))*temp;
            Sh(el,i,j)=l(el,i)/6*(3*(i==j+1-3*(j+1>3))-1);
        end;
    end;
end;
lmin=min(min(l));

vc=299792458;
mu=pi*4e-7;
eps=1/(mu*vc^2);%free space
eta=sqrt(mu/eps);
freq=5e8;
k=2*pi*freq/vc;
ky=1*pi/(max(p(2,:))-ymin);
kx=sqrt(k^2-ky^2);
snumber=2;
dt=lmin/vc/6;
Nt=floor(1/freq/dt);         %number of points of t
time=0:dt:dt*Nt; time=time(:);
xd=p(1,:); yd=p(2,:); Ed=zeros(np,size(time,1));
xd=xd(:);yd=yd(:);
tri=delaunay(xd,yd);
clear scenes;

%initial E and H
E=zeros(ne,3,size(time,1));
H=zeros(ne,3,size(time,1));
for el=1:ne
      for i=1:3
```

```
            x=p(1,t(i,el))-xmin;
            y=p(2,t(i,el))-ymin;
            nt=1;
            %for nt=1:size(time,1)
        E(el,i,nt)=1*sin(2*pi*freq*time(nt)-kx*x)*sin(ky*y);
            %end;
            x=p(1,t(i+2-3*(i+2>3),el))-2*A(el)*b(el,i+2-3*(i+2>3))/(l(el,i)
^2)-xmin;
        y=p(2,t(i+2-3*(i+2>3),el))-2*A(el)*c(el,i+2-3*(i+2>3))/(l(el,i)^2)-
ymin;
            nt=1;
            %for nt=1:size(time,1)
        Hy=-1*kx/(k*eta)*sin(ky*y)*sin(2*pi*freq*time(nt)-kx*x);
        Hx=1*ky/(k*eta)*cos(ky*y)*cos(2*pi*freq*time(nt)-kx*x);
        H(el,i,nt)=Hx*c(el,i+2-3*(i+2>3))/l(el,i)-Hy*b(el,i+2-3*(i+2>3))/l
(el,i);
            %end;
      end;
end;

for nt=1:size(time,1)-1
      %plot
      nt
%     Edpoints=zeros(np,1);
%     for el=1:ne
%           for i=1:3
%                 Ed(t(i,el),nt)=Ed(t(i,el),nt)+E(el,i,nt);
%                 Edpoints(t(i,el))=Edpoints(t(i,el))+1;
%           end;
%     end;
%     for el=1:np
%           Ed(el,nt)=Ed(el,nt)/Edpoints(el);
%     end;
%     figure(2);
%     trisurf(tri,xd,yd,Ed(:,nt));%-Ead(:,nt));
%     xlim([min(p(1,:)) max(p(1,:))]); ylim([min(p(2,:)) max(p(2,:))]);
%     zlim([-2 2]); caxis([-2 2]); %shading interp;
%     axis square; view([0 0]);
%     scenes(nt)=getframe;
      %calculate
      for el=1:ne
        Met=zeros(3,3); Mht=Met; Set=Met; Sht=Met;
        Et=zeros(3,1); Ht=Et; Fe=Et; Fh=Et;
        for i=1:3
            Et(i)=E(el,i,nt);
            Ht(i)=H(el,i,nt);
            for j=1:3
                Met(i,j)=Me(el,i,j);
                Mht(i,j)=Mh(el,i,j);
                Set(i,j)=Se(el,i,j);
                Sht(i,j)=Sh(el,i,j);
            end;
        end;
        Met=inv(Met); Mht=inv(Mht);
        for i=1:3
            for j=1:3
                op=opposite(t(j,el),t(j+1-3*(j+1>3),el),1);
                if op==el
                      op=opposite(t(j,el),t(j+1-3*(j+1>3),el),2);
                end;
              if op==0
```

```
                    Eop0=-Et(j);
                Eop1=-Et(j+1-3*(j+1>3));
                    Hop=-Ht(j);
            else
            if op==-1
                        x=p(1,t(j,el))-xmin;
                        y0=p(2,t(j,el))-ymin;
                        y1=p(2,t(j+1-3*(j+1>3),el))-ymin;
                        y2=p(2,t(j+2-3*(j+2>3),el))-ymin;
                        Eop0=1*sin(2*pi*freq*time(nt)-kx*x)*sin(ky*y0);
                        Eop1=1*sin(2*pi*freq*time(nt)-kx*x)*sin(ky*y1);
                        Hop=(-1+2*(y1>y0))*kx/(k*eta)*sin(2*pi*freq*time
(nt)-kx*x)*sin(ky*y2);
            else
            if op==-2
                Eop0=Et(j);
                Eop1=Et(j+1-3*(j+1>3));
                    Hop=-Ht(j);
                    else
                    opj=0;
                    if opposite(t(1,op),t(2,op),1)==el || opposite(t
(1,op),t(2,op),2)==el
                        opj=1; end;
                    if opposite(t(2,op),t(3,op),1)==el || opposite(t
(2,op),t(3,op),2)==el
                        opj=2; end;
                    if opposite(t(3,op),t(1,op),1)==el || opposite(t
(3,op),t(1,op),2)==el
                        opj=3; end;
                    Hop=H(op,opj,nt);
                    Eop0=E(op,opj+1-3*(opj+1>3),nt);
                Eop1=E(op,opj,nt);
            end;
            end;
            end;
            temp=eta*(-Hop-Ht(j))*3+(Eop0-Et(j))*(1+(i==j));
            temp=temp+(Eop1-Et(j+1-3*(j+1>3)))*(1+(i==j+1-3*(j+1>3)));
            temp=temp*l(el,j)/6*(1-(i==j+2-3*(j+2>3)))/(eta+eta);
            Fe(i)=Fe(i)+temp;
            temp=(-Hop-Ht(j))*2;
            temp=temp+(1/eta)*(Eop0-Et(j)+Eop1-Et(j+1-3*(j+1>3)));
            Fh(i)=Fh(i)+temp*l(el,j)/2*(i==j)/(1/eta+1/eta);
        end;end;
        %FDTD
            Erka=1/eps*Met*(Set*Ht+Fe);
            Hrka=1/mu*Mht*(-Sht*Et+Fh);
%           E(el,:,nt+1)=(Et+Erka*dt)';
%           H(el,:,nt+1)=(Ht+Hrka*dt)';
        %second-order Runge-Kutta method
            Erkb=1/eps*Met*(Set*(Ht+Hrka*dt/2)+Fe);
            Hrkb=1/mu*Mht*(-Sht*(Et+Erka*dt/2)+Fh);
            E(el,:,nt+1)=(Et+Erkb*dt)';
            H(el,:,nt+1)=(Ht+Hrkb*dt)';
        %fourth order
%           Erkc=1/eps*Met*(Set*(Ht+Hrkb*dt/2)+Fe);
%           Hrkc=1/mu*Mht*(-Sht*(Et+Erkb*dt/2)+Fh);
%           Erkd=1/eps*Met*(Set*(Ht+Hrkc*dt)+Fe);
%           Hrkd=1/mu*Mht*(-Sht*(Et+Erkc*dt)+Fh);
%           E(el,:,nt+1)=(Et+(Erka+2*Erkb+2*Erkc+Erkd)*dt/6)';
%           H(el,:,nt+1)=(Ht+(Hrka+2*Hrkb+2*Hrkc+Hrkd)*dt/6)';
end;end;comptime(method,scale)=toc;
```

Cavity with adaptive time steps

```
%p: matrix of points (nodes)
%t: matrix of triangles (elements)
%e: matrix of points at the boundaries
tic;
%2D, TM
ne=size(t,2); %number of elements
Me=zeros(ne,3,3); Se=Me; Mh=Me; Sh=Me;
b=zeros(ne,3); c=b; l=b;
A=zeros(ne,1); class=A; f=zeros(3,3);

np=size(p,2); %number of nodes
opposite=zeros(np,np,2);
for el=1:ne
      for i=1:3
            for j=1:3
                  if i~=j
                  if opposite(t(i,el),t(j,el),1)==0
                        opposite(t(i,el),t(j,el),1)=el;
                  else
                        opposite(t(i,el),t(j,el),2)=el;
                  end;
                  end;
            end;
      end;
end;

for el=1:ne
    b(el,1)=p(2,t(2,el))-p(2,t(3,el));
    b(el,2)=p(2,t(3,el))-p(2,t(1,el));
    b(el,3)=p(2,t(1,el))-p(2,t(2,el));
    c(el,1)=p(1,t(3,el))-p(1,t(2,el));
    c(el,2)=p(1,t(1,el))-p(1,t(3,el));
    c(el,3)=p(1,t(2,el))-p(1,t(1,el));
    l(el,1)=sqrt(c(el,3)^2+b(el,3)^2);
    l(el,2)=sqrt(c(el,1)^2+b(el,1)^2);
    l(el,3)=sqrt(c(el,2)^2+b(el,2)^2);
    A(el)=(b(el,1)*c(el,2)-b(el,2)*c(el,1))/2;
      for i=1:3
        for j=1:3
            Me(el,i,j)=A(el)/12*(1+(i==j));
            Se(el,i,j)=l(el,j)/3;
            f(i,j)=b(el,i)*b(el,j)+c(el,i)*c(el,j);
        end;
      end;
    for i=1:3
        for j=1:3
            temp=(f(i+1-3*(i+1>3),j+1-3*(j+1>3))+f(i,j))*(1+(i==j));
            temp=temp-f(i+1-3*(i+1>3),j)*(1+(i==j+1-3*(j+1>3)))-f(i,j+1-3*
(j+1>3))*(1+(i==j+2-3*(j+2>3)));
            Mh(el,i,j)=l(el,i)*l(el,j)/(48*A(el))*temp;
            Sh(el,i,j)=l(el,i)/6*(3*(i==j+1-3*(j+1>3))-1);
        end;
    end;
end;
lmin=min(min(l));

vc=299792458;
mu=pi*4e-7;
eps=1/(mu*vc^2);%free space
```

```
eta=sqrt(mu/eps);
ky=1*pi/(max(p(2,:))-min(p(2,:)));
kx=1*pi/(max(p(1,:))-min(p(1,:)));
k=sqrt(kx^2+ky^2);
freq=vc*k/(2*pi);
snumber=4;
division=2;
dt=lmin/vc/6;
Nt=floor(1/freq/dt);         %number of points of t
time=0:dt:dt*Nt; time=time(:);
xd=p(1,:); yd=p(2,:); Ed=zeros(np,size(time,1));
xd=xd(:);yd=yd(:);
tri=delaunay(xd,yd);
clear scenes;
for el=1:ne
      class(el)=floor(log(min(l(el,:))/lmin)/log(division));
end;

%initial E and H
E=zeros(ne,3,size(time,1));
H=zeros(ne,3,size(time,1));
for el=1:ne
      for i=1:3
            x=p(1,t(i,el))-min(p(1,:));
            y=p(2,t(i,el))-min(p(2,:));
            nt=1;
            %for nt=1:size(time,1)
        E(el,i,nt)=1*sin(2*pi*freq*time(nt))*sin(kx*x)*sin(ky*y);
            %end;
            x=p(1,t(i+2-3*(i+2>3),el))-2*A(el)*b(el,i+2-3*(i+2>3))/(l(el,i)
^2)-min(p(1,:));
        y=p(2,t(i+2-3*(i+2>3),el))-2*A(el)*c(el,i+2-3*(i+2>3))/(l(el,i)^2)-
min(p(2,:));
            nt=1;
            %for nt=1:size(time,1)
        Hy=-1*kx/(k*eta)*sin(ky*y)*cos(2*pi*freq*time(nt))*cos(kx*x);
        Hx=1*(k^2-kx^2)/(ky*k*eta)*cos(ky*y)*cos(2*pi*freq*time(nt))*sin
(kx*x);
        H(el,i,nt)=Hx*c(el,i+2-3*(i+2>3))/l(el,i)-Hy*b(el,i+2-3*(i+2>3))/l
(el,i);
            %end;
      end;
end;

for nt=1:size(time,1)-1
      %plot
%     nt
%     Edpoints=zeros(np,1);
%     for el=1:ne
%           for i=1:3
%                 Ed(t(i,el),nt)=Ed(t(i,el),nt)+E(el,i,nt);
%                 Edpoints(t(i,el))=Edpoints(t(i,el))+1;
%           end;
%     end;
%     for el=1:np
%           Ed(el,nt)=Ed(el,nt)/Edpoints(el);
%     end;
%     figure(2);
%     trisurf(tri,xd,yd,Ed(:,nt));%-Ead(:,nt));
%     xlim([min(p(1,:)) max(p(1,:))]); ylim([min(p(2,:)) max(p(2,:))]);
%     zlim([-2 2]); caxis([-2 2]); %shading interp;
```

```
%     axis square; view([0 0]);
%     scenes(nt)=getframe;
      %calculate
      for el=1:ne
      classfactor=division^class(el);
      if mod(nt-1,classfactor)==0
            Met=zeros(3,3); Mht=Met; Set=Met; Sht=Met;
            Et=zeros(3,1); Ht=Et; Fe=Et; Fh=Et;
            for i=1:3
                  Et(i)=E(el,i,nt);
                  Ht(i)=H(el,i,nt);
                  for j=1:3
                        Met(i,j)=Me(el,i,j);
                        Mht(i,j)=Mh(el,i,j);
                        Set(i,j)=Se(el,i,j);
                        Sht(i,j)=Sh(el,i,j);
                  end;
            end;
            Met=inv(Met); Mht=inv(Mht);
            for i=1:3
                  for j=1:3
                        op=opposite(t(j,el),t(j+1-3*(j+1>3),el),1);
                        if op==el
                              op=opposite(t(j,el),t(j+1-3*(j+1>3),el),2);
                        end;
                        if op==0
                              Eop0=-Et(j);
                              Eop1=-Et(j+1-3*(j+1>3));
                              Hop=-Ht(j);
                        else
                              opj=0;
                              if opposite(t(1,op),t(2,op),1)==el || opposite(t
(1,op),t(2,op),2)==el
                                    opj=1; end;
                              if opposite(t(2,op),t(3,op),1)==el || opposite(t
(2,op),t(3,op),2)==el
                                    opj=2; end;
                              if opposite(t(3,op),t(1,op),1)==el || opposite(t
(3,op),t(1,op),2)==el
                                    opj=3; end;
                              Hop=H(op,opj,nt);
                              Eop0=E(op,opj+1-3*(opj+1>3),nt);
                              Eop1=E(op,opj,nt);
                        end;
                        temp=eta*(-Hop-Ht(j))*3+(Eop0-Et(j))*(1+(i==j));
                        temp=temp+(Eop1-Et(j+1-3*(j+1>3)))*(1+(i==j+1-3*
(j+1>3)));
                        temp=temp*l(el,j)/6*(1-(i==j+2-3*(j+2>3)))/(eta+eta);
                        Fe(i)=Fe(i)+temp;
                        temp=(-Hop-Ht(j))*2;
                        temp=temp+(1/eta)*(Eop0-Et(j)+Eop1-Et(j+1-3*(j+1>3)));
                        Fh(i)=Fh(i)+temp*l(el,j)/2*(i==j)/(1/eta+1/eta);
                  end;
            end;
            %second-order Runge-Kutta method
            Erka=1/eps*Met*(Set*Ht+Fe);
            Hrka=1/mu*Mht*(-Sht*Et+Fh);
            Erkb=1/eps*Met*(Set*(Ht+Hrka*dt/2*classfactor)+Fe);
            Hrkb=1/mu*Mht*(-Sht*(Et+Erka*dt/2*classfactor)+Fh);
            E(el,:,nt+classfactor)=(Et+Erkb*dt*classfactor)';
            H(el,:,nt+classfactor)=(Ht+Hrkb*dt*classfactor)';
```

```
            for i=1:classfactor-1
                  E(el,:,nt+i)=(Et+Erkb*dt*i)';
                  H(el,:,nt+i)=(Ht+Hrkb*dt*i)';
            end;
      end;
      end;
end;
comptime2(snumber)=toc;
```

Plane wave with PML

```
%p: matrix of points (nodes)
%t: matrix of triangles (elements)
%e: matrix of points at the boundaries

%frequency, number of periods and incident angle from plane.m
abcorpec=0; %0 for ABC, 1 for PEC
smax=2*pi*freq*0;
sorder=2;
snumber=2;

%2D, TM
ne=size(t,2); %number of elements
Me=zeros(ne,3,3); Se=Me; Mh=Me; Sh=Me;
b=zeros(ne,3); c=b; l=b;
A=zeros(ne,1); f=zeros(3,3);
%additional matrices and variables for PML
Mhw=Me; Mhp=Me; Mhq=Me;
fw=f; fp=f; fq=f; sx=A; sy=A;

np=size(p,2); %number of nodes
opposite=zeros(np,np,2);
for el=1:ne
      for i=1:3
            for j=1:3
                  if i~=j
                  if opposite(t(i,el),t(j,el),1)==0
                        opposite(t(i,el),t(j,el),1)=el;
                  else
                        opposite(t(i,el),t(j,el),2)=el;
                  end;
                  end;
            end;
      end;
end;

%find extreme points
xmax=max(p(1,:)); xmin=min(p(1,:));
ymax=max(p(2,:)); ymin=min(p(2,:));

%set conductivity components sx and sy
boundaryx=zeros(2,2);
for region=1:2
      tempx=[]; tempy=[];
      for el=1:ne
            if(t(4,el)==region)
                  tempx=[tempx p(1,t(1,el)) p(1,t(2,el)) p(1,t(3,el))];
            end;
      end;
      boundaryx(region,1)=min(tempx); boundaryx(region,2)=max(tempx);
end;
%regions are numbered 1 to 2, from left to right
boundary=zeros(2,1);
for region=1:2
      if boundaryx(region,1)==xmin
            boundary(1)=region; end;
      if boundaryx(region,2)==xmax
            boundary(2)=region; boundary4=boundaryx(region,1); end;
end;
for el=1:ne
```

```
      region=t(4,el);
      centerx=(p(1,t(1,el))+p(1,t(2,el))+p(1,t(3,el)))/3;
      centery=(p(2,t(1,el))+p(2,t(2,el))+p(2,t(3,el)))/3;
      if region==boundary(2)
            sx(el)=smax*((centerx-boundary4)/(xmax-boundary4))^2; end;
end;
%clear boundary* temp* center*;

%boundary
bpointsh=[];
for el=1:size(e,2)
      if (e(6,el)==0 || e(7,el)==0) %&& (e(6,el)==boundary(1) || e(7,el)
==boundary(1))
            bpointsh=[bpointsh [e(1,el);e(2,el)]];
      end;
end;
for el=1:ne
      for i=1:3
            for j=1:size(bpointsh,2)
                  if ((t(i,el)==bpointsh(2,j) && t(i+1-3*(i+1>3),el)==bpointsh
(1,j))...
            || (t(i,el)==bpointsh(1,j) && t(i+1-3*(i+1>3),el)==bpointsh
(2,j)))...
            && (abs(p(1,t(i,el))-xmax)>1e-10 || abs(p(1,t(i+1-3*(i+1>3),el))-
xmax)>1e-10)
                        opposite(t(i,el),t(i+1-3*(i+1>3),el),2)=-1;
                        opposite(t(i+1-3*(i+1>3),el),t(i,el),2)=-1;
                  end;
            end;
      end;
end;

for el=1:ne
    b(el,1)=p(2,t(2,el))-p(2,t(3,el));
    b(el,2)=p(2,t(3,el))-p(2,t(1,el));
    b(el,3)=p(2,t(1,el))-p(2,t(2,el));
    c(el,1)=p(1,t(3,el))-p(1,t(2,el));
    c(el,2)=p(1,t(1,el))-p(1,t(3,el));
    c(el,3)=p(1,t(2,el))-p(1,t(1,el));
    l(el,1)=sqrt(c(el,3)^2+b(el,3)^2);
    l(el,2)=sqrt(c(el,1)^2+b(el,1)^2);
    l(el,3)=sqrt(c(el,2)^2+b(el,2)^2);
    A(el)=(b(el,1)*c(el,2)-b(el,2)*c(el,1))/2;
    for i=1:3
        for j=1:3
            Me(el,i,j)=A(el)/12*(1+(i==j));
            Se(el,i,j)=l(el,j)/3;
            f(i,j)=b(el,i)*b(el,j)+c(el,i)*c(el,j);
                  fw(i,j)=sx(el)*b(el,i)*b(el,j)+sy(el)*c(el,i)*c(el,j);
                  fp(i,j)=(sy(el)-sx(el))*b(el,i)*b(el,j)+(sx(el)-sy(el))*c
(el,i)*c(el,j);
                  fq(i,j)=sx(el)*(sx(el)-sy(el))*b(el,i)*b(el,j)+sy(el)*(sy
(el)-sx(el))*c(el,i)*c(el,j);
        end;
    end;
    for i=1:3
        for j=1:3
            temp=(f(i+1-3*(i+1>3),j+1-3*(j+1>3))+f(i,j))*(1+(i==j));
            temp=temp-f(i+1-3*(i+1>3),j)*(1+(i==j+1-3*(j+1>3)))-f(i,j+1-3*
(j+1>3))*(1+(i==j+2-3*(j+2>3)));
            Mh(el,i,j)=l(el,i)*l(el,j)/(48*A(el))*temp;
```

```
            Sh(el,i,j)=l(el,i)/6*(3*(i==j+1-3*(j+1>3))-1);
                  temp=(fw(i+1-3*(i+1>3),j+1-3*(j+1>3))+fw(i,j))*(1+(i==j));
            temp=temp-fw(i+1-3*(i+1>3),j)*(1+(i==j+1-3*(j+1>3)))-fw(i,j+1-3*
(j+1>3))*(1+(i==j+2-3*(j+2>3)));
            Mhw(el,i,j)=l(el,i)*l(el,j)/(48*A(el))*temp;
                  temp=(fp(i+1-3*(i+1>3),j+1-3*(j+1>3))+fp(i,j))*(1+(i==j));
            temp=temp-fp(i+1-3*(i+1>3),j)*(1+(i==j+1-3*(j+1>3)))-fp(i,j+1-3*
(j+1>3))*(1+(i==j+2-3*(j+2>3)));
            Mhp(el,i,j)=l(el,i)*l(el,j)/(48*A(el))*temp;
                  temp=(fq(i+1-3*(i+1>3),j+1-3*(j+1>3))+fq(i,j))*(1+(i==j));
            temp=temp-fq(i+1-3*(i+1>3),j)*(1+(i==j+1-3*(j+1>3)))-fq(i,j+1-3*
(j+1>3))*(1+(i==j+2-3*(j+2>3)));
            Mhq(el,i,j)=l(el,i)*l(el,j)/(48*A(el))*temp;
        end;
    end;
end;
lmin=min(min(l));

vc=299792458;
mu=pi*4e-7;
eps=1/(mu*vc^2);%free space
eta=sqrt(mu/eps);
k=2*pi*freq/vc;
dt=lmin/vc/6;
Nt=floor(periods/freq/dt);         %number of points of t
time=0:dt:dt*Nt; time=time(:);
xd=p(1,:); yd=p(2,:); Ed=zeros(np,size(time,1));
xd=xd(:);yd=yd(:);
tri=delaunay(xd,yd);
clear scenes;

%initial E and H
E=zeros(ne,3,size(time,1)); H=E;
E1=E; J=E; H1=E; Hz=E;
% for nt=1:size(time,1)
%     for el=1:ne
%           for i=1:3
%                 if(t(i,el)==center)
%                       J(el,i,nt)=1*sin(2*pi*freq*time(nt));
%                 end;
%           end;
%     end;
% end;

for nt=1:size(time,1)-1
      %plot
      nt
      Edpoints=zeros(np,1);
      for el=1:ne
            for i=1:3
                  Ed(t(i,el),nt)=Ed(t(i,el),nt)+E(el,i,nt);
                  Edpoints(t(i,el))=Edpoints(t(i,el))+1;
            end;
      end;
      for el=1:np
            Ed(el,nt)=Ed(el,nt)/Edpoints(el);
      end;
%     figure(1);
%     trisurf(tri,xd,yd,Ed(:,nt));%-Ead(:,nt));
%     xlim([xmin xmax]); ylim([ymin ymax]);
%     zlim([-2 2]); caxis([-1 1]); %shading interp;
```

```
%     view([30 30]); axis square;
%     scenes(nt)=getframe;
      %calculate
      for el=1:ne
        Met=zeros(3,3); Mht=Met; Set=Met; Sht=Met;
        Et=zeros(3,1); Ht=Et; Fe=Et; Fh=Et;
            Mhwt=Met; Mhpt=Met; Mhqt=Met;
            E1t=Et; Jt=Et; H1t=Et; Hzt=Et;
        for i=1:3
            Et(i)=E(el,i,nt);
            Ht(i)=H(el,i,nt);
                  E1t(i)=E1(el,i,nt);
            Jt(i)=J(el,i,nt);
                  H1t(i)=H1(el,i,nt);
            Hzt(i)=Hz(el,i,nt);
            for j=1:3
                Met(i,j)=Me(el,i,j);
                Mht(i,j)=Mh(el,i,j);
                Set(i,j)=Se(el,i,j);
                Sht(i,j)=Sh(el,i,j);
                       Mhwt(i,j)=Mhw(el,i,j); Mhpt(i,j)=Mhp(el,i,j); Mhqt
(i,j)=Mhq(el,i,j);
            end;
        end;
        Met=inv(Met); Mht=inv(Mht);
        for i=1:3
            for j=1:3
                  op=opposite(t(j,el),t(j+1-3*(j+1>3),el),1);
                  if op==el
                        op=opposite(t(j,el),t(j+1-3*(j+1>3),el),2);
                  end;
                if op==0
                        Eop0=-Et(j)*abcorpec;
                    Eop1=-Et(j+1-3*(j+1>3))*abcorpec;
                        Hop=-Ht(j)*abcorpec;
                else
                  if op==-1
                              x0=p(1,t(j,el))-xmin;
                              y0=p(2,t(j,el))-ymin;
                    x1=p(1,t(j+1-3*(j+1>3),el))-xmin;
                    y1=p(2,t(j+1-3*(j+1>3),el))-ymin;
                              x2=p(1,t(j+2-3*(j+2>3),el))-xmin;
                    y2=p(2,t(j+2-3*(j+2>3),el))-ymin;
                    Eop0=1*sin(k*(x0*cos(theta)+y0*sin(theta))-2*pi*freq*time
(nt))...
                        *(time(nt)>(x0*cos(theta)+y0*sin(theta))/vc);
                    Eop1=1*sin(k*(x1*cos(theta)+y1*sin(theta))-2*pi*freq*time
(nt))...
                        *(time(nt)>(x1*cos(theta)+y1*sin(theta))/vc);
                    if abs(y1-y0)>1e-10 %vertical boundary
                        Hop=-cos(theta)/eta*sin(k*(x0*cos(theta)+y2*sin
(theta))-2*pi*freq*time(nt))...
                            *(1-2*(y1>y0))*(time(nt)>(x0*cos(theta)+y2*sin
(theta))/vc);
                    else %horizontal boundary
                        Hop=-sin(theta)/eta*sin(k*(x2*cos(theta)+y0*sin
(theta))-2*pi*freq*time(nt))...
                            *(1-2*(x0>x1))*(time(nt)>(x2*cos(theta)+y0*sin
(theta))/vc);
                    end;
                        else
```

```
                        opj=0;
                    if opposite(t(1,op),t(2,op),1)==el || opposite(t
(1,op),t(2,op),2)==el
                        opj=1; end;
                    if opposite(t(2,op),t(3,op),1)==el || opposite(t
(2,op),t(3,op),2)==el
                        opj=2; end;
                    if opposite(t(3,op),t(1,op),1)==el || opposite(t
(3,op),t(1,op),2)==el
                        opj=3; end;
                    Hop=H(op,opj,nt);
                    Eop0=E(op,opj+1-3*(opj+1>3),nt);
                 Eop1=E(op,opj,nt);
               end;
             end;
             temp=eta*(-Hop-Ht(j))*3+(Eop0-Et(j))*(1+(i==j));
             temp=temp+(Eop1-Et(j+1-3*(j+1>3)))*(1+(i==j+1-3*(j+1>3)));
             temp=temp*l(el,j)/6*(1-(i==j+2-3*(j+2>3)))/(eta+eta);
             Fe(i)=Fe(i)+temp;
             temp=(-Hop-Ht(j))*2;
             temp=temp+(1/eta)*(Eop0-Et(j)+Eop1-Et(j+1-3*(j+1>3)));
             Fh(i)=Fh(i)+temp*l(el,j)/2*(i==j)/(1/eta+1/eta);
         end;
     end;
     %second-order Runge-Kutta method
         %Runge-Kutta a
         Erka=1/eps*Met*(Set*Hzt+Fe)-(sx(el)+sy(el))*Et-sx(el)*sy(el)*E1t-
Jt/eps;
         Hzrka=Mht*((-Sht*Et+Fh)/mu-Mhpt*Hzt-Mhqt*H1t);
         E1rka=Et;
         H1rka=Hzt-Mht*Mhwt*H1t;
         Jt=(Jt+J(el,:,nt+1)')/2;
         %Runge-Kutta b
         Erkb=1/eps*Met*(Set*(Hzt+Hzrka*dt/2)+Fe)-(sx(el)+sy(el))*
(Et+Erka*dt/2)-sx(el)*sy(el)*(E1t+E1rka*dt/2)-Jt/eps;
         Hzrkb=Mht*((-Sht*(Et+Erka*dt/2)+Fh)/mu-Mhpt*(Hzt+Hzrka*dt/2)-Mhqt*
(H1t+H1rka*dt/2));
         E1rkb=Et+Erka*dt/2;
         H1rkb=Hzt+Hzrka*dt/2-Mht*Mhwt*(H1t+H1rka*dt/2);
         %update next values
         E(el,:,nt+1)=(Et+Erkb*dt)';
         E1(el,:,nt+1)=(E1t+E1rkb*dt)';
         H1(el,:,nt+1)=(H1t+H1rkb*dt)';
         Hz(el,:,nt+1)=(Hzt+Hzrkb*dt)';
         H(el,:,nt+1)=(Hzt+Hzrkb*dt-Mht*Mhwt*(H1t+H1rkb*dt))';
   end;
end;
Edpml=Ed;
```

Current source with PML

```
%p: matrix of points (nodes)
%t: matrix of triangles (elements)
%e: matrix of points at the boundaries

abcorpec=0; %0 for ABC, 1 for PEC
%frequency, periods and lmin from pml.m
smax=2*pi*freq*0.07;
sorder=2;
snumber=4;

%remove region 1
np=size(p,2); %number of nodes
xmax=max(p(1,:)); xmin=min(p(1,:));
ymax=max(p(2,:)); ymin=min(p(2,:));
centerx=(xmax+xmin)/2;
centery=(ymax+ymin)/2;
center=1;
for el=1:np
      if (p(1,el)-centerx)^2+(p(2,el)-centery)^2<(p(1,center)-centerx)^2+(p
(2,center)-centery)^2
            center=el;
      end;
end;
ne=size(t,2); %number of elements
boundaryx=zeros(10,2); boundaryy=boundaryx;
for region=1:10
      tempx=[]; tempy=[];
      for el=1:ne
            if t(4,el)==region
                  tempx=[tempx p(1,t(1,el)) p(1,t(2,el)) p(1,t(3,el))];
                  tempy=[tempy p(2,t(1,el)) p(2,t(2,el)) p(2,t(3,el))];
            end;
      end;
      boundaryx(region,1)=min(tempx); boundaryx(region,2)=max(tempx);
      boundaryy(region,1)=min(tempy); boundaryy(region,2)=max(tempy);
end;
boundaryx(1,:)=[]; boundaryy(1,:)=[];
xmin=min(boundaryx(:,1)); xmax=max(boundaryx(:,2));
ymin=min(boundaryy(:,1)); ymax=max(boundaryy(:,2));
newp=zeros(np,1);
tempp=[];
counterp=1;
for el=1:np
      if p(1,el)>=xmin-1e-10 && p(1,el)<=xmax+1e-10 && p(2,el)>=ymin-1e-10 &&
p(2,el)<=ymax+1e-10
            tempp=[tempp [p(1,el);p(2,el)]];
            newp(el)=counterp;
            counterp=counterp+1;
      end;
end;
p=tempp;
center=newp(center);
tempt=[];
for el=1:ne
      if t(4,el)~=1
            tempt=[tempt [newp(t(1,el));newp(t(2,el));newp(t(3,el));t(4,el)-
1]];
      end;
end;
```

```
t=tempt;

%2D, TM
ne=size(t,2); %number of elements
Me=zeros(ne,3,3); Se=Me; Mh=Me; Sh=Me;
b=zeros(ne,3); c=b; l=b;
A=zeros(ne,1); f=zeros(3,3);
%additional matrices and variables for PML
Mhw=Me; Mhp=Me; Mhq=Me;
fw=f; fp=f; fq=f; sx=A; sy=A;

np=size(p,2); %number of nodes
opposite=zeros(np,np,2);
for el=1:ne
      for i=1:3
            for j=1:3
                  if i~=j
                  if opposite(t(i,el),t(j,el),1)==0
                        opposite(t(i,el),t(j,el),1)=el;
                  else
                        opposite(t(i,el),t(j,el),2)=el;
                  end;
                  end;
            end;
      end;
end;

%set conductivity components sx and sy
boundaryx=zeros(9,2); boundaryy=boundaryx;
for region=1:9
      tempx=[]; tempy=[];
      for el=1:ne
            if t(4,el)==region
                  tempx=[tempx p(1,t(1,el)) p(1,t(2,el)) p(1,t(3,el))];
                  tempy=[tempy p(2,t(1,el)) p(2,t(2,el)) p(2,t(3,el))];
            end;
      end;
      boundaryx(region,1)=min(tempx); boundaryx(region,2)=max(tempx);
      boundaryy(region,1)=min(tempy); boundaryy(region,2)=max(tempy);
end;
%regions are numbered 1 to 8, starting from the top left and increasing
clockwise
boundary=zeros(8,1);
for region=1:9
      if abs(boundaryx(region,1)-xmin)<1e-10 && abs(boundaryy(region,2)-ymax)
<1e-10
            boundary(1)=region; end;
      if boundaryx(region,1)>xmin+1e-10 && boundaryx(region,2)<xmax-1e-10 &&
abs(boundaryy(region,2)-ymax)<1e-10
            boundary(2)=region; boundary2=boundaryy(region,1); end;
      if abs(boundaryx(region,2)-xmax)<1e-10 && abs(boundaryy(region,2)-ymax)
<1e-10
            boundary(3)=region; end;
      if abs(boundaryx(region,2)-xmax)<1e-10 && boundaryy(region,1)>ymin+1e-10
&& boundaryy(region,2)<ymax-1e-10
            boundary(4)=region; boundary4=boundaryx(region,1); end;
      if abs(boundaryx(region,2)-xmax)<1e-10 && abs(boundaryy(region,1)-ymin)
<1e-10
            boundary(5)=region; end;
      if boundaryx(region,1)>xmin+1e-10 && boundaryx(region,2)<xmax-1e-10 &&
abs(boundaryy(region,1)-ymin)<1e-10
```

```
            boundary(6)=region; boundary6=boundaryy(region,2); end;
      if abs(boundaryx(region,1)-xmin)<1e-10 && abs(boundaryy(region,1)-ymin)
<1e-10
            boundary(7)=region; end;
      if abs(boundaryx(region,1)-xmin)<1e-10 && boundaryy(region,1)>ymin+1e-10
&& boundaryy(region,2)<ymax-1e-10
            boundary(8)=region; boundary8=boundaryx(region,2); end;
end;
for el=1:ne
      region=t(4,el);
      centerx=(p(1,t(1,el))+p(1,t(2,el))+p(1,t(3,el)))/3;
      centery=(p(2,t(1,el))+p(2,t(2,el))+p(2,t(3,el)))/3;
      if region==boundary(1) || region==boundary(2) || region==boundary(3)
            sy(el)=smax*((centery-boundary2)/(ymax-boundary2))^sorder; end;
      if region==boundary(3) || region==boundary(4) || region==boundary(5)
            sx(el)=smax*((centerx-boundary4)/(xmax-boundary4))^sorder; end;
      if region==boundary(5) || region==boundary(6) || region==boundary(7)
            sy(el)=smax*((centery-boundary6)/(ymin-boundary6))^sorder; end;
      if region==boundary(7) || region==boundary(8) || region==boundary(1)
            sx(el)=smax*((centerx-boundary8)/(xmin-boundary8))^sorder; end;
end;
%clear boundary boundaryx boundaryy temp*;

for el=1:ne
    b(el,1)=p(2,t(2,el))-p(2,t(3,el));
    b(el,2)=p(2,t(3,el))-p(2,t(1,el));
    b(el,3)=p(2,t(1,el))-p(2,t(2,el));
    c(el,1)=p(1,t(3,el))-p(1,t(2,el));
    c(el,2)=p(1,t(1,el))-p(1,t(3,el));
    c(el,3)=p(1,t(2,el))-p(1,t(1,el));
    l(el,1)=sqrt(c(el,3)^2+b(el,3)^2);
    l(el,2)=sqrt(c(el,1)^2+b(el,1)^2);
    l(el,3)=sqrt(c(el,2)^2+b(el,2)^2);
    A(el)=(b(el,1)*c(el,2)-b(el,2)*c(el,1))/2;
    for i=1:3
        for j=1:3
            Me(el,i,j)=A(el)/12*(1+(i==j));
            Se(el,i,j)=l(el,j)/3;
            f(i,j)=b(el,i)*b(el,j)+c(el,i)*c(el,j);
                  fw(i,j)=sx(el)*b(el,i)*b(el,j)+sy(el)*c(el,i)*c(el,j);
                  fp(i,j)=(sy(el)-sx(el))*b(el,i)*b(el,j)+(sx(el)-sy(el))*c
(el,i)*c(el,j);
                  fq(i,j)=sx(el)*(sx(el)-sy(el))*b(el,i)*b(el,j)+sy(el)*(sy
(el)-sx(el))*c(el,i)*c(el,j);
        end;
    end;
    for i=1:3
        for j=1:3
            temp=(f(i+1-3*(i+1>3),j+1-3*(j+1>3))+f(i,j))*(1+(i==j));
            temp=temp-f(i+1-3*(i+1>3),j)*(1+(i==j+1-3*(j+1>3)))-f(i,j+1-3*
(j+1>3))*(1+(i==j+2-3*(j+2>3)));
            Mh(el,i,j)=l(el,i)*l(el,j)/(48*A(el))*temp;
            Sh(el,i,j)=l(el,i)/6*(3*(i==j+1-3*(j+1>3))-1);
                  temp=(fw(i+1-3*(i+1>3),j+1-3*(j+1>3))+fw(i,j))*(1+(i==j));
            temp=temp-fw(i+1-3*(i+1>3),j)*(1+(i==j+1-3*(j+1>3)))-fw(i,j+1-3*
(j+1>3))*(1+(i==j+2-3*(j+2>3)));
            Mhw(el,i,j)=l(el,i)*l(el,j)/(48*A(el))*temp;
                  temp=(fp(i+1-3*(i+1>3),j+1-3*(j+1>3))+fp(i,j))*(1+(i==j));
            temp=temp-fp(i+1-3*(i+1>3),j)*(1+(i==j+1-3*(j+1>3)))-fp(i,j+1-3*
(j+1>3))*(1+(i==j+2-3*(j+2>3)));
            Mhp(el,i,j)=l(el,i)*l(el,j)/(48*A(el))*temp;
```

```
                  temp=(fq(i+1-3*(i+1>3),j+1-3*(j+1>3))+fq(i,j))*(1+(i==j));
            temp=temp-fq(i+1-3*(i+1>3),j)*(1+(i==j+1-3*(j+1>3)))-fq(i,j+1-3*
(j+1>3))*(1+(i==j+2-3*(j+2>3)));
            Mhq(el,i,j)=l(el,i)*l(el,j)/(48*A(el))*temp;
        end;
    end;
end;
%lmin=min(min(l));

vc=299792458;
mu=pi*4e-7;
eps=1/(mu*vc^2);%free space
eta=sqrt(mu/eps);
k=2*pi*freq/vc;
dt=lmin/vc/6;
Nt=floor(periods/freq/dt);         %number of points of t
time=0:dt:dt*Nt; time=time(:);
xd=p(1,:); yd=p(2,:); Ed=zeros(np,size(time,1));
xd=xd(:);yd=yd(:);
tri=delaunay(xd,yd);
clear scenes;

%initial E and H
E=zeros(ne,3,size(time,1)); H=E;
E1=E; J=E; H1=E; Hz=E;
for nt=1:size(time,1)
      for el=1:ne
            for i=1:3
                  if(t(i,el)==center)
                        J(el,i,nt)=1*sin(2*pi*freq*time(nt));
                  end;
            end;
      end;
end;

for nt=1:size(time,1)-1
      %plot
      nt
      Edpoints=zeros(np,1);
      for el=1:ne
            for i=1:3
                  Ed(t(i,el),nt)=Ed(t(i,el),nt)+E(el,i,nt);
                  Edpoints(t(i,el))=Edpoints(t(i,el))+1;
            end;
      end;
      for el=1:np
            Ed(el,nt)=Ed(el,nt)/Edpoints(el);
      end;
%     figure(2);
%     trisurf(tri,xd,yd,Ed(:,nt));%-Ead(:,nt));
%     xlim([xmin xmax]); ylim([ymin ymax]);
%     zlim([-8 8]); caxis([-4 4]); %shading interp;
%     %view([0 90]); axis square;
%     scenes(nt)=getframe;
      %calculate
      for el=1:ne
        Met=zeros(3,3); Mht=Met; Set=Met; Sht=Met;
        Et=zeros(3,1); Ht=Et; Fe=Et; Fh=Et;
            Mhwt=Met; Mhpt=Met; Mhqt=Met;
            E1t=Et; Jt=Et; H1t=Et; Hzt=Et;
        for i=1:3
```

```
            Et(i)=E(el,i,nt);
            Ht(i)=H(el,i,nt);
                  E1t(i)=E1(el,i,nt);
            Jt(i)=J(el,i,nt);
                  H1t(i)=H1(el,i,nt);
            Hzt(i)=Hz(el,i,nt);
            for j=1:3
                Met(i,j)=Me(el,i,j);
                Mht(i,j)=Mh(el,i,j);
                Set(i,j)=Se(el,i,j);
                Sht(i,j)=Sh(el,i,j);
                       Mhwt(i,j)=Mhw(el,i,j); Mhpt(i,j)=Mhp(el,i,j); Mhqt
(i,j)=Mhq(el,i,j);
            end;
        end;
        Met=inv(Met); Mht=inv(Mht);
        for i=1:3
            for j=1:3
                  op=opposite(t(j,el),t(j+1-3*(j+1>3),el),1);
                  if op==el
                       op=opposite(t(j,el),t(j+1-3*(j+1>3),el),2);
                  end;
                  if op==0
                       Eop0=-Et(j)*abcorpec;
                    Eop1=-Et(j+1-3*(j+1>3))*abcorpec;
                       Hop=-Ht(j)*abcorpec;
                else
                       opj=0;
                       if opposite(t(1,op),t(2,op),1)==el || opposite(t
(1,op),t(2,op),2)==el
                            opj=1; end;
                       if opposite(t(2,op),t(3,op),1)==el || opposite(t
(2,op),t(3,op),2)==el
                            opj=2; end;
                       if opposite(t(3,op),t(1,op),1)==el || opposite(t
(3,op),t(1,op),2)==el
                            opj=3; end;
                       Hop=H(op,opj,nt);
                       Eop0=E(op,opj+1-3*(opj+1>3),nt);
                    Eop1=E(op,opj,nt);
                  end;
                temp=eta*(-Hop-Ht(j))*3+(Eop0-Et(j))*(1+(i==j));
                temp=temp+(Eop1-Et(j+1-3*(j+1>3)))*(1+(i==j+1-3*(j+1>3)));
                temp=temp*l(el,j)/6*(1-(i==j+2-3*(j+2>3)))/(eta+eta);
                Fe(i)=Fe(i)+temp;
                temp=(-Hop-Ht(j))*2;
                temp=temp+(1/eta)*(Eop0-Et(j)+Eop1-Et(j+1-3*(j+1>3)));
                Fh(i)=Fh(i)+temp*l(el,j)/2*(i==j)/(1/eta+1/eta);
            end;
        end;
        %second-order Runge-Kutta method
            %Runge-Kutta a
            Erka=1/eps*Met*(Set*Hzt+Fe)-(sx(el)+sy(el))*Et-sx(el)*sy(el)*E1t-
Jt/eps;
            Hzrka=Mht*((-Sht*Et+Fh)/mu-Mhpt*Hzt-Mhqt*H1t);
            E1rka=Et;
            H1rka=Hzt-Mht*Mhwt*H1t;
            Jt=(Jt+J(el,:,nt+1)')/2;
            %Runge-Kutta b
            Erkb=1/eps*Met*(Set*(Hzt+Hzrka*dt/2)+Fe)-(sx(el)+sy(el))*
(Et+Erka*dt/2)-sx(el)*sy(el)*(E1t+E1rka*dt/2)-Jt/eps;
```

```
            Hzrkb=Mht*((-Sht*(Et+Erka*dt/2)+Fh)/mu-Mhpt*(Hzt+Hzrka*dt/2)-Mhqt*
(H1t+H1rka*dt/2));
            E1rkb=Et+Erka*dt/2;
            H1rkb=Hzt+Hzrka*dt/2-Mht*Mhwt*(H1t+H1rka*dt/2);
            %update next values
            E(el,:,nt+1)=(Et+Erkb*dt)';
            E1(el,:,nt+1)=(E1t+E1rkb*dt)';
            H1(el,:,nt+1)=(H1t+H1rkb*dt)';
            Hz(el,:,nt+1)=(Hzt+Hzrkb*dt)';
            H(el,:,nt+1)=(Hzt+Hzrkb*dt-Mht*Mhwt*(H1t+H1rkb*dt))';
      end;
end;
Edpml=Ed;
```

Scattering by a conducting cylinder with PML

```
%p: matrix of points (nodes)
%t: matrix of triangles (elements)
%e: matrix of points at the boundaries

freq=2e8;
periods=4;
theta=0/180*pi; %incident angle
abcorpec=0; %0 for ABC, 1 for PEC
smax=2*pi*freq*0.2;
sorder=2;
snumber=2;

%2D, TM
ne=size(t,2); %number of elements
Me=zeros(ne,3,3); Se=Me; Mh=Me; Sh=Me;
b=zeros(ne,3); c=b; l=b;
A=zeros(ne,1); f=zeros(3,3);
%additional matrices and variables for PML
Mhw=Me; Mhp=Me; Mhq=Me;
fw=f; fp=f; fq=f; sx=A; sy=A;

np=size(p,2); %number of nodes
opposite=zeros(np,np,2);
for el=1:ne
      for i=1:3
            for j=1:3
                  if i~=j
                  if opposite(t(i,el),t(j,el),1)==0
                        opposite(t(i,el),t(j,el),1)=el;
                  else
                        opposite(t(i,el),t(j,el),2)=el;
                  end;
                  end;
            end;
      end;
end;

%find extreme points
xmax=max(p(1,:)); xmin=min(p(1,:));
ymax=max(p(2,:)); ymin=min(p(2,:));

%find limits of regions
boundaryx=zeros(11,2); boundaryy=boundaryx;
for reg=1:11
      tempx=[]; tempy=[];
      for el=1:ne
            if t(4,el)==reg
                  tempx=[tempx p(1,t(1,el)) p(1,t(2,el)) p(1,t(3,el))];
                  tempy=[tempy p(2,t(1,el)) p(2,t(2,el)) p(2,t(3,el))];
            end;
      end;
      boundaryx(reg,1)=min(tempx); boundaryx(reg,2)=max(tempx);
      boundaryy(reg,1)=min(tempy); boundaryy(reg,2)=max(tempy);
end;
%PML regions are numbered 1 to 8, starting from the top left and increasing
clockwise
region=zeros(11,1);
for reg=1:11
      if abs(boundaryx(reg,1)-xmin)<1e-10 && abs(boundaryy(reg,2)-ymax)<1e-10
```

```
            region(1)=reg; end;
      if boundaryx(reg,1)>xmin+1e-10 && boundaryx(reg,2)<xmax-1e-10 && abs
(boundaryy(reg,2)-ymax)<1e-10
            region(2)=reg; boundary2=boundaryy(reg,1); end;
      if abs(boundaryx(reg,2)-xmax)<1e-10 && abs(boundaryy(reg,2)-ymax)<1e-10
            region(3)=reg; end;
      if abs(boundaryx(reg,2)-xmax)<1e-10 && boundaryy(reg,1)>ymin+1e-10 &&
boundaryy(reg,2)<ymax-1e-10
            region(4)=reg; boundary4=boundaryx(reg,1); end;
      if abs(boundaryx(reg,2)-xmax)<1e-10 && abs(boundaryy(reg,1)-ymin)<1e-10
            region(5)=reg; end;
      if boundaryx(reg,1)>xmin+1e-10 && boundaryx(reg,2)<xmax-1e-10 && abs
(boundaryy(reg,1)-ymin)<1e-10
            region(6)=reg; boundary6=boundaryy(reg,2); end;
      if abs(boundaryx(reg,1)-xmin)<1e-10 && abs(boundaryy(reg,1)-ymin)<1e-10
            region(7)=reg; end;
      if abs(boundaryx(reg,1)-xmin)<1e-10 && boundaryy(reg,1)>ymin+1e-10 &&
boundaryy(reg,2)<ymax-1e-10
            region(8)=reg; boundary8=boundaryx(reg,2); end;
end;
%set conductivity components sx and sy
for el=1:ne
      reg=t(4,el);
      centerx=(p(1,t(1,el))+p(1,t(2,el))+p(1,t(3,el)))/3;
      centery=(p(2,t(1,el))+p(2,t(2,el))+p(2,t(3,el)))/3;
      if reg==region(1) || reg==region(2) || reg==region(3)
            sy(el)=smax*((centery-boundary2)/(ymax-boundary2))^sorder; end;
      if reg==region(3) || reg==region(4) || reg==region(5)
            sx(el)=smax*((centerx-boundary4)/(xmax-boundary4))^sorder; end;
      if reg==region(5) || reg==region(6) || reg==region(7)
            sy(el)=smax*((centery-boundary6)/(ymin-boundary6))^sorder; end;
      if reg==region(7) || reg==region(8) || reg==region(1)
            sx(el)=smax*((centerx-boundary8)/(xmin-boundary8))^sorder; end;
end;
%outer Huygens region
for reg=1:11
      if abs(boundaryx(reg,1)-boundary8)<1e-10 && abs(boundaryx(reg,2)-
boundary4)<1e-10...
      && abs(boundaryy(reg,1)-boundary6)<1e-10 && abs(boundaryy(reg,2)-
boundary2)<1e-10
            region(9)=reg;
      end;
end;
%scatterer region
centerx=(xmax+xmin)/2;
centery=(ymax+ymin)/2;
center=1;
for el=1:ne
      centerpx=mean([p(1,t(1,el)) p(1,t(2,el)) p(1,t(3,el))]);
      centerpy=mean([p(2,t(1,el)) p(2,t(2,el)) p(2,t(3,el))]);
      centercx=mean([p(1,t(1,center)) p(1,t(2,center)) p(1,t(3,center))]);
      centercy=mean([p(2,t(1,center)) p(2,t(2,center)) p(2,t(3,center))]);
      if (centerpx-centerx)^2+(centerpy-centery)^2<(centercx-centerx)^2+
(centercy-centery)^2
            center=el;
      end;
end;
region(11)=t(4,center);
%inner Huygens region
for reg=1:11
      if boundaryx(reg,2)>boundaryx(region(11),2)+1e-10...
```

```
      && boundaryx(reg,2)<boundaryx(region(9),2)-1e-10
            region(10)=reg;
      end;
end;
reg=region(10);
boundary2h=boundaryy(reg,2); boundary4h=boundaryx(reg,2);
boundary6h=boundaryy(reg,1); boundary8h=boundaryx(reg,1);
%clear boundaryx boundaryy temp* center*;

for el=1:ne
    b(el,1)=p(2,t(2,el))-p(2,t(3,el));
    b(el,2)=p(2,t(3,el))-p(2,t(1,el));
    b(el,3)=p(2,t(1,el))-p(2,t(2,el));
    c(el,1)=p(1,t(3,el))-p(1,t(2,el));
    c(el,2)=p(1,t(1,el))-p(1,t(3,el));
    c(el,3)=p(1,t(2,el))-p(1,t(1,el));
    l(el,1)=sqrt(c(el,3)^2+b(el,3)^2);
    l(el,2)=sqrt(c(el,1)^2+b(el,1)^2);
    l(el,3)=sqrt(c(el,2)^2+b(el,2)^2);
    A(el)=(b(el,1)*c(el,2)-b(el,2)*c(el,1))/2;
    for i=1:3
        for j=1:3
            Me(el,i,j)=A(el)/12*(1+(i==j));
            Se(el,i,j)=l(el,j)/3;
            f(i,j)=b(el,i)*b(el,j)+c(el,i)*c(el,j);
                  fw(i,j)=sx(el)*b(el,i)*b(el,j)+sy(el)*c(el,i)*c(el,j);
                  fp(i,j)=(sy(el)-sx(el))*b(el,i)*b(el,j)+(sx(el)-sy(el))*c
(el,i)*c(el,j);
                  fq(i,j)=sx(el)*(sx(el)-sy(el))*b(el,i)*b(el,j)+sy(el)*(sy
(el)-sx(el))*c(el,i)*c(el,j);
        end;
    end;
    for i=1:3
        for j=1:3
            temp=(f(i+1-3*(i+1>3),j+1-3*(j+1>3))+f(i,j))*(1+(i==j));
            temp=temp-f(i+1-3*(i+1>3),j)*(1+(i==j+1-3*(j+1>3)))-f(i,j+1-3*
(j+1>3))*(1+(i==j+2-3*(j+2>3)));
            Mh(el,i,j)=l(el,i)*l(el,j)/(48*A(el))*temp;
            Sh(el,i,j)=l(el,i)/6*(3*(i==j+1-3*(j+1>3))-1);
                  temp=(fw(i+1-3*(i+1>3),j+1-3*(j+1>3))+fw(i,j))*(1+(i==j));
            temp=temp-fw(i+1-3*(i+1>3),j)*(1+(i==j+1-3*(j+1>3)))-fw(i,j+1-3*
(j+1>3))*(1+(i==j+2-3*(j+2>3)));
            Mhw(el,i,j)=l(el,i)*l(el,j)/(48*A(el))*temp;
                  temp=(fp(i+1-3*(i+1>3),j+1-3*(j+1>3))+fp(i,j))*(1+(i==j));
            temp=temp-fp(i+1-3*(i+1>3),j)*(1+(i==j+1-3*(j+1>3)))-fp(i,j+1-3*
(j+1>3))*(1+(i==j+2-3*(j+2>3)));
            Mhp(el,i,j)=l(el,i)*l(el,j)/(48*A(el))*temp;
                  temp=(fq(i+1-3*(i+1>3),j+1-3*(j+1>3))+fq(i,j))*(1+(i==j));
            temp=temp-fq(i+1-3*(i+1>3),j)*(1+(i==j+1-3*(j+1>3)))-fq(i,j+1-3*
(j+1>3))*(1+(i==j+2-3*(j+2>3)));
            Mhq(el,i,j)=l(el,i)*l(el,j)/(48*A(el))*temp;
        end;
    end;
end;
lmin=min(min(l));

vc=299792458;
mu=pi*4e-7;
eps=1/(mu*vc^2);%free space
eta=sqrt(mu/eps);
k=2*pi*freq/vc;
```

```
dt=lmin/vc/6;
Nt=floor(periods/freq/dt)         %number of points of t
time=0:dt:dt*Nt; time=time(:);
xd=p(1,:); yd=p(2,:);
xd=xd(:);yd=yd(:);
tri=delaunay(xd,yd);
Ed=zeros(np,size(time,1));
clear scenes;

%initial E and H
E=zeros(ne,3,size(time,1)); H=E;
E1=E; J=E; H1=E; Hz=E;
% for nt=1:size(time,1)
%     for el=1:ne
%           for i=1:3
%                 if(t(i,el)==center)
%                       J(el,i,nt)=1*sin(2*pi*freq*time(nt));
%                 end;end;end;end;

for nt=1:size(time,1)-1
      %plot
      nt
      Edpoints=zeros(np,1);
      for el=1:ne
            for i=1:3
                  Ed(t(i,el),nt)=Ed(t(i,el),nt)+E(el,i,nt);
                  Edpoints(t(i,el))=Edpoints(t(i,el))+1;
            end;
      end;
      for el=1:np
            Ed(el,nt)=Ed(el,nt)/Edpoints(el);
      end;
%     figure(1);
%     trisurf(tri,xd,yd,Ed(:,nt));%-Ead(:,nt));
%     xlim([xmin xmax]); ylim([ymin ymax]);
%     zlim([-2 2]); caxis([-1 1]); shading interp;
%     view([0 90]); axis square;
%     scenes(nt)=getframe;
      %calculate
      for el=1:ne
        Met=zeros(3,3); Mht=Met; Set=Met; Sht=Met;
        Et=zeros(3,1); Ht=Et; Fe=Et; Fh=Et;
            Mhwt=Met; Mhpt=Met; Mhqt=Met;
            E1t=Et; Jt=Et; H1t=Et; Hzt=Et;
        for i=1:3
            Et(i)=E(el,i,nt);
            Ht(i)=H(el,i,nt);
                  E1t(i)=E1(el,i,nt);
            Jt(i)=J(el,i,nt);
                  H1t(i)=H1(el,i,nt);
            Hzt(i)=Hz(el,i,nt);
            for j=1:3
                Met(i,j)=Me(el,i,j);
                Mht(i,j)=Mh(el,i,j);
                Set(i,j)=Se(el,i,j);
                Sht(i,j)=Sh(el,i,j);
                      Mhwt(i,j)=Mhw(el,i,j); Mhpt(i,j)=Mhp(el,i,j); Mhqt
(i,j)=Mhq(el,i,j);
            end;
        end;
        Met=inv(Met); Mht=inv(Mht);
```

```
        for i=1:3
            for j=1:3
                  op=opposite(t(j,el),t(j+1-3*(j+1>3),el),1);
                  if op==el
                        op=opposite(t(j,el),t(j+1-3*(j+1>3),el),2);
                  end;
                 if op==0
                        Eop0=-Et(j)*abcorpec;
                     Eop1=-Et(j+1-3*(j+1>3))*abcorpec;
                        Hop=Ht(j)*abcorpec;
                        else
                              opj=0;
                        if opposite(t(1,op),t(2,op),1)==el || opposite(t
(1,op),t(2,op),2)==el
                              opj=1; end;
                        if opposite(t(2,op),t(3,op),1)==el || opposite(t
(2,op),t(3,op),2)==el
                              opj=2; end;
                        if opposite(t(3,op),t(1,op),1)==el || opposite(t
(3,op),t(1,op),2)==el
                              opj=3; end;
                        Hop=-H(op,opj,nt);
                        Eop0=E(op,opj+1-3*(opj+1>3),nt);
                    Eop1=E(op,opj,nt);
                              if t(4,el)==region(9) && t(4,op)==region(10)
                                    x0=p(1,t(j,el))-boundary8h;
                                    y0=p(2,t(j,el))-boundary6h;
                                    x1=p(1,t(j+1-3*(j+1>3),el))-boundary8h;
                                    y1=p(2,t(j+1-3*(j+1>3),el))-boundary6h;
                                    x2=p(1,t(j+2-3*(j+2>3),el))-boundary8h;
                                    y2=p(2,t(j+2-3*(j+2>3),el))-boundary6h;
                                    Eop0=Eop0-1*sin(k*(x0*cos(theta)+y0*sin
(theta))-2*pi*freq*time(nt))...
                                          *(time(nt)>(x0*cos(theta)+y0*sin
(theta))/vc);
                                    Eop1=Eop1-1*sin(k*(x1*cos(theta)+y1*sin
(theta))-2*pi*freq*time(nt))...
                                          *(time(nt)>(x1*cos(theta)+y1*sin
(theta))/vc);
                                    if abs(y1-y0)>1e-10 %vertical boundary
                                          Hop=Hop-cos(theta)/eta*sin(k*(x0*cos
(theta)+y2*sin(theta))-2*pi*freq*time(nt))...
                                                *(1-2*(y1>y0))*(time(nt)>
(x0*cos(theta)+y2*sin(theta))/vc);
                                    else %horizontal boundary
                                          Hop=Hop-sin(theta)/eta*sin(k*(x2*cos
(theta)+y0*sin(theta))-2*pi*freq*time(nt))...
                                                *(1-2*(x0>x1))*(time(nt)>
(x2*cos(theta)+y0*sin(theta))/vc);
                                    end;
                              end;
                              if t(4,el)==region(10) && t(4,op)==region(9)
                                    x0=p(1,t(j,el))-boundary8h;
                                    y0=p(2,t(j,el))-boundary6h;
                                    x1=p(1,t(j+1-3*(j+1>3),el))-boundary8h;
                                    y1=p(2,t(j+1-3*(j+1>3),el))-boundary6h;
                                    x2=p(1,t(j+2-3*(j+2>3),el))-boundary8h;
                                    y2=p(2,t(j+2-3*(j+2>3),el))-boundary6h;
                                    Eop0=Eop0+1*sin(k*(x0*cos(theta)+y0*sin
(theta))-2*pi*freq*time(nt))...
                                          *(time(nt)>(x0*cos(theta)+y0*sin
```

```
(theta))/vc);
                                    Eop1=Eop1+1*sin(k*(x1*cos(theta)+y1*sin
(theta))-2*pi*freq*time(nt))...
                                          *(time(nt)>(x1*cos(theta)+y1*sin
(theta))/vc);
                                    if abs(y1-y0)>1e-10 %vertical boundary
                                          Hop=Hop+cos(theta)/eta*sin(k*(x0*cos
(theta)+y2*sin(theta))-2*pi*freq*time(nt))...
                                                *(1-2*(y1>y0))*(time(nt)>
(x0*cos(theta)+y2*sin(theta))/vc);
                                    else %horizontal boundary
                                          Hop=Hop+sin(theta)/eta*sin(k*(x2*cos
(theta)+y0*sin(theta))-2*pi*freq*time(nt))...
                                                *(1-2*(x0>x1))*(time(nt)>
(x2*cos(theta)+y0*sin(theta))/vc);
                                    end;
                              end;
                              if t(4,el)==region(10) && t(4,op)==region(11)
                                    Eop0=-Et(j);
                                    Eop1=-Et(j+1-3*(j+1>3));
                                    Hop=Ht(j);
                              end;
                              if t(4,el)==region(11)
                                    Eop0=0;
                                    Eop1=0;
                                    Hop=0;
                              end;
                        end;
                        temp=eta*(Hop-Ht(j))*3+(Eop0-Et(j))*(1+(i==j));
                        temp=temp+(Eop1-Et(j+1-3*(j+1>3)))*(1+(i==j+1-3*(j+1>3)));
                        temp=temp*l(el,j)/6*(1-(i==j+2-3*(j+2>3)))/(eta+eta);
                        Fe(i)=Fe(i)+temp;
                        temp=(Hop-Ht(j))*2;
                        temp=temp+(1/eta)*(Eop0-Et(j)+Eop1-Et(j+1-3*(j+1>3)));
                        Fh(i)=Fh(i)+temp*l(el,j)/2*(i==j)/(1/eta+1/eta);
                  end;end;
                  %second-order Runge-Kutta method
                      %Runge-Kutta a
                      Erka=1/eps*Met*(Set*Hzt+Fe)-(sx(el)+sy(el))*Et-sx(el)*sy(el)*E1t-
Jt/eps;
                      Hzrka=Mht*((-Sht*Et+Fh)/mu-Mhpt*Hzt-Mhqt*H1t);
                      E1rka=Et;
                      H1rka=Hzt-Mht*Mhwt*H1t;
                      Jt=(Jt+J(el,:,nt+1)')/2;
                      %Runge-Kutta b
                      Erkb=1/eps*Met*(Set*(Hzt+Hzrka*dt/2)+Fe)-(sx(el)+sy(el))*
(Et+Erka*dt/2)-sx(el)*sy(el)*(E1t+E1rka*dt/2)-Jt/eps;
                      Hzrkb=Mht*((-Sht*(Et+Erka*dt/2)+Fh)/mu-Mhpt*(Hzt+Hzrka*dt/2)-Mhqt*
(H1t+H1rka*dt/2));
                      E1rkb=Et+Erka*dt/2;
                      H1rkb=Hzt+Hzrka*dt/2-Mht*Mhwt*(H1t+H1rka*dt/2);
                      %update next values
                      E(el,:,nt+1)=(Et+Erkb*dt)';
                      E1(el,:,nt+1)=(E1t+E1rkb*dt)';
                      H1(el,:,nt+1)=(H1t+H1rkb*dt)';
                      Hz(el,:,nt+1)=(Hzt+Hzrkb*dt)';
                      H(el,:,nt+1)=(Hzt+Hzrkb*dt-Mht*Mhwt*(H1t+H1rkb*dt))';
            end;
      end;
      Edpml=Ed;
```