The implementation of the discontinuous Galerkin method for two-dimensional Maxwell equations in Nektar++

by

Mamadou Baïlo SALL

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Applied Mathematics in the Faculty of Science at Stellenbosch University

Department of Mathematical Sciences,
Mathematics Division,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Supervisor: Dr. Sehun Chun

March 2016

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Signature: .....

Date: March 2016

Abstract

The implementation of the discontinuous Galerkin method for two-dimensional Maxwell equations in Nektar++

Department of Mathematical Sciences,
Mathematics Division,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MSc

May 2015

Maxwell's equations consist of various laws of electromagnetism and can be written in two different modes in two dimensions: the transverse electric(TE) mode and the transverse magnetic(TM) mode. Various methods have been developed in Computational electromagnetics for numerical simulations of electrodynamic applications. In this thesis, a spectral/*hp* discontinuous Galerkin(DG) scheme is implemented for Maxwell's equations in TE as well as in TM polarization, in **Nektar++** a spectral/*hp* object-oriented open-source software. The DG space discretization leads to a semi-discrete scheme to be integrated in time with the Runge-Kutta method, and two numerical fluxes are used to interconnect elements in the mesh namely the centered and the upwind numerical fluxes. To show the *p*-convergence and the *h*-convergence of the scheme, numerical tests in TE and TM modes are performed in linear and isotropic media, followed by an application to the scattering of an electromagnetic wave by a circular cylinder and a rectangular perfect electric conductor. For both modes, the induced current on the surface of the scatterer is computed, using the total field/scattered field formulation.

Uittreksel

Die implementering van die diskontinue Galerkin metode vir tweedimensionele Maxwell vergelykings in Nektar++

Departement Wiskundige Wetenskappe,
Universiteit van Stellenbosch,
Privaatsak X1, Matieland 7602, Suid Afrika.

Tesis: MSc

Mei 2015

Die Maxwell vergelykings bestaan uit verskeie wette van elektromagnetisme, en kan geskryf word in twee verskillende modusse in twee dimensies: die dwars elektriese (DE) modus en die dwars magnetiese (DM) modus. In komputasionele elektromagnetisme is verskeie metodes al ontwikkel om elektrodinamiese toepassings numeries te simuleer. In hierdie tesis word 'n spektrale diskontinue Galerkin (DG) skema geïmplementeer vir Maxwell vergelykings in dwars elektriese sowel as dwars magnetiese polarisasie, in Nektar++, 'n spektrale oopbronsagteware. Die DG ruimtelike diskretisering lei tot 'n semidiskrete skema wat in tyd geïntegreer word met die Runge-Kutta metode, en twee numeriese fluxusse word gebruik om elemente in die maas te interkonnekteer, naamlik die gesentreerde en die "upwind" numeriese fluxusse. Om die p-konvergensie en h-konvergensie van die skema te wys, word numeriese toetse in die DE en DM modusse uitgevoer in lineêre en isotropiese media, gevolg deur 'n toepassing op die verstrooiing van 'n elektromagnetiese golf deur 'n ronde silinder en 'n reghoekige volmaakte elektriese geleier. Vir altwee modusse word die geïnduseerde stroom op die oppervlak van die verstrooier uitgewerk deur gebruik te maak van die totaleveld/verstrooiingsveld formulering.

Acknowledgements

I am deeply grateful to my supervisor Doctor Sehun Chun who tirelessly advised and guided me throughout the duration of the program.
I would also like to thank the African Institute for Mathematical Sciences, all the members of the academic staff and the administrative staff. Special thanks to Prof. Barry Green and Prof. Jeff Sanders for their steadfast commitment.
Finally, I thank my family, my friends and my teachers who taught me starting from grade 1.

Dedications

Hierdie tesis word opgedra aan ...

Contents

List of Figures

List of Tables

Nomenclature

Constants

$\epsilon_0 =$	8.854×10^{-12} Electric permittivity of vacuum	[F/m]
$\mu_0 =$	$4\pi \times 10^{-7}$ Magnetic permeability of vacuum	[H/m]
n_1	Index of refraction of medium 1	
n_2	Index of refraction of medium 2	

Variables

ϵ	Electric permittivity	[F/m]
μ	Magnetic permeability	[H/m]
$\epsilon_\mathbf{r}$	Relative permittivity	[F/m]
$\mu_\mathbf{r}$	Relative permeability	[H/m]
σ	Conductivity of the medium	[S/m]
ρ	Charge density	[C/m^3]

Vectors

$\mathbf{E}$	Electric field
$\mathbf{D}$	Electric flux density
$\mathbf{H}$	Magnetic field
$\mathbf{B}$	Magnetic flux density
$\mathbf{J}$	Surface current density
$\mathbf{u}$	Unknown vector function
$\mathbf{F}$	Flux vector
$\mathbf{F}^*$	Numerical flux
$\mathbf{n}$	Unit normal vector

Matrices

$\mathbf{A}$	Jacobian matrix
$\mathbf{M}$	Mass matrix
$\mathbf{S}$	Stiffness matrix

Operators

$\mathbf{L}$	Differential operator
∇	Gradient operator
$\nabla\cdot$	Divergence operator
$\nabla\times$	Curl operator

Subscripts

x	x coordinate
y	y coordinate
z	z coordinate

Abreviations

PEC	Perfect Electric Conductor
TM	Transverse Magnetic
TE	Transverse Electric
CEM	Computational electromagnetics

Chapter 1

Introduction

Electromagnetism, which studies the electric and magnetic field as the propagation and interaction of charged particles, is one of the largest field in physics which provides many direct and useful applications in our life. By James C. Maxwell (1831-1879), electromagnetism includes the propagation of light, thus the optics can also be regarded as an application of electromagnetism. The classical applications are electrical motors, electrical generators and transformers which transform electrical energy to mechanical energy and vice versa. Other communicational applications are radar systems and antennas which enable wireless communication through electromagnetic waves. Electromagnetic phenomena were expressed in a mathematical way and named after James C. Maxwell. Maxwell's equations consist of various laws of electromagnetism, such as Gauss's law, Faraday's law, and Ampére's law, though only the Ampére's law was modified by James C. Maxwell.

This mathematical model is complex and difficult to solve analytically even for simple cases. Therefore, scientists frequently rely on numerical techniques to adapt Maxwell's equations to real electromagnetic phenomena. The numerical approximation of Maxwell's equations has been thus extensively developed since 1950s, known as computational electromagnetics or **CEM**. Popular methods for **CEM** include: (i) differential-based approach such as the finite difference frequency domain and the finite difference time domain methods, (ii) integral-based approach such as volume integral methods and boundary integral methods, (iii) variational-based approach such as the finite element method and the spectral element method, and (iv) hybrid-based methods as the combination of the aforementioned methods.

1.1 Selected literature review

Over the past few decades, many scientists have contributed to the field of computational electromagnetics by developping algorithms to solve Maxwell's equations both in the time domain and in the frequency domain. In 1966,

Kane Yee [1] presented a finite difference time domain (**FDTD**) scheme to solve Maxwell's equations in isotropic media. In this paper, Yee uses a finite difference approximation of the partial derivatives of the electromagnetic field components with a staggered grid. An application to the scattering of an electromagnetic pulse by a perfect electric conducting cylinder is given in this paper. During the 1970s, the **FDTD** underwent several improvements. Taflove and Brodwin [25] used the Yee scheme to compute the electromagnetic fields in a dielectric scatterer. The diffraction of waves from a plane-wave source is modeled by applying a finite difference scheme to Maxwell's equations in the time domain. In [26], Taflove and Brodwin again modeled the irradiation of a plane-wave into the human eye and its surrounding bony orbit. For two different frequencies, the computation of the electromagnetic fields is carried out with a finite difference algorithm for the solution of time-dependant Maxwell's equations.
In 1992, Jurgens, Taflove, Umashankar and Moore [24] presented a generalization of the **FDTD** called the contour path (**CP**) method in order to model accurately curved surfaces. Despite the difficulty in dealing with corners and edges, they showed that the contour path method is suitable for modeling the illumination of bodies with curved surfaces. They also applied the **CP** method to the scattering of an electromagnetic wave by objects of various shapes in two dimensions. In 1999, Ditkowski, Dridi and Hesthaven [13] studied a convergent Cartesian grid method for the solution of Maxwell's equations in complex geometries. The scheme consists of a staggered grid in space and eliminates problems caused by staircasing. Unlike the Yee scheme, it enforces jump conditions on the field components across material interfaces. The scheme is proven to be of bounded error, thus a convergent and detailed analysis showed that it has a second order global accuracy. The analysis has been validated by test cases in one and two dimensions.

The limitations of the **FDTD** prompted scientists to resort to other numerical methods. Hesthaven and Warburton [14] developed a discontinuous Galerkin method for the time-domain Maxwell's equations which, in principle, is a variant of the finite element method. In this paper, a one-dimensional scheme is presented at first followed by an extension to two dimensions in transverse electric form. Furthermore, in their paper published in 2000, Hesthaven and Warburton discussed a high-order accurate method for the time-domain Maxwell's equations in three dimensions. The computational scheme consists of a domain that is meshed with non-overlapping tetrahedra. The approximate solution is expanded into a nodal Lagrangian basis.

Also related to the field of computational electromagnetics is the paper written by Hassan Fahs [21] who studied the numerical solutions of Maxwell's equations in two dimensions in transverse magnetic polarisation. The computational domain is represented by non-conforming triangular meshes. The resulting numerical flux is approximated by a centered numerical flux and for the time integration, he used a leap frog scheme to advance in time the result-

ing semi-discrete scheme. He made numerical experiments for 2D propagation problems for both inhomogeneous and heterogeneous media.

In 2010, Konig, Busch and Niegemann [22] provided further insight about the discontinuous Galerkin time-domain method for Maxwell's equations in two-dimensional transverse electric mode with anisotropic materials. While authors focused on isotropic and sometimes dispersive materials in other recent papers, Konig, Busch and Niegemann consider anisotropic media by allowing anisotropic permittivity tensors. In order to interconnect neighbouring elements, they used an upwind numerical flux. Like the case of isotropic media they showed that the scheme is convergent for anisotropic materials and simulations of cylindrical cloaking structures have been performed.

Lastly, Hesthaven, Warburton, Chauvière and Wilcox [16] examined a high-order discontinuous Galerkin method for computational electromagnetics and uncertainty quantification. The problem consists of the scattering of an electromagnetic wave by perfectly electric conducting (**PEC**) devices with random shapes and uncertainties in the incident field. The method has been applied to two examples of experiments: the first one is the scattering of a plane wave with normalized frequency($\omega = 1$) from a **PEC** sphere and in the second experiment the **PEC** sphere is replaced by a **PEC** rocket. For both experiments, the radar cross section (**RCS**) of the scattering device is computed.

1.2 Comparison to Yee's scheme

In this thesis, the spectral/hp element discontinuous Galerkin method will be used to solve Maxwell's equations. Comparisons between the spectral element method and the popular Yee's scheme based on the finite difference time domain method will be provided later.

The Yee's algorithm, a finite difference time domain method for Maxwell's equations, was first introduced by Kane Yee in 1966 in his article([1]) entitled "*Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media*" and underwent several improvements in the early 1970s. The Yee's algorithm uses finite difference formulas from Taylor's expansion (see [7] and [8]). The computational domain is divided into a number of cells called Yee cells of dimensions $\Delta x \Delta y \Delta z$. The components of the electric field $\mathbf{E} = (E_x, E_y, E_z)$ and the magnetic field $\mathbf{H} = (H_x, H_y, H_z)$ are located such that $\mathbf{H}$ nodes are situated by half a step with respect to the $\mathbf{E}$ corresponding nodes (see figure 1.1). For instance, H_x is displaced with respect to E_x by $\left(\frac{\Delta x}{2}, \frac{\Delta y}{2}, \frac{\Delta z}{2}\right)$ and so on for H_y and H_z [7].

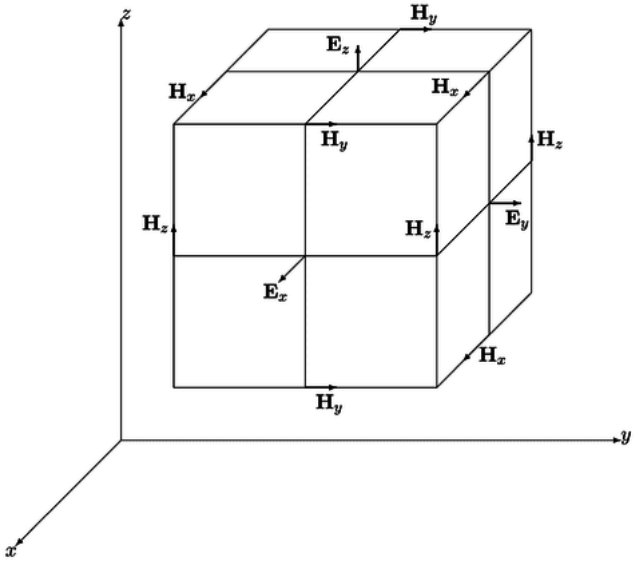


Figure 1.1: Positions of the components of **E** and **H** in a Yee cell.

In the Yee's algorithm, the partial derivatives are approximated by using central difference approximations. Considering $u(x,t)$ the unknown function, the following sampling is adopted:

$$\begin{aligned} u_{k,n} &\approx u(k\Delta x, n\Delta t), \\ u_{k+\frac{1}{2},n+\frac{1}{2}} &\approx u\left((k+\frac{1}{2})\Delta x, (n+\frac{1}{2})\Delta t\right), \end{aligned}$$

where Δx is the spatial step and Δt the time step. Referring to [1], [7] the electric field nodes are located at integer values k and the magnetic field nodes at half integer values of k which leads to a staggered grid approximation. In one dimension the numerical scheme by Yee's scheme can be expressed for the electric field $E(x,t)$ and the magnetic field $H(x,t)$ such as

$$\begin{aligned} \frac{\partial E}{\partial x} &\approx \frac{E_{k+1,n} - E_{k,n}}{\Delta x}, \\ \frac{\partial E}{\partial t} &\approx \frac{E_{k,n+1} - E_{k,n}}{\Delta t}, \\ \frac{\partial H}{\partial x} &\approx \frac{H_{k+\frac{1}{2},n+\frac{1}{2}} - H_{k-\frac{1}{2},n+\frac{1}{2}}}{\Delta x}, \\ \frac{\partial H}{\partial t} &\approx \frac{H_{k+\frac{1}{2},n+\frac{1}{2}} - H_{k+\frac{1}{2},n-\frac{1}{2}}}{\Delta t}. \end{aligned} \tag{1.2.1}$$

To illustrate the use of the Yee's algorithm, we consider the one-dimensional Maxwell equations (2.4.1) in transverse magnetic (**TM**) mode which we will

discuss in greater detail in chapter 2 and chapter 4:

$$\begin{aligned} \mu_0 \frac{\partial H}{\partial t} &= \frac{\partial E}{\partial x}, \\ \epsilon_0 \frac{\partial E}{\partial t} &= \frac{\partial H}{\partial x}. \end{aligned} \tag{1.2.2}$$

We replace the partial derivatives in system (1.2.2) with the finite difference approximations (1.2.1):

$$\begin{aligned} \mu_0 \frac{H_{k+\frac{1}{2},n+\frac{1}{2}} - H_{k+\frac{1}{2},n-\frac{1}{2}}}{\Delta t} &= \frac{E_{k+1,n} - E_{k,n}}{\Delta x}, \\ \epsilon_0 \frac{E_{k,n+1} - E_{k,n}}{\Delta t} &= \frac{H_{k+\frac{1}{2},n+\frac{1}{2}} - H_{k-\frac{1}{2},n+\frac{1}{2}}}{\Delta x} \end{aligned}$$

which in other terms can be written:

$$H_{k+\frac{1}{2},n+\frac{1}{2}} = H_{k+\frac{1}{2},n-\frac{1}{2}} + \frac{1}{\mu_0}\frac{\Delta t}{\Delta x}(E_{k+1,n} - E_{k,n}),$$

$$E_{k,n+1} = E_{k,n} + \frac{1}{\epsilon_0}\frac{\Delta t}{\Delta x}(H_{k+\frac{1}{2},n+\frac{1}{2}} - H_{k-\frac{1}{2},n+\frac{1}{2}}).$$

The algorithm can be summarised by the following steps:

- Replace all the derivatives in Ampère's and Faraday's laws with finite differences both in space and time, so that the electric and magnetic fields are staggered.
- Solve the resulting difference equations to obtain a relation that expresses the unknown future fields($H_{k+\frac{1}{2},n+\frac{1}{2}}$ and $E_{k,n+1}$) in terms of known past fields($H_{k+\frac{1}{2},n-\frac{1}{2}}$ and $E_{k,n}$).
- Evaluate the magnetic fields one time-step into the future($H_{k+\frac{1}{2},n+\frac{1}{2}}$).
- Evaluate the electric fields one time-step into the future($E_{k,n+1}$).
- Repeat the two previous steps until the fields have been determined over the desired duration.

The Yee's algorithm is a versatile method requiring no preprocessing of Maxwell's equations to arrive at the governing equations. Since The Yee's algorithm calculates the **E** and **H** fields everywhere in the computational domain, it provides animated displays of the electromagnetic field movement through the model. This method helps to specify the material at all points within the computational domain. A large variety of dielectric and magnetic materials can be naturally and easily modeled ([35]).

However, the Yee's algorithm has some drawbacks. Since it requires that the entire domain be gridded, and the spatial discretization grid be sufficiently fine to resolve the smallest electromagnetic wavelength and the smallest geometrical feature, the user may end up with large computational domains which result in very long computational running time. The Yee's algorithm computes the **E** and **H** field directly everywhere in the computational domain, thus if one wants to find the values at long distances, this might force the computational domain to be large and this can cause problems related to computer memory ([35]).

Since its introduction in 1966, the Yee's algorithm has become one of the most widespread methods in computational electromagnetics, but it is unable to deal with complex geometries, curved and oblique boundaries. It only works on uniform Cartesian grids. If staircase approximation of boundaries that are not aligned with the grid is used, the computational solution does not yield very accurate results ([11]). This motivated certain scientists to introduce other methods, such as the finite element method and the spectral element method. The spectral element method offers some advantages over the Yee's algorithm. The use of unstructured grids facilitates the handling of complex geometries of the physical domain and allows the application of mesh refinement to increase accuracy without compromising stability. The Galerkin approach in the weak formulation provides an efficient way of handling continuity conditions at the interface of two media. It also provides various choices of basis functions pertaining to p-refinement and the use of different shapes of finite elements in the same mesh. Spectral element methods have the capacity to enforce high order polynomial expansions to achieve fast convergence rates.

1.3 Objective of the study and organisation of the thesis

The main goal of this thesis is to implement the Maxwell solver in the **Nektar++** open-source spectral/hp software library ([34]). **Nektar++** is an efficient spectral element framework permitting users to construct numerical solvers for different kinds of partial differential equations. It is an object-oriented toolkit written in the **C++** language and makes use of prominent attributes of this language such as polymorphism, inheritance, templates, virtual functions and so forth. Furthermore, it possesses all the basic tools required by the finite element method. For starters, it allows one to approximate the domain of study with triangles, quadrilaterals, tetrahedra or hexahedra. It also has sublibraries for polynomial expansion (modal and nodal expansion basis), numerical integration, differentiation and inner products evaluation. Finally, both continuous and discontinuous Galerkin operators are available. The **Nektar++** library is composed of five different sublibraries:

- The standard elemental sublibrary, *StdRegions*
  It allows the user to define the expansion basis only once on the standard region. Subsequently, all the basic operations, such as numerical integration, differentiation are performed on the standard element.

- The parametric mapping sublibrary, *SpatialDomains*
  It contains the classes related to the geometry of the mesh: Geometry1D(segments), Geometry2D(triangles or quadrilaterals) and Geometry3D(tetrahedra or hexahedra).

- The local region sublibrary, *LocalRegions* which encompasses all relevant classes for polynomial expansion in physical space. Those classes are derived from the *StdExpansion* class in the *StdRegions* sublibrary.

- The Multi-regions sublibrary, *MultiRegions* which contains the classes for global region expansion(that is from the local element to the entire domain). In a mathematical point of view this is formulated as:

$$u(x) = \sum_{\tau \in \mathcal{M}} \sum_{i=1}^{N} \hat{u}_i^{\tau} \phi_i^{\tau}(x),$$

  with τ representing the elements in the mesh $\mathcal{M}$.

- The supporting utilities sublibrary *LibUtilities* in which all the operations pertaining to linear algebra, matrix generation, polynomial manipulation are implemented.

For a deeper insight about the **Nektar**++ library and the spectral/hp element method consult [9], [32] and [34].
The remaining part of this thesis is structured as follows: In Chapter 2, we will talk about some preliminaries and general overview about electromagnetic propagation. In Chapter 3, we focus on the concepts of the spectral/hp element discontinuous Galerkin method with polynomial approximation using orthogonal polynomials. We will also discuss some basic concepts about conservation laws and the notion of numerical flux. Finally, in Chapter 4, we will draw up the numerical scheme and present computational results as well as the analysis and conclude in Chapter 5.

Chapter 2

Overview of Electromagnetic propagation

In this chapter, we present Maxwell's equations in differential form in three dimensions, and their reduction in two and one dimension. Two essential polarizations are considered, namely the transverse magnetic mode and the transverse electric mode. We shall also discuss media classification, boundary and interface conditions for electromagnetic problems.

2.1 Maxwell's equations

The theory of electromagnetic wave propagation is mathematically described by Maxwell's equations, which are a system of partial differential equations relating electromagnetic fields and fluxes to sources (currents and charges). These equations can be written in differential form as well as in integral form. For the time varying electromagnetic fields, the Maxwell's equations written in differential form are as follows ([7]):

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\epsilon_0}, \tag{2.1.1}$$

$$\nabla \cdot \mathbf{B} = 0, \tag{2.1.2}$$

$$\nabla \times \mathbf{E} = -\mu_0 \frac{\partial \mathbf{H}}{\partial t}, \tag{2.1.3}$$

$$\nabla \times \mathbf{H} = \mathbf{J} + \epsilon_0 \frac{\partial \mathbf{E}}{\partial t}, \tag{2.1.4}$$

where $\mathbf{E}$ is the electric field intensity, $\mathbf{H}$ is the magnetic field intensity, $\mathbf{B}$ is the magnetic flux density, $\mathbf{J}$ is the electric surface density, ρ is the electric charge density. ϵ_0 and μ_0 are respectively the electric permittivity and the magnetic permeability of free space and t is the time variable.

The equation (2.1.1) is the Gauss's law, (2.1.2) the Gauss's law for magnetism, (2.1.3) the Faraday's law and (2.1.4) the Ampère's law. Maxwell's

equations are general and hold for fields with arbitrary time dependence in any medium and at any location. They reduce to the simpler form for special cases like static case, sinusoidal time varying or time-harmonic fields, and in source-free media.

2.2 Maxwell's equations in three dimensions

We consider a region of space that has no electric or magnetic current sources that is ($\rho = 0$ and $\mathbf{J} = 0$) [8] with ϵ and μ its electric permittivity and magnetic permeability. Furthermore we consider, the Faraday's law and the Ampère's law:

$$\nabla \times \mathbf{E} = -\mu \frac{\partial \mathbf{H}}{\partial t},$$
$$\nabla \times \mathbf{H} = \epsilon \frac{\partial \mathbf{E}}{\partial t}.$$

For each equation, we write the vector components of the curl operators in Cartesian coordinates [8]. This yields the following three scalar equations for the Faraday's law

$$\mu \frac{\partial H_x}{\partial t} = \frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y}, \tag{2.2.1a}$$
$$\mu \frac{\partial H_y}{\partial t} = \frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z}, \tag{2.2.1b}$$
$$\mu \frac{\partial H_z}{\partial t} = \frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x}, \tag{2.2.1c}$$

and the following three scalar equations for the Ampère's law

$$\epsilon \frac{\partial E_x}{\partial t} = \frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z}, \tag{2.2.2a}$$
$$\epsilon \frac{\partial E_y}{\partial t} = \frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x}, \tag{2.2.2b}$$
$$\epsilon \frac{\partial E_z}{\partial t} = \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y}. \tag{2.2.2c}$$

2.3 Reduction to two dimensions

As stated in [8], if the system is uniform in one direction, for instance in the z-direction, then all partial derivatives of the fields with respect to z must vanish. Under that condition, the set of equations (2.2.1) and (2.2.2) reduces to

$$\mu\frac{\partial H_x}{\partial t} = -\frac{\partial E_z}{\partial y}, \tag{2.3.1a}$$

$$\mu\frac{\partial H_y}{\partial t} = \frac{\partial E_z}{\partial x}, \tag{2.3.1b}$$

$$\mu\frac{\partial H_z}{\partial t} = \frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x}, \tag{2.3.1c}$$

and to

$$\epsilon\frac{\partial E_x}{\partial t} = \frac{\partial H_z}{\partial y}, \tag{2.3.2a}$$

$$\epsilon\frac{\partial E_y}{\partial t} = -\frac{\partial H_z}{\partial x}, \tag{2.3.2b}$$

$$\epsilon\frac{\partial E_z}{\partial t} = \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y}. \tag{2.3.2c}$$

2.3.1 TM polarization

Grouping the equations (2.3.1a), (2.3.1b) and (2.3.2c) we obtain the following system:

$$\begin{aligned}
\mu\frac{\partial H_x}{\partial t} &= -\frac{\partial E_z}{\partial y},\\
\mu\frac{\partial H_y}{\partial t} &= \frac{\partial E_z}{\partial x},\\
\epsilon\frac{\partial E_z}{\partial t} &= \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y}.
\end{aligned} \tag{2.3.3}$$

In this system, only H_x, H_y and E_z are involved. This set of field components is called the tranverse magnetic mode in two dimensions.

2.3.2 TE polarization

Now grouping the equations (2.3.2a), (2.3.2b) and (2.3.1c) gives us the following system:

$$\begin{aligned}
\epsilon\frac{\partial E_x}{\partial t} &= \frac{\partial H_z}{\partial y},\\
\epsilon\frac{\partial E_y}{\partial t} &= -\frac{\partial H_z}{\partial x},\\
\mu\frac{\partial H_z}{\partial t} &= \frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x}.
\end{aligned} \tag{2.3.4}$$

Only E_x, E_y and H_z appear in the system. This set of field components is called the tranverse electric mode in two dimensions.
From these two polarizations, we notice that the **TE** and **TM** modes do not have any field component in common. Therefore, they can coexist without influencing each other.

2.4 Reduction to one dimension

2.4.1 TM mode in one dimension

We consider the **TM** polarization in two dimensions, and moreover assume that there is no variation in the y direction. Thus all derivatives with respect to y and z equal to zero. Then we obtain the **TM** mode in one dimension:

$$\begin{aligned} \mu\frac{\partial H_y}{\partial t} &= \frac{\partial E_z}{\partial x}, \\ \epsilon\frac{\partial E_z}{\partial t} &= \frac{\partial H_y}{\partial x}. \end{aligned} \tag{2.4.1}$$

2.4.2 TE mode in one dimension

The same assumptions are considered as before, but considering the **TE** polarization in two dimensions, we recover the one-dimensional **TE** mode:

$$\begin{aligned} \epsilon\frac{\partial E_y}{\partial t} &= -\frac{\partial H_z}{\partial x}, \\ \mu\frac{\partial H_z}{\partial t} &= -\frac{\partial E_y}{\partial x}. \end{aligned} \tag{2.4.2}$$

The one-dimensional scalar wave equation can be derived from either the **TE** mode or the **TM** mode. Indeed considering the system (2.4.2), we differentiate in time the first equation and differentiate in space the second equation. We then obtain

$$\begin{aligned} \epsilon\frac{\partial^2 E_y}{\partial^2 t} &= -\frac{\partial^2 H_z}{\partial t \partial x}, \\ \mu\frac{\partial^2 H_z}{\partial x \partial t} &= -\frac{\partial^2 E_y}{\partial^2 x}. \end{aligned}$$

Since the system is linear, the second derivative of H_z is continuous, thus $\frac{\partial^2 H_z}{\partial t \partial x} = \frac{\partial^2 H_z}{\partial x \partial t}$. Replacing $\frac{\partial^2 H_z}{\partial t \partial x}$ by $\frac{\partial^2 H_z}{\partial x \partial t}$ yields the wave equation with E_y the unknown:

$$\begin{aligned} \frac{\partial^2 E_y}{\partial^2 t} &= \frac{1}{\epsilon\mu}\frac{\partial^2 E_y}{\partial^2 x}, \\ \frac{\partial^2 E_y}{\partial^2 t} &= c^2\frac{\partial^2 E_y}{\partial^2 x}, \end{aligned}$$

where $c^2 = 1/\epsilon\mu$. Similarly, in system (2.4.2), by deriving the first equation in space and the second equation in time we obtain the wave equation for H_z:

$$\frac{\partial^2 H_z}{\partial^2 t} = c^2 \frac{\partial^2 H_z}{\partial^2 x}.$$

Studying these particular modes can be useful because for transverse electromagnetic fields, there is an interesting matching between electric fields and voltage on one side and magnetic fields and current on the other side. For instance, let γ be a coaxial cable with inner radius r_1 and outer radius r_2 such that the voltage between the inner conductor and the outer conductor is U and the current flowing through the cable is I. Subsequently, the electric field and the magnetic field at a distance $r(r_1 < r < r_2)$ are respectively given as follows:

$$\mathbf{E} = \frac{U}{r \ln(\frac{r_2}{r_1})} \mathbf{e_r},$$
$$\mathbf{H} = \frac{I}{2\pi r} \mathbf{e}_\phi.$$

2.4.3 Electrical properties of the media and constitutive relations

We consider a medium characterized by the material parameters such as the electric permittivity (ϵ), the magnetic permeability (μ) and the conductivity (σ). The constitutive relations are relations between the electromagnetic field vectors and the corresponding electromagnetic flux densities. Firstly, the electric field $\mathbf{E}$ and the electric flux density denoted by $\mathbf{D}$ are related by the electric permittivity ϵ. Secondly, the magnetic field $\mathbf{H}$ and the magnetic flux density denoted by $\mathbf{B}$ are related by the electric permeability μ. Furthermore, on the other side, we have the constitutive relation between the surface current density $\mathbf{J}$ and the electric field $\mathbf{E}$ with the conductivity σ of the medium. In vectorial terms, the constitutive relations in a simple medium are

$$\mathbf{D} = \epsilon \mathbf{E}, \tag{2.4.3}$$
$$\mathbf{B} = \mu \mathbf{H}, \tag{2.4.4}$$
$$\mathbf{J} = \sigma \mathbf{E}. \tag{2.4.5}$$

The nature of a medium is determined by its parameters and the constitutive relations. There are different kinds of media encountered in problems involving electromagnetic wave propagation.

Definition 2.1 *A medium is said to be linear if* $\mathbf{D}$, $\mathbf{B}$ *and* $\mathbf{J}$ *vary linearly with* $\mathbf{E}$, $\mathbf{H}$ *and* $\mathbf{E}$ *respectively. In that case* ϵ, μ *and* σ *do not depend on the field amplitudes. Otherwise the medium is nonlinear [12].*

Definition 2.2 *A medium is said to be homogeneous if ϵ, μ and σ are independent of the spatial coordinates. Otherwise it is inhomogeneous [12].*

Definition 2.3 *A medium is isotropic if* $\mathbf{D}$ *is parallel to* $\mathbf{E}$, $\mathbf{B}$ *parallel to* $\mathbf{H}$ *and* $\mathbf{J}$ *parallel to* $\mathbf{E}$. *If not, it is said to be anisotropic. For anisotropic materials ϵ and μ are matrix values [12].*

Generally for any medium, $\epsilon = \epsilon_0\epsilon_\mathbf{r}$ and $\mu = \mu_0\mu_\mathbf{r}$ where $\epsilon_\mathbf{r}$ is the relative electric permittivity and $\mu_\mathbf{r}$ is the relative magnetic permeability. $\epsilon_\mathbf{r}$ and $\mu_\mathbf{r}$ are dimensionless values. Materials are classified according to their electrical properties and most of them are conductors, insulators or semiconductors.

Good conductors are characterized by a high value of σ. Their permittivity and permeability are approximately equal to that of the vacuum. An example of a good conductor in practice is copper with $\sigma = 5.8 \times 10^7 \mathrm{S}.m^{-1}$. The insulators, also known as dielectrics, are materials that have no free charge. The conductivity in a perfect dielectric is equal to zero. Semiconductors are materials in between conductors and insulators. Thus, they present both conducting and insulating properties. In the study of electromagnetic phenomena, one has to bear in mind that the media are often nonperfect, there are losses in any practical media([7]). In this thesis, the different test problems of our study will be focused on linear, homogeneous, isotropic and non-dispersive media in chapter(4).

2.4.4 Boundary and Interface conditions

To obtain the unique solution for electromagnetic problems, necessary boundary conditions are enforced at the boundaries of the medium. For a device consisting of two contiguous media M_1 and M_2, continuity conditions should be applied especially at the interface of these two media. These boundary conditions should be of Dirichlet type(boundary condition related to the value of the unknown function) or Neumann type (boundary condition related to the value of the derivative of the unknown function), homogeneous or inhomogeneous. For problems involving electromagnetics, boundary conditions are derived by applying the integral form of Maxwell's equations to a small region at the interface([17], [5], [12]).
The electric field $\mathbf{E}$ and the magnetic field $\mathbf{H}$ can be separated into two components, one which is tangent to the interface and one normal to the interface. We adopt the following notations: the electromagnetic fields in medium M_1 are $\mathbf{E}_1$ and $\mathbf{H}_1$ with the tangential components E_{t1}, H_{t1} and the normal components E_{n1}, H_{n1}. In medium M_2, we have $\mathbf{E}_2$ and $\mathbf{H}_2$ with the tangential components E_{t2}, H_{t2} and the normal components E_{n2}, H_{n2}. This configuration is illustrated with figure 2.1:

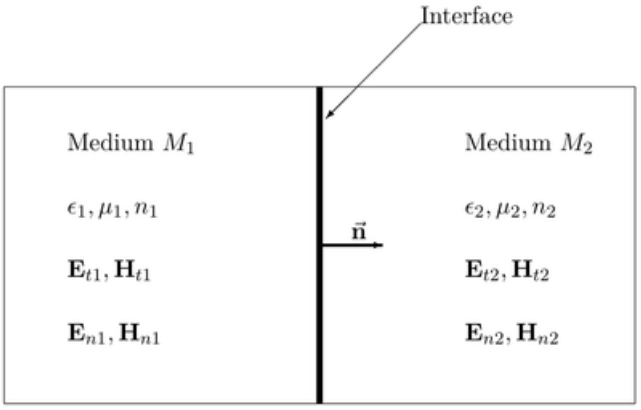


Figure 2.1: Two media separated by an interface with their characteristics.

In vectorial terms, the interface conditions between the two media M_1 and M_2 read:

$$\mathbf{n} \times (\mathbf{E_2} - \mathbf{E_1}) = 0, \tag{2.4.6}$$
$$\mathbf{n} \times (\mathbf{H_2} - \mathbf{H_1}) = \mathbf{J}, \tag{2.4.7}$$
$$\mathbf{n} \cdot (\mathbf{D_2} - \mathbf{D_1}) = \rho, \tag{2.4.8}$$
$$\mathbf{n} \cdot (\mathbf{B_2} - \mathbf{B_1}) = 0, \tag{2.4.9}$$

where the unit vector $\mathbf{n}$ is normal to the interface between the media and points towards the medium 2. The above conditions also can be written in scalar form as follows:

$$E_{t2} - E_{t1} = 0, \tag{2.4.10}$$
$$H_{t2} - H_{t1} = J, \tag{2.4.11}$$
$$D_{n2} - D_{n1} = \rho, \tag{2.4.12}$$
$$B_{n2} - B_{n1} = 0. \tag{2.4.13}$$

The relations (2.4.10) and (2.4.13) state that the tangential component of $\mathbf{E}$ and the normal component of $\mathbf{B}$ are continuous across the interface. The relation (2.4.11) states that at a point of the boundary, the tangential components of $\mathbf{H}$ are discontinuous by the amount equal to J. Finally, the relation (2.4.12) means that at a point of the boundary the difference between the normal component of $\mathbf{D_2}$ that is D_{n2} and the normal component of $\mathbf{D_1}$, D_{n1} amounts to the density of charge ρ.

The interface conditions for the normal and tangential components of the fields between any two media are not independent of each other. If the conditions for the tangential components are satisfied the conditions on the normal components are satisfied. For certain special cases of media, the interface conditions get simplified. For example, if the two media are perfect dielectrics,

then $\rho = 0$, $\mathbf{J} = 0$. Therefore, the tangential components of $\mathbf{E}$ and $\mathbf{H}$ and the normal components of $\mathbf{D}$ and $\mathbf{B}$ are continuous across the interface such as

$$E_{t2} - E_{t1} = 0, \tag{2.4.14}$$
$$H_{t2} - H_{t1} = 0, \tag{2.4.15}$$
$$D_{n2} - D_{n1} = 0, \tag{2.4.16}$$
$$B_{n2} - B_{n1} = 0. \tag{2.4.17}$$

On the other hand, for the medium which is a perfect conductor with $\mathbf{E}$ and $\mathbf{H}$ the electric field and the magnetic field in the medium, the boundary conditions in vectorial form are:

$$\mathbf{n} \times \mathbf{E} = 0, \tag{2.4.18}$$
$$\mathbf{n} \cdot \mathbf{H} = 0, \tag{2.4.19}$$

which in scalar form are:

$$E = 0, \tag{2.4.20}$$
$$H = 0. \tag{2.4.21}$$

2.5 Functional spaces associated with Maxwell's equations

As we have seen in the previous section, Maxwell's equations require that the normal components and the tangential components of the electromagnetic field be well-defined at the boundary and at the interface between two media. Referring to the book written by Peter Monk[18], we give the definitions of some important functional spaces in which these conditions are verified. Let Ω be a bounded and regular domain and $\partial\Omega$ its boundary.

Definition 2.4 $H(\mathrm{div}, \Omega)$
$H(\mathrm{div}, \Omega)$ is the set of vector-valued functions whose divergence is square-integrable:

$$H(\mathrm{div}, \Omega) = \Big\{ \mathbf{v} \in (L^2(\Omega))^3 / \ \nabla \cdot \mathbf{v} \in L^2(\Omega) \Big\},$$

with the norm

$$\|\mathbf{v}\|_{H(\mathrm{div},\Omega)} = \Big(\|\mathbf{v}\|^2_{(L^2(\Omega))^3} + \|\nabla \cdot \mathbf{v}\|^2_{L^2(\Omega)} \Big)^{\frac{1}{2}}.$$

Furthermore, we have:

$$H_0(\mathrm{div}, \Omega) = \Big\{ \mathbf{v} \in H(\mathrm{div}, \Omega) / \ \mathbf{n} \cdot \mathbf{v}_{|\partial\Omega} = 0 \Big\}.$$

Definition 2.5 $H(\text{curl}, \Omega)$
$H(\text{curl}, \Omega)$ is the set of vector-valued functions with square-integrable curl:

$$H(\text{curl}, \Omega) = \Big\{ \mathbf{v} \in (L^2(\Omega))^3 / \ \nabla \times \mathbf{v} \in (L^2(\Omega))^3 \Big\},$$

with the norm

$$\|\mathbf{v}\|_{H(\text{curl},\Omega)} = \Big(\|\mathbf{v}\|^2_{(L^2(\Omega))^3} + \|\nabla \times \mathbf{v}\|^2_{(L^2(\Omega))^3} \Big)^{\frac{1}{2}}.$$

Moreover,

$$H_0(\text{curl}, \Omega) = \Big\{ \mathbf{v} \in H(\text{curl}, \Omega) / \ \mathbf{n} \times \mathbf{v}_{|\partial\Omega} = 0 \Big\}.$$

As we can see, in order to fulfil the conditions (2.4.18), (2.4.19), the electric field **E** and the magnetic field **H** must belong to the spaces $H_0(\text{curl}, \Omega)$ and $H_0(\text{div}, \Omega)$.

2.6 Summary of the chapter

In this chapter, we have learned that the three-dimensional Maxwell's equations can be reduced to two dimensions, and to one dimension in two polarizations the transverse magnetic mode **TM** and the transverse electric mode **TE**. We saw that electromagnetic fields **E** and **H** are respectively related to the electomagnetic flux densities **D** and **B** via the constitutive relations. The definition of a medium depends on these constitutive relations and the electrical properties of the medium ϵ, μ and σ. Thus, a medium can be linear, nonlinear, homogeneous, inhomogeneous, isotropic or anisotropic. We have also learned that to ensure the uniqueness of the solution for electromagnetic problems, boundary conditions must be applied. In this regard, for perfect dielectrics the fields are continuous across the interface and for perfect electric conductors the tangential component of **E** and the normal component of **H** are equal to zero at the boundary of the domain.
In chapter(4), we will be considering test problems in **TM** mode as well as in **TE** mode in a simple medium(that is linear, isotropic) subject to perfect electric conductor boundary conditions($E = 0$ and $H = 0$).

Chapter 3

The spectral/hp element method and the notion of numerical flux

The spectral/hp element method is a high order finite element method combining the attributes of the h-version and the p-version of the finite element method. For the h-version, the degree p of the polynomial basis functions is fixed and convergence is achieved by reducing the size of the mesh h (i.e increasing the number of elements). For the p-version, the size of the mesh is kept fixed and the discretization is performed by increasing the order of the polynomials in the elements. The spectral element method uses a high order polynomial approximation by a truncated series of global functions by employing the Fourier, Legendre or Chebychev polynomials.

3.1 The Galerkin formulation

In order to obtain the weak formulation of the partial differential equation, classical finite element methods use the Galerkin variational formulation. Consider a boundary value problem in a domain Ω

$$\mathbf{L}(u) = f \tag{3.1.1}$$

with appropriate boundary conditions, where $\mathbf{L}$ is a differential operator. The weak form is obtained by multiplying (3.1.1) by a regular test function v and integrating over the domain Ω. The problem is equivalent to :

$$\text{Find} \quad u \in U \quad \text{such that} \quad \int_\Omega \mathbf{L}(u) v dx = \int_\Omega f v dx, \quad \forall v \in V, \tag{3.1.2}$$

where U and V are the space of solutions and the space of test functions, respectively. Note that, for the classical Galerkin finite element method, U and V are identical. Set $a(u,v) = \int_\Omega \mathbf{L}(u) v dx$ and $l(v) = \int_\Omega f v dx$, where $a(.,.)$

is a bilinear form and l a linear form. Then, (3.1.2) becomes

$$\text{Find} \quad u \in U \quad \text{such that} \quad a(u,v) = l(v), \quad \forall v \in U.$$

Since U is an infinite dimensional space, one must look for an approximate solution in the reduced finite dimensional space $U^\delta \subset U$. The space U^δ is assumed to be spanned by basis functions $(\phi_i)_{1\leq i\leq N}$. Then, any element u^δ of U^δ can be represented by a linear combination of the basis functions ϕ_i that is:

$$u^\delta = \sum_{i=1}^{N} \hat{u}_i \phi_i.$$

Taking v to be each of the basis functions ϕ_j, $1 \leq j \leq N$, we derive the following discrete problem:

$$\text{find} \quad \hat{u}_i \quad \text{such that} \quad \sum_{i=1}^{N} a(\phi_i, \phi_j)\hat{u}_i = (f, \phi_j),$$

which can be written in the form of a linear system:

$$\mathbf{A}\hat{\mathbf{u}} = \hat{\mathbf{f}},$$

where $\hat{\mathbf{u}}$ is a vector whose components are the coefficients $\hat{u}_i$, $\mathbf{A}$ and $\hat{\mathbf{f}}$ are repectively defined by

$$\mathbf{A}_{ij} \equiv a(\phi_i, \phi_j) = \int_\Omega \mathbf{L}(\phi_i)\phi_j dx,$$
$$\hat{\mathbf{f}}_j \equiv (f, \phi_j) = \int_\Omega f\phi_j dx.$$

3.2 Conservation laws

Conservation laws are systems of partial differential equations which can be written in the form([10]):

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{u})}{\partial x} = 0, \tag{3.2.1}$$

where

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{pmatrix}, \quad \mathbf{F}(\mathbf{u}) = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{pmatrix}.$$

The variable $\mathbf{u}$ is called the vector of conserved variables and $\mathbf{F} = \mathbf{F}(\mathbf{u})$ is the vector of fluxes. Each of f_i is a function of the components u_j of $\mathbf{u}$.

Definition 3.1 (Jacobian matrix) *The Jacobian of the flux function* $\mathbf{F}(\mathbf{u})$ *is the matrix*

$$\mathbf{A}(\mathbf{u}) = \begin{pmatrix} \frac{\partial f_1}{\partial u_1} & \cdots & \frac{\partial f_1}{\partial u_m} \\ \frac{\partial f_2}{\partial u_1} & \cdots & \frac{\partial f_2}{\partial u_m} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_m}{\partial u_1} & \cdots & \frac{\partial f_m}{\partial u_m} \end{pmatrix}.$$

Applying the chain rule to $\frac{\partial \mathbf{F}(\mathbf{u})}{\partial x}$ gives

$$\frac{\partial \mathbf{F}(\mathbf{u})}{\partial x} = \frac{\partial \mathbf{F}(\mathbf{u})}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial x} = \mathbf{A}(\mathbf{u}) \frac{\partial \mathbf{u}}{\partial x},$$

then (3.2.1) becomes

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{A}(\mathbf{u}) \frac{\partial \mathbf{u}}{\partial x} = 0. \tag{3.2.2}$$

Definition 3.2 *The eigenvalues* λ_i *of a matrix* $\mathbf{A}$ *are the solutions of the characteristic polynomial*

$$|\mathbf{A} - \lambda \mathbf{I}| = det(\mathbf{A} - \lambda \mathbf{I}) = 0,$$

with $\mathbf{I}$ *the identity matrix.*

In the equation (3.2.2), the eigenvalues of the matrix $\mathbf{A}(\mathbf{u})$ are called eigenvalues of the system (3.2.2). In the physical sense, the eigenvalues stand for the speed of propagation of information.
For a detailed analysis and examples of conservation laws, one can consult ref.[10].

3.3 Discontinuous Galerkin approach

The discontinuous Galerkin(DG) method is a numerical method which encompasses features of the finite element method and the finite volume method. It is applied in solving several types of partial differential equations namely the shallow water equations, the Navier-Stokes equations, the equations in magnetohydrodynamics, the Maxwell's equations and so forth. The DG method is a high order accurate method. It can deal with complex geometries, since it enables one to use meshes with any shape. Also, the DG method can handle

both space and time discretisation. In this section, we present a general formulation of the DG method in space discretisation which will be used in the following chapter for Maxwell's equations.

Let us consider the following hyperbolic system in conservative form: for $x \in \partial\Omega$

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{u}) &= 0 \\ \mathbf{u}(x,t) &= g(x,t), \\ \mathbf{u}(x,0) &= f(x), \end{cases} \tag{3.3.1}$$

where $\mathbf{u}$ is the unknown function and $\mathbf{F}(\mathbf{u})$ the flux vector. Multiplying (3.3.1) by a test function Ψ and integrating over the domain Ω we obtain

$$\int_\Omega \left(\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{u}) \right) \Psi \mathrm{d}x = 0,$$
$$\int_\Omega \frac{\partial \mathbf{u}}{\partial t} \Psi \mathrm{d}x + \int_\Omega \nabla \cdot \mathbf{F}(\mathbf{u}) \Psi \mathrm{d}x = 0.$$

Using integration by parts we transform the second term

$$\int_\Omega \nabla \cdot \mathbf{F}(\mathbf{u}) \Psi \mathrm{d}x = \int_{\partial\Omega} \Big(\mathbf{n} \cdot \mathbf{F}(\mathbf{u}) \Big) \Psi \mathrm{d}\sigma - \int_\Omega \mathbf{F}(\mathbf{u}) \cdot \nabla \Psi \mathrm{d}x,$$

then

$$\int_\Omega \left(\frac{\partial \mathbf{u}}{\partial t} \Psi - \mathbf{F}(\mathbf{u}) \cdot \nabla \Psi \right) \mathrm{d}x = - \int_{\partial\Omega} \Big(\mathbf{n} \cdot \mathbf{F}(\mathbf{u}) \Big) \Psi \mathrm{d}\sigma.$$

Now we partition the domain Ω in K non overlapping elements Ω_k such as $\Omega \approx \Omega^h = \bigcup_{k=1}^{K} \Omega_k$. Therefore, in each element Ω_k the formulation of the problem is

$$\int_{\Omega_k} \left(\frac{\partial \mathbf{u}}{\partial t} \Psi - \mathbf{F}(\mathbf{u}) \cdot \nabla \Psi \right) \mathrm{d}x = - \int_{\partial\Omega_k} \Big(\mathbf{n} \cdot \mathbf{F}(\mathbf{u}) \Big) \Psi \, \mathrm{d}\sigma, \tag{3.3.2}$$

where $\mathbf{n}$ is the unit normal vector pointing from Ω_{k_1} to a neighbouring element Ω_{k_2}.

We define the finite element space of discontinuous functions

$$V_h = \left\{ \mathbf{u}_h \in L^2(\Omega) : \mathbf{u}_h^k \in V(\Omega_k) \ \ \forall \Omega_k \right\}, \tag{3.3.3}$$

where $\mathbf{u}_h^k$ is the restriction of $\mathbf{u}_h$ into Ω_k and $V(\Omega_k)$ the local space which can be identified to $\mathcal{P}_p(\Omega_k)$ the space of polynomials of degree p.

Since we are dealing with a discontinuous Galerkin method, one must connect the local solution $\mathbf{u}_h^{k_1}$ in an element Ω_{k_1} to that of a neighbouring element Ω_{k_2} let say $\mathbf{u}_h^{k_2}$. To achieve this, we use a numerical flux which enables to reconstruct the global solution from the local solutions. It is a function of $\mathbf{u}_h^{k_1}$ and $\mathbf{u}_h^{k_2}$. In the next section we will talk in details about the numerical flux.

3.4 Notion of numerical flux

Definition 3.3 (Flux) *The flux is the rate of flow of a physical property through a surface.*

Definition 3.4 (Numerical flux) *The numerical flux is the flux through the interface between two elements following the normal direction.*

As stated in the previous section, the discontinuous Galerkin method includes a space discretisation with the definition of a finite element space with discontinuous functions. On each space $V(\Omega_k)$, the local solution is expressed as a polynomial of order N. With respect to the polynomial basis, we can try either a modal expansion or a nodal expansion. For the modal expansion, the user may have

$$\forall\ x \in \Omega_k, \quad \mathbf{u}_h^k(x,t) = \sum_{n=1}^{N} \hat{\mathbf{u}}_n^k \Psi_n(x),$$

where $(\Psi_n(x))_{n=1}^N$ is either a Fourier polynomial basis, a Legendre polynomial basis or a Chebychev polynomial basis. For the nodal expansion, we have

$$\mathbf{u}_h^k(x,t) = \sum_{i=1}^{N} \mathbf{u}_i^k(t) l_i^k(x),$$

where $\mathbf{u}_i^k(t) = \mathbf{u}^k(x_i,t)$ and the polynomials l_i^k represent the Lagrange polynomials. Then, in order to obtain a global approximation of the exact solution, we need to combine all local solutions on the elements. As mentioned in the previous section, this is achieved by using the numerical flux which connects all local elementwise solutions to obtain the global solution. To achieve this, we consider the equation (3.3.2) with the right hand side where we have the integral on the boundary $\partial\Omega_k$. We denote the numerical flux by $\mathbf{F}^*$ and since the numerical flux depends on $\mathbf{u}_h^{k_1}$ and $\mathbf{u}_h^{k_2}$ we have

$$\int_{\Omega_{k_1}} \left(\frac{\partial \mathbf{u}_h^{k_1}}{\partial t} \Psi_h - \mathbf{F}(\mathbf{u}_h^{k_1}) \cdot \nabla \Psi_h \right) \, \mathrm{d}x = - \int_{\partial\Omega_{k_1}} \Big(\mathbf{n} \cdot \mathbf{F}^*(\mathbf{u}_h^{k_1}, \mathbf{u}_h^{k_2}) \Big) \Psi_h \, \mathrm{d}\sigma.$$

Integrating by parts again, we obtain the strong form

$$\int_{\Omega_{k_1}} \left(\frac{\partial \mathbf{u}_h^{k_1}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{u}_h^{k_1}) \right) \Psi_h \, \mathrm{d}x = \int_{\partial\Omega_{k_1}} \mathbf{n} \cdot \Big(\mathbf{F}(\mathbf{u}_h^{k_1}) - \mathbf{F}^*(\mathbf{u}_h^{k_1}, \mathbf{u}_h^{k_2}) \Big) \Psi_h \, \mathrm{d}\sigma.$$

Between two neighbouring elements the functions $\mathbf{u}_h^k$ can have different values on the interface $e = \partial\Omega_{k_1} \cap \partial\Omega_{k_2}$. Therefore, we designate the interior value of $\mathbf{u}_h^{k_1}$ by $\mathbf{u}_h^-$, the exterior value by $\mathbf{u}_h^+$ and the numerical flux by $\mathbf{F}^*(\mathbf{u}_h^{k_1}) = \mathbf{F}^*(\mathbf{u}_h^-, \mathbf{u}_h^+)$. The information through the interface between the two elements is carried along the unit normal vector $\mathbf{n}$. That configuration is depicted on figure 3.1:

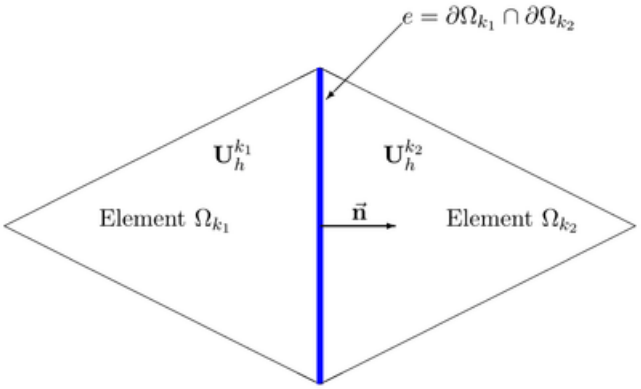


Figure 3.1: Two neighboring elements sharing the same edge e.

$$\mathbf{F}^*_{\mathbf{n}}(\mathbf{u}_h^-, \mathbf{u}_h^+) = \mathbf{n} \cdot \mathbf{F}^*(\mathbf{u}_h^-, \mathbf{u}_h^+).$$

3.5 Computation of the numerical flux

Consider the following system of conservation law with one initial condition and two boundary conditions

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} + \dfrac{\partial \mathbf{F}(\mathbf{u})}{\partial x} &= 0, \\ \mathbf{u}(x,0) &= \mathbf{u}_0(x), \\ \mathbf{u}(0,t) &= \mathbf{u}_l(t), \\ \mathbf{u}(d,t) &= \mathbf{u}_r(t). \end{cases} \tag{3.5.1}$$

In this problem, we consider a time domain $[0,T]$ and a spatial domain $[0,d]$ with T and d positive values. $\mathbf{u}_0$ is a function of the variable x, $\mathbf{u}_l$ and $\mathbf{u}_r$ functions of the temporal variable t. In this case, we use two-state initial conditions as illustrated in page 317 of ref. [10]

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} + \dfrac{\partial \mathbf{F}(\mathbf{u})}{\partial x} &= 0, \\ \mathbf{u}(x,0) &= \mathbf{u}_L, \quad \text{if } x < 0, \\ \mathbf{u}(x,0) &= \mathbf{u}_R, \quad \text{if } x > 0, \end{cases} \tag{3.5.2}$$

where $\mathbf{u}_L$ and $\mathbf{u}_R$ are given constant values. At $t = 0$, $\mathbf{u}$ takes the value $\mathbf{u}_L$ for negative x and $\mathbf{u}_R$ for positive x. As in finite volume methods, the spatial domain is split into N finite volumes $I_i = \left[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}\right]$, $1 \leq i \leq N$ of size

$\Delta x = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}$ and the temporal domain into M intervalls $K_p = [t_p, t_{p+1}]$, $1 \leq p \leq M$ of size $\Delta t = t_{p+1} - t_p$.
In this way for each i and p we define a control volume $V = I_i \times K_p = \left[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}\right] \times [t_p, t_{p+1}]$.

Let us now integrate the equation (3.5.2) over the control volume V we obtain

$$\int_{t_p}^{t_{p+1}} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \frac{\partial \mathbf{u}}{\partial t} dx dt + \int_{t_p}^{t_{p+1}} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \frac{\partial \mathbf{F}(\mathbf{u})}{\partial x} dx dt = 0$$

,

$$\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \Big(\mathbf{u}(x, t_{p+1}) - \mathbf{u}(x, t_p)\Big) dx = \int_{t_p}^{t_{p+1}} \Big(\mathbf{F}(\mathbf{u}(x_{i-\frac{1}{2}}, t)) - \mathbf{F}(\mathbf{u}(x_{i+\frac{1}{2}}, t))\Big)\, dt$$

,

$$\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \mathbf{u}(x, t_{p+1}) dx - \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \mathbf{u}(x, t_p) dx = \int_{t_p}^{t_{p+1}} \mathbf{F}(\mathbf{u}(x_{i-\frac{1}{2}}, t)) dt - \int_{t_p}^{t_{p+1}} \mathbf{F}(\mathbf{u}(x_{i+\frac{1}{2}}, t)) dt$$

,

$$\Delta x \left[\frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \mathbf{u}(x, t_{p+1}) dx - \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \mathbf{u}(x, t_p) dx\right] = \Delta t \left[\frac{1}{\Delta t} \int_{t_p}^{t_{p+1}} \mathbf{F}(\mathbf{u}(x_{i-\frac{1}{2}}, t)) dt - \frac{1}{\Delta t} \int_{t_p}^{t_{p+1}} \mathbf{F}(\mathbf{u}(x_{i+\frac{1}{2}}, t)) dt\right].$$

Introducing $\mathbf{u}_i^p$, $\mathbf{u}_i^{p+1}$, $\mathbf{F}_{i-\frac{1}{2}}$ and $\mathbf{F}_{i+\frac{1}{2}}$ as integral averages as follows

$$\begin{aligned}
\mathbf{u}_i^p &= \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \mathbf{u}(x, t_p) dx, \\
\mathbf{u}_i^{p+1} &= \frac{1}{\Delta x} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \mathbf{u}(x, t_{p+1}) dx, \\
\mathbf{F}_{i-\frac{1}{2}} &= \frac{1}{\Delta t} \int_{t_p}^{t_{p+1}} \mathbf{F}(\mathbf{u}(x_{i-\frac{1}{2}}, t)) dt, \\
\mathbf{F}_{i+\frac{1}{2}} &= \frac{1}{\Delta t} \int_{t_p}^{t_{p+1}} \mathbf{F}(\mathbf{u}(x_{i+\frac{1}{2}}, t)) dt,
\end{aligned}$$

we end up with the following formula

$$\mathbf{u}_i^{n+1} = \mathbf{u}_i^n + \frac{\Delta t}{\Delta x} \left(\mathbf{F}_{i-\frac{1}{2}} - \mathbf{F}_{i+\frac{1}{2}}\right). \tag{3.5.3}$$

Let x_L and x_R be two values in the spatial domain with $x_L < x_R$ and T a given time. Integrating (3.5.2) over $[x_L, x_R] \times [0, T]$ gives

$$\int_{x_L}^{x_R} \mathbf{u}(x,T)dx = \int_{x_L}^{x_R} \mathbf{u}(x,0)dx + \int_0^T \mathbf{F}\Big(\mathbf{u}(x_L,t)\Big)dt - \int_0^T \mathbf{F}\Big(\mathbf{u}(x_R,t)\Big)dt,$$

Or,

$$\int_{x_L}^{x_R} \mathbf{u}(x,T)dx = x_R\mathbf{u}_R - x_L\mathbf{u}_L + T(\mathbf{F}_L - \mathbf{F}_R). \tag{3.5.4}$$

Moreover,

$$\begin{aligned}
\int_{x_L}^{x_R} \mathbf{u}(x,T)dx &= \int_{x_L}^{TS_L} \mathbf{u}(x,T)dx + \int_{TS_L}^{TS_R} \mathbf{u}(x,T)dx + \int_{TS_R}^{x_R} \mathbf{u}(x,T)dx,\\
\int_{x_L}^{x_R} \mathbf{u}(x,T)dx &= \int_{TS_L}^{TS_R} \mathbf{u}(x,T)dx + (TS_L - x_L)\mathbf{u}_L + (x_R - TS_R)\mathbf{u}_R,\\
\int_{x_L}^{x_R} \mathbf{u}(x,T)dx &= \int_{TS_L}^{TS_R} \mathbf{u}(x,T)dx + TS_L\mathbf{u}_L - x_L\mathbf{u}_L + x_R\mathbf{u}_R - TS_R\mathbf{u}_R.
\end{aligned} \tag{3.5.5}$$

The equations (3.5.4) and (3.5.5) yield

$$\begin{aligned}
\int_{TS_L}^{TS_R} \mathbf{u}(x,T)dx &= T(S_R\mathbf{u}_R - S_L\mathbf{u}_L + \mathbf{F}_L - \mathbf{F}_R),\\
\frac{1}{T(S_R - S_L)}\int_{TS_L}^{TS_R} \mathbf{u}(x,T)dx &= \frac{S_R\mathbf{u}_R - S_L\mathbf{u}_L + \mathbf{F}_L - \mathbf{F}_R}{S_R - S_L},
\end{aligned}$$

$\mathbf{u}^{HLL}$, which is evaluated as $\dfrac{S_R\mathbf{u}_R - S_L\mathbf{u}_L + \mathbf{F}_L - \mathbf{F}_R}{S_R - S_L}$, is the integral average of the exact solution of the Riemann problem (3.5.2) and plays an important role in determining the numerical flux. The abreviation HLL stands for Harten Lax and van Leer. For more details about numerical fluxes, one can consult the book written by Toro([10]).

Definition 3.5 (Rankine Hugoniot condition) *Considering a discontinuous wave solution of the conservation law (3.5.2) with speed S, the Rankine Hogoniot condition expresses that $\mathbf{F}_R - \mathbf{F}_L = \Delta\mathbf{F} = S\Delta\mathbf{u} = S(\mathbf{u}_R - \mathbf{u}_L)$ where $\mathbf{F}_R = \mathbf{F}(\mathbf{u}_R)$ and $\mathbf{F}_L = \mathbf{F}(\mathbf{u}_L)$.*

After applying the Rankine Hugoniot condition between the states $\mathbf{u}^{HLL}$ and $\mathbf{u}_L$ with the speed S_L and between $\mathbf{u}^{HLL}$ and $\mathbf{u}_R$ with the speed S_R, we obtain the following relations

$$\begin{aligned}
\mathbf{F}^{HLL} &= \mathbf{F}_L + S_L(\mathbf{u}^{HLL} - \mathbf{u}_L),\\
\mathbf{F}^{HLL} &= \mathbf{F}_R + S_R(\mathbf{u}^{HLL} - \mathbf{u}_R).
\end{aligned}$$

Summing these two relations and replacing $\mathbf{u}^{HLL}$ by its value, we end up with the following formula for the numerical flux

$$\mathbf{F}^{HLL} = \frac{S_R\mathbf{F}_L - S_L\mathbf{F}_R + S_LS_R(\mathbf{u}_R - \mathbf{u}_L)}{S_R - S_L}. \tag{3.5.6}$$

Depending on the choice of S_L and S_R, we can have different numerical fluxes. For instance, taking $S_L = -S$ and $S_R = S$ with S a positive speed, we obtain a Rusanov numerical flux

$$\mathbf{F}_{i+\frac{1}{2}} = \frac{1}{2}(\mathbf{F}_L + \mathbf{F}_R) + \frac{1}{2}S(\mathbf{u}_L - \mathbf{u}_R).$$

A Lax-Friedrichs numerical flux is recovered by choosing S as the biggest eigenvalue of the Jacobian matrix $\mathbf{A}(\mathbf{u})$. The simple case of the Lax-Friedrichs numerical flux is the central case where we consider only the average of the two values in the two neighbouring elements, that is

$$\mathbf{F}^{\text{aver}}_{i+\frac{1}{2}} = \frac{1}{2}(\mathbf{F}_L + \mathbf{F}_R).$$

The other case is a flux which takes information where it is coming, called an upwind flux which is

$$\mathbf{F}_{i+\frac{1}{2}} = \frac{1}{2}(\mathbf{F}_L + \mathbf{F}_R) + \frac{1}{2}|\lambda_{\max}|(\mathbf{u}_L - \mathbf{u}_R).$$

3.6 The Runge-Kutta fourth order method

Since we are dealing with conservation laws, our model depends on the time as well as on the space. After discretizing in space using the discontinuous Galerkin approach, we obtain a semi-discrete scheme which has to be integrated in time ([29]). For the time stepping, we shall use the Runge-Kutta fourth order method which is implemented in the **Nektar**++ open source library. Consider the following initial value problem

$$\begin{cases} u' &= f(t,u),\ a \le t \le b, \\ u(a) &= \alpha. \end{cases} \tag{3.6.1}$$

The Runge-Kutta method consists of a series of algorithms of increasing order. It has a desirable property of high order local truncation error and does not require the computation and the evaluation of the derivatives of $f(t,u)$ which is tedious and time-consuming for most problems. There are different kinds of Runge-Kutta methods. The most common in use is that of order four which is

enunciated as follows with $\mathbf{\Gamma}$ the approximation of u, α the initial value, and Δt the time step:

$$\begin{aligned}
\mathbf{\Gamma}_0 &= \alpha \\
k_1 &= \Delta t f(t_i, \mathbf{\Gamma}_i) \\
k_2 &= \Delta t f(t_i + \frac{\Delta t}{2}, \mathbf{\Gamma}_i + \frac{k_1}{2}) \\
k_3 &= \Delta t f(t_i + \frac{\Delta t}{2}, \mathbf{\Gamma}_i + \frac{k_2}{2}) \\
k_4 &= \Delta t f(t_{i+1}, \mathbf{\Gamma}_i + k_3) \\
\mathbf{\Gamma}_{i+1} &= \mathbf{\Gamma}_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}$$

for $i = 0, \cdots, N-1$.

We end this chapter by giving a stability result and two L^2-error estimates(from ref.[19]) in the linear case($F(u) = cu$, c constant) for the Runge-Kutta discontinuous Galerkin discretization. We consider the problem (3.5.1) with appropriate initial and boundary conditions, k the degree of approximating polynomials.

Proposition 3.6 *Let u_h be the approximation of u and u_0 the initial condition. Then it holds,*

$$\frac{1}{2}\|u_h(T)\|^2_{L^2(\Omega)} + \Theta_T(u_h) \leq \frac{1}{2}\|u_0\|^2_{L^2(\Omega)},$$

where $\Theta_T(u_h)$ depends on the jumps across the interfaces.

Theorem 3.7 *Let u_h be the approximation of u and suppose that $u_0 \in H^{k+1}(\Omega)$. Then we have,*

$$\|u(T) - u_h(T)\|_{L^2(\Omega)} \leq C|u_0|_{H^{k+1}(\Omega)}(\Delta x)^{k+\frac{1}{2}},$$

where C depends on k, $|c|$ and T.

Theorem 3.8 *Let u_h be the approximation of u and suppose that $u_0 \in H^{k+2}(\Omega)$. Then we have,*

$$\|u(T) - u_h(T)\|_{L^2(\Omega)} \leq C|u_0|_{H^{k+2}(\Omega)}(\Delta x)^{k+1},$$

where C depends on k, $|c|$ and T.

$H^m(\Omega)$ is the Sobolev space $W^{m,2}(\Omega) = \{u \in L^2(\Omega) : \ \forall |\alpha| \leq m, \ D^\alpha u \in L^2(\Omega)\}$. The proofs of the stability and error estimates can be found in page 106, 108 and 111 of ref.[19].

3.7 Summary of the chapter

In this chapter, we learned that the finite element method and the spectral/hp element method are linked and the latter can be seen as a high order h-p finite element method which uses orthogonal polynomials for the approximation. We carried out the variational formulation of a conservation law with the discontinuous Galerkin approach. Indeed, given that there are discontinuities at the interface between neighbouring elements one needs to apply a numerical flux $\mathbf{F}^*$ which is in charge of connecting the local solutions. There are different kinds of numerical flux among which we have the Lax-Friedrichs numerical flux. Finally, we introduced the fourth order Runge-Kutta method for the time integration.
In chapter(4), we shall use the DG approach for the space discretisation and in the code, the fourth order Runge-Kutta method for time-stepping. As numerical flux, we shall consider the two cases of Lax-Friedrichs numerical flux: the average and the upwind numerical flux.

Chapter 4

Numerical scheme and computational results

In this chapter, we shall draw up the numerical scheme for a variety of test problems and present the computational results for the p-convergence as well as for the h-convergence. We begin with a test problem in one dimension with two different cases: a domain composed of two contiguous media with index of refraction $n_1 = 1$ and $n_2 = 1.5$. The second case consists of a homogeneous medium with index of refraction $n = 1$. Secondly, for the two-dimensional case, we present numerical results for two test problems: one in transverse magnetic mode and one in transverse electric mode. In the last part of this chapter, we shall give an application of the **2D** test problems to the scattering of an electromagnetic wave by a circular cylinder and a rectangular object with **PEC** boundary conditions for both **TM** and **TE** modes. We implemented the Maxwell solver in the **Nektar++** software library. The code is structured into four essential parts: the definition of initial conditions, the definition of boundary conditions, the definition of flux vectors for the different test problems *Maxwell1D*, *Maxwell2DTM* and *Maxwell2DTE*, and finally the implementation of the numerical fluxes namely the centered numerical flux and the upwind numerical flux. The *cpp* file of the code can be found in appendix (C).

4.1 The one-dimensional scheme

Before tackling the problem in two dimensions, we shall study in this section Maxwell's equations in one dimension. In that study, we consider the system (2.4.2) and for sake of notation we replace E_y by E and H_z by H. Then (2.4.2)

becomes

$$\begin{cases} \epsilon \dfrac{\partial E}{\partial t} &= -\dfrac{\partial H}{\partial x}, \\ \mu \dfrac{\partial H}{\partial t} &= -\dfrac{\partial E}{\partial x}, \end{cases} \tag{4.1.1}$$

where E is the electric field, H the magnetic field; ϵ and μ represent respectively the electric permittivity and the magnetic permeability. We write (4.1.1) in the form of a conservation law as follows:

$$\begin{cases} \epsilon \dfrac{\partial E}{\partial t} + \dfrac{\partial H}{\partial x} &= 0, \\ \mu \dfrac{\partial H}{\partial t} + \dfrac{\partial E}{\partial x} &= 0, \end{cases} \tag{4.1.2}$$

which we can write in the following

$$\begin{pmatrix} \epsilon & 0 \\ 0 & \mu \end{pmatrix} \frac{\partial}{\partial t} \begin{pmatrix} E \\ H \end{pmatrix} + \frac{\partial}{\partial x} \begin{pmatrix} H \\ E \end{pmatrix} = 0,$$

$$\mathbf{Q} \frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{u})}{\partial x} = 0, \tag{4.1.3}$$

where $\mathbf{Q} = \begin{pmatrix} \epsilon & 0 \\ 0 & \mu \end{pmatrix}$, $\mathbf{u} = \begin{pmatrix} E \\ H \end{pmatrix}$ and $\mathbf{F}(\mathbf{u}) = \begin{pmatrix} H \\ E \end{pmatrix}$.
$\mathbf{Q}$ contains the materials, $\mathbf{u}$ the unknown fields and $\mathbf{F}(\mathbf{u})$ the flux. For this problem, we consider $\Omega = [x_0, x_N]$, $x_N = x_0 + L$, $L > 0$ with PEC boundary conditions [13] and [14], i.e,

$$E(x_0, t) = E(x_N, t) = 0.$$

4.1.1 Formulation of the problem

We consider the governing equation (4.1.3) which we multiply by a test function ψ_i. Integrating by parts on each element Ω_k gives:

$$\int_{\Omega_k} \mathbf{Q} \frac{\partial \mathbf{u}}{\partial t} \psi_i(x) \, \mathrm{d}x - \int_{\Omega_k} \mathbf{F}(\mathbf{u}) \frac{\partial \psi_i(x)}{\partial x} \, \mathrm{d}x = - \int_{\partial \Omega_k} \mathbf{F}(\mathbf{u}) \psi_i(x) \, \mathrm{d}x, \tag{4.1.4}$$

At the interface $\partial\Omega_k$, the flux $\mathbf{F}(\mathbf{u})$ is replaced by a numerical flux $\mathbf{F}^* = \mathbf{F}(\mathbf{u}^-, \mathbf{u}^+)$. As we said in chapter 3, $\mathbf{F}^*$ is responsible for interconnecting the neighbouring elements, with $\mathbf{u}^-$ the value of the solution in the local element

and $\mathbf{u}^+$ that of the solution in the contiguous element. Within each element the solution is expanded in a polynomial basis $(\psi_j)_{1\leq j\leq N}$ on the form

$$\forall x \in \Omega_k : \mathbf{u}(x,t) \approx \mathbf{u}_N(x,t) = \sum_{j=1}^{N} \hat{\mathbf{u}}_j(t)\psi_j(x). \tag{4.1.5}$$

The functions ψ_j can be Lagrange polynomials, Legendre polynomials or Chebychev polynomials. In the expression (4.1.6) we replace $\mathbf{u}$ by $\mathbf{u}_N$ with the polynomial expansion (4.1.5)

$$\int_{\Omega_k} \mathbf{Q}\frac{\partial \mathbf{u}_N}{\partial t}\psi_i(x)\ \mathrm{d}x - \int_{\Omega_k} \mathbf{F}(\mathbf{u}_N)\frac{\partial \psi_i(x)}{\partial x}\ \mathrm{d}x = -\int_{\partial\Omega_k} \mathbf{F}^*(\mathbf{u}_N)\psi_i(x)\ \mathrm{d}x, \tag{4.1.6}$$

In order to complete the numerical scheme, we need to define the numerical flux $\mathbf{F}^*$. It is a function of two states, one on the left side and one on the right, following the normal direction at the interface. For that, we choose either an average numerical flux or an upwind numerical flux. For a centered numerical flux we have ([6], [14], [15]):

$$\mathbf{F}^*(\mathbf{u}^-,\mathbf{u}^+) = \begin{cases} \frac{1}{2}(H^+ + H^-) \\ \\ \frac{1}{2}(E^+ + E^-) \end{cases}$$

and for an upwind numerical flux the following

$$\mathbf{F}^*(\mathbf{u}^-,\mathbf{u}^+) = \begin{cases} \frac{1}{Z^+ + Z^-}(Z^+H^+ + Z^-H^- - [E^+ - E^-]) \\ \\ \frac{1}{Y^+ + Y^-}(Y^+E^+ + Y^-E^- - [H^+ - H^-]) \end{cases}$$

where $Z^\pm = \sqrt{\frac{\mu^\pm}{\epsilon^\pm}}$ and $Y^\pm = \sqrt{\frac{\epsilon^\pm}{\mu^\pm}}$ respectively the impedance and the conductance in the local element(Z^-,Y^-) and in the neighbouring element(Z^+,Y^+). In the case $\mu = \epsilon = 1$, $Z^\pm$ and $Y^\pm$ are equal to one, and the upwind numerical flux simplifies to

$$\mathbf{F}^*(\mathbf{u}^-,\mathbf{u}^+) = \begin{cases} \frac{1}{2}(H^+ + H^- - [E^+ - E^-]) \\ \\ \frac{1}{2}(E^+ + E^- - [H^+ - H^-]). \end{cases}$$

For our numerical test problem in one dimension, the domain Ω comprises two parts Ω_1 and Ω_2 with index of refraction n_1 and n_2 respectively. The exact values of the electric field and the magnetic field in the domain Ω are ([6],[13]):

$$E_{\mathbf{e}}(x,t) = -\left[A_k e^{in_k\omega x} - B_k e^{in_k\omega x}\right] e^{i\omega t},$$
$$H_{\mathbf{e}}(x,t) = n_k\left[A_k e^{in_k\omega x} + B_k e^{in_k\omega x}\right] e^{i\omega t};$$

where $k = 1, 2$ indicates the values in medium 1 and 2,

$$A_1 = \frac{n_2\cos(n_2\omega)}{n_1\cos(n_1\omega)}, \qquad A_2 = e^{-i\omega(n_1+n_2)},$$

and

$$B_1 = A_1 e^{-2in_1\omega}, \quad B_2 = A_2 e^{-2in_2\omega}.$$

ω is the wavenumber and if $n_1 = n_2 = n$, then $\omega = \frac{2\pi}{n}$.
$E_{\mathbf{e}}(x,t)$ and $H_{\mathbf{e}}(x,t)$ are complex fields, thus to obtain the real fields corresponding to them, we take the real part.

4.1.2 Results of numerical experiments

In this section, we provide the computational results that we obtained with two different numerical fluxes namely the upwind numerical flux and the average numerical flux. To evaluate the accuracy of the numerical scheme, we compute the difference between the exact solution let say u and the approximate solution u_h using the N_2 and N_∞ norms:

$$N_2 = \left(\int_\Omega (u-u_h)^2 \mathrm{d}x\right)^{\frac{1}{2}}, \qquad N_\infty = \max_{x\in\Omega}|u-u_h|.$$

We must point out that since the numerical solution is computed at discrete points, the difference is evaluated at each point. Hence in the discrete case, the error is calculated as follows:

$$N_2 = \left(\sum_{i=1}^{N}|u_i - u_h^i|^2\right)^{\frac{1}{2}}, \qquad N_\infty = \max_{1\le i\le N}|u_i - u_h^i|.$$

The code is run on a DELL desktop with these features: OS type 64-bit, Processor Intel Core 2 Duo CPU E8400 @ 3.00 GHz×2.

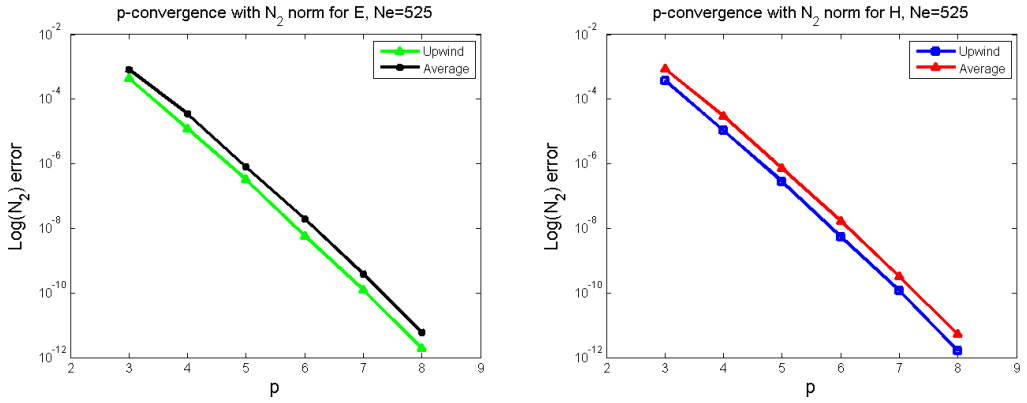
4.1.3 p-convergence: $n_1 = 1$ and $n_2 = 1$

Norm	p	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	3	0.000433069	38.2775	0.000823945	37.8258
	4	$1.21577\ 10^{-5}$	39.7246	$3.49143\ 10^{-5}$	38.6353
	5	$3.18643\ 10^{-7}$	40.2076	$8.02441\ 10^{-7}$	40.0989
	6	$5.79124\ 10^{-9}$	41.596	$1.92526\ 10^{-8}$	42.1547
	7	$1.24897\ 10^{-10}$	44.2145	$3.84286\ 10^{-10}$	43.383
	8	$1.88212\ 10^{-12}$	45.2042	$5.95531\ 10^{-12}$	44.9771
N_∞	3	0.00296447	38.2775	0.0078574	37.8258
	4	0.000113196	39.7246	0.000324575	38.6353
	5	$4.648\ 10^{-6}$	40.2076	$1.68261\ 10^{-5}$	40.0989
	6	$1.18026\ 10^{-7}$	41.596	$3.25984\ 10^{-7}$	42.1547
	7	$2.91196\ 10^{-9}$	44.2145	$1.58002\ 10^{-8}$	43.383
	8	$6.91419\ 10^{-11}$	45.2042	$1.99814\ 10^{-10}$	44.9771

Table 4.1: p-convergence by fixing Ne=525 for the electric field E

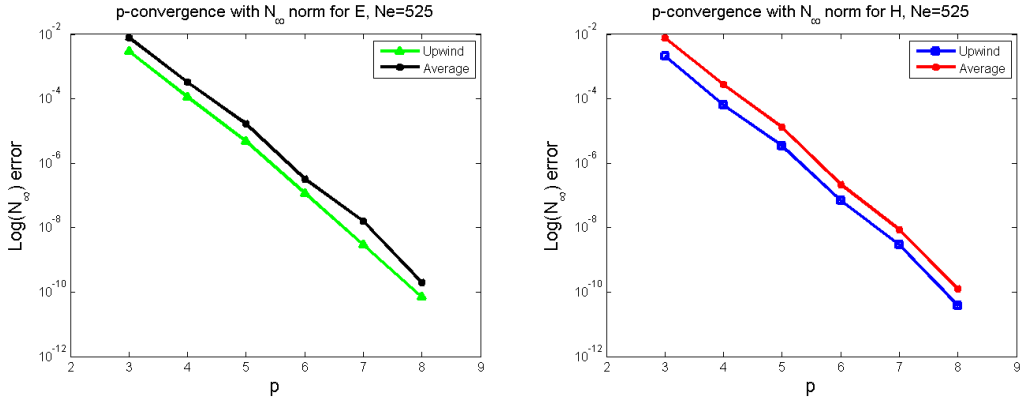
Norm	p	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	3	0.000376175	38.2775	0.000841573	37.8258
	4	$1.06343\ 10^{-5}$	39.7246	$2.98572\ 10^{-5}$	38.6353
	5	$2.77381\ 10^{-7}$	40.2076	$7.20149\ 10^{-7}$	40.0989
	6	$5.47164\ 10^{-9}$	41.596	$1.68707\ 10^{-8}$	42.1547
	7	$1.1688\ 10^{-10}$	44.2145	$3.18156\ 10^{-10}$	43.383
	8	$1.636\ 10^{-12}$	45.2042	$5.21445\ 10^{-12}$	44.9771
N_∞	3	0.00211323	38.2775	0.00761715	37.8258
	4	$6.34747\ 10^{-5}$	39.7246	0.000275651	38.6353
	5	$3.46016\ 10^{-6}$	40.2076	$1.29447\ 10^{-5}$	40.0989
	6	$6.91346\ 10^{-8}$	41.596	$2.20323\ 10^{-7}$	42.1547
	7	$2.90891\ 10^{-9}$	44.2145	$8.59575\ 10^{-9}$	43.383
	8	$3.71951\ 10^{-11}$	45.2042	$1.26401\ 10^{-10}$	44.9771

Table 4.2: p-convergence by fixing Ne=525 for the magnetic field H



(a) (b)

Figure 4.1: p-convergence with N_2 norm in Log scale for E and H with upwind and centered numerical flux, number of elements Ne=525.



(a) (b)

Figure 4.2: p-convergence with N_∞ norm in Log scale for E and H with upwind and centered numerical flux, number of elements Ne=525.

4.1.4 h-convergence relative to N_2 and N_∞ norms

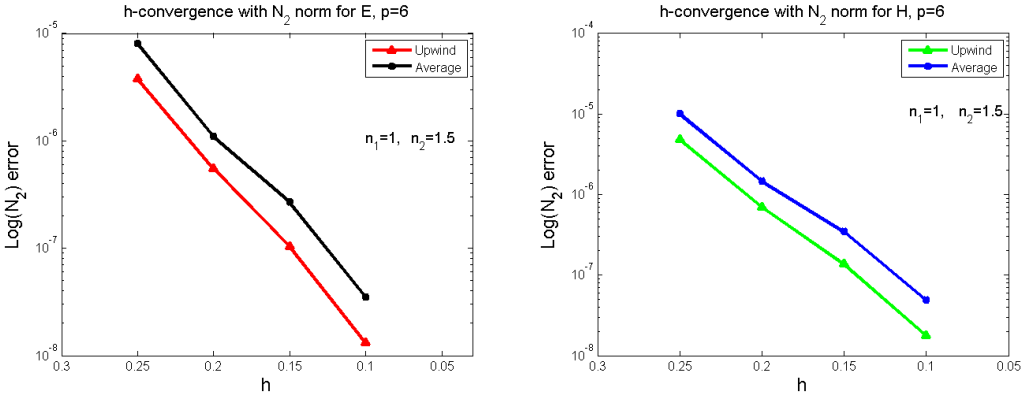
$n_1 = 1$, $n_2 = 1.5$

Norm	h	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	0.1	$1.31383\ 10^{-8}$	82.8899	$3.50731\ 10^{-8}$	82.7453
	0.15	$1.04369\ 10^{-7}$	41.648	$2.69399\ 10^{-7}$	41.382
	0.2	$5.49315\ 10^{-7}$	20.8174	$1.10197\ 10^{-6}$	20.6613
	0.25	$3.77748\ 10^{-6}$	13.4267	$8.08069\ 10^{-6}$	13.3643
N_∞	0.1	$3.15675\ 10^{-7}$	82.8899	$6.18806\ 10^{-7}$	82.7453
	0.15	$2.41496\ 10^{-6}$	41.648	$6.88607\ 10^{-6}$	41.382
	0.2	$1.21861\ 10^{-5}$	20.8174	$3.66393\ 10^{-5}$	20.6613
	0.25	$7.89714\ 10^{-5}$	13.4267	0.000134903	13.3643

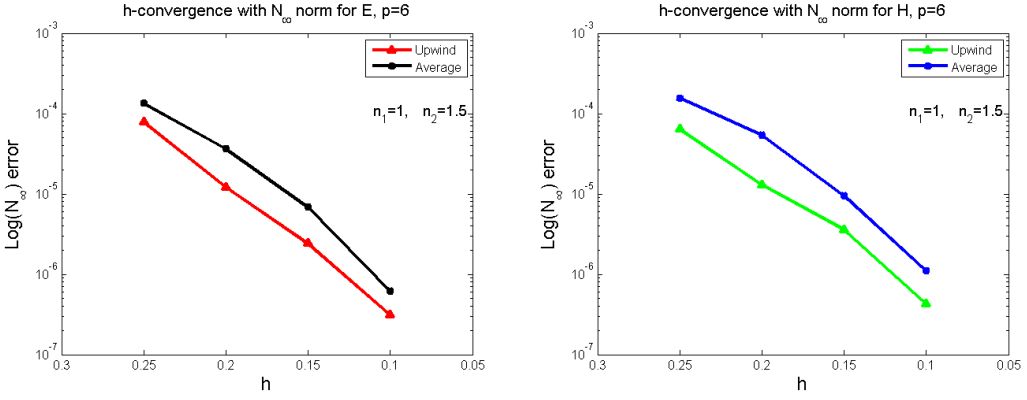
Table 4.3: h-convergence by fixing $p = 6$ for the electric field E

Norm	h	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	0.1	$1.78319\ 10^{-8}$	82.8899	$4.90363\ 10^{-8}$	82.7453
	0.15	$1.37517\ 10^{-7}$	41.648	$3.48482\ 10^{-7}$	41.382
	0.2	$7.03542\ 10^{-7}$	20.8174	$1.47042\ 10^{-6}$	20.6613
	0.25	$4.88153\ 10^{-6}$	13.4267	$1.0077\ 10^{-5}$	13.3643
N_∞	0.1	$4.30746\ 10^{-7}$	82.8899	$1.10784\ 10^{-6}$	82.7453
	0.15	$3.60965\ 10^{-6}$	41.648	$9.54245\ 10^{-6}$	41.382
	0.2	$1.30892\ 10^{-5}$	20.8174	$5.4088\ 10^{-5}$	20.6613
	0.25	$6.37361\ 10^{-5}$	13.4267	0.000156304	13.3643

Table 4.4: h-convergence by fixing $p = 6$ for the magnetic field H



(a) (b)



(a) (b)

Figure 4.4: h-convergence with N_2(top) and N_∞(bottom) norms with upwind and centered numerical flux, polynomial degree $p = 6$.

$\underline{n_1 = n_2 = 1}$

Norm	h	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	0.1	$4.65408\ 10^{-8}$	80.354	$1.14131\ 10^{-7}$	80.2642
	0.15	$3.18643\ 10^{-7}$	40.413	$8.02441\ 10^{-7}$	40.9204
	0.2	$1.18393\ 10^{-6}$	20.6893	$2.62405\ 10^{-6}$	20.5247
	0.25	$5.34008\ 10^{-6}$	13.0274	$1.12371\ 10^{-5}$	12.9841
N_∞	0.1	$6.79996\ 10^{-7}$	80.354	$1.52746\ 10^{-6}$	80.2642
	0.15	$4.648\ 10^{-6}$	40.413	$1.68261\ 10^{-5}$	40.9204
	0.2	$2.27826\ 10^{-5}$	20.6893	$3.6326\ 10^{-5}$	20.5247
	0.25	$8.78056\ 10^{-5}$	13.0274	0.000189606	12.9841

Table 4.5: h-convergence by fixing $p = 5$ for the electric field E

Norm	h	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	0.1	$4.22204\ 10^{-8}$	80.354	$8.92256\ 10^{-6}$	80.2642
	0.15	$2.77381\ 10^{-7}$	40.413	0.000111572	40.9204
	0.2	$1.06546\ 10^{-6}$	20.6893	0.000313137	20.5247
	0.25	$4.79445\ 10^{-6}$	13.0274	0.00196619	12.9841
N_∞	0.1	$4.61442\ 10^{-7}$	80.354	$1.56813\ 10^{-6}$	80.2642
	0.2	$3.46016\ 10^{-6}$	40.413	$1.29447\ 10^{-5}$	40.9204
	0.3	$1.10143\ 10^{-5}$	20.6893	$3.89437\ 10^{-5}$	20.5247
	0.4	$3.6193\ 10^{-5}$	13.0274	0.000114805	12.9841

Table 4.6: h-convergence by fixing $p = 5$ for the magnetic field H

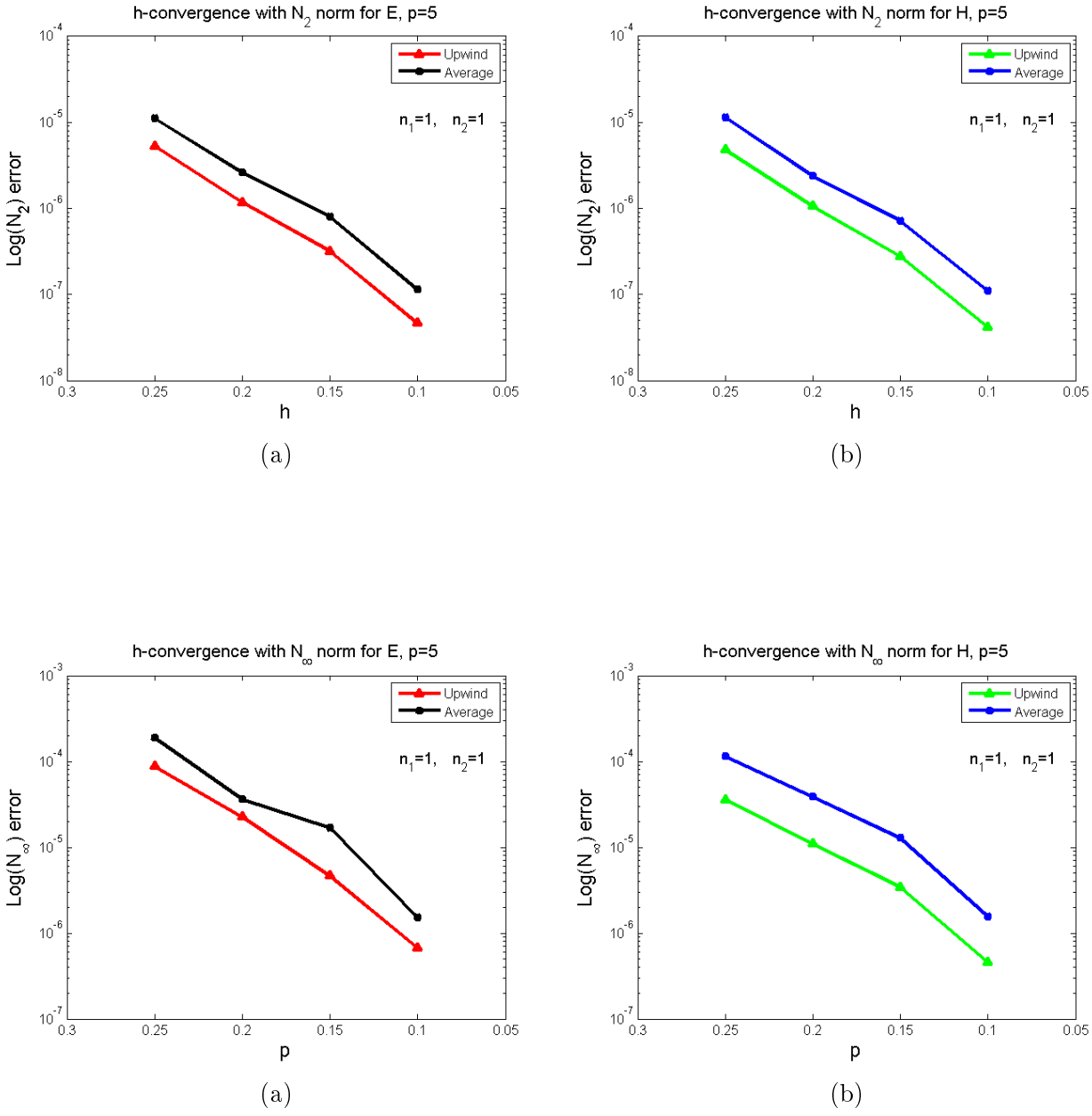


Figure 4.6: h-convergence with N_2(top) and N_∞(bottom) norms with upwind and centered numerical flux, polynomial degree $p = 5$.

4.2 The two-dimensional scheme

In chapter 2, we learned that Maxwell's equations can be reduced to two dimensions in two different modes, the transverse electric mode and the transverse magnetic mode. Each of them consists of a system of three partial differential equations. The two systems can be written in conservative form like the equation (3.3.1) in chapter(3). In this section, we carry out the variational formulation of the two-dimensional problems as we did in chapter 3. We start with the **TM** polarization.

4.2.1 Test problem in TM polarization

For this test problem, we consider the transverse magnetic polarisation that we defined in section (2.3.1) which includes H_x,H_y and E_z.

$$\begin{aligned} \mu\frac{\partial H_x}{\partial t} &= -\frac{\partial E_z}{\partial y}, \\ \mu\frac{\partial H_y}{\partial t} &= \frac{\partial E_z}{\partial x}, \\ \epsilon\frac{\partial E_z}{\partial t} &= \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y}. \end{aligned} \tag{4.2.1}$$

Like the problem in one dimension, we define $\mathbf{Q}$ the matrix of materials, $\mathbf{u}$ the unknown and $\mathbf{F}(\mathbf{u})$ the flux:

$$\mathbf{Q} = \begin{pmatrix} \mu & 0 & 0 \\ 0 & \mu & 0 \\ 0 & 0 & \epsilon \end{pmatrix}, \mathbf{u} = \begin{pmatrix} H_x \\ H_y \\ E_z \end{pmatrix} \text{ and } \mathbf{F}(\mathbf{u}) = \begin{pmatrix} 0 & E_z \\ -E_z & 0 \\ -H_y & H_x \end{pmatrix}.$$

We rewrite the system (4.2.1) in the form (3.3.1) in chapter (3)

$$\mathbf{Q}\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{u}) = 0. \tag{4.2.2}$$

We multiply (4.2.2) by a test function ϕ_i and integrate over the domain Ω. After a double integration by parts and approximating the flux by a numerical flux $\mathbf{F}^*$ on $\partial\Omega_k$, we end up with the variational formulation:

$$\int_{\Omega_k} \mathbf{Q}\frac{\partial \mathbf{u}}{\partial t}\phi_i(r)\ \mathrm{d}r + \int_{\Omega_k} (\nabla \cdot \mathbf{F}(\mathbf{u}))\,\phi_i(r)\ \mathrm{d}r = \int_{\partial\Omega_k} \mathbf{n} \cdot (\mathbf{F} - \mathbf{F}^*)\,\phi_i(r)\ \mathrm{d}r. \tag{4.2.3}$$

Expanding the unknown function $\mathbf{u}$ into a polynomial basis $(\phi_i)_{i=1}^N$ ($\mathbf{u}(r,t) \approx \mathbf{u}_N(r,t) = \sum_{j=1}^{N} \hat{\mathbf{u}}_j(t)\phi_j(r)$) and replacing $\mathbf{u}$ by $\mathbf{u}_N$ in (4.2.3) yield a scheme

similar to the one we obtained in the one-dimensional problem although there is a slight difference:

$$\mathbf{QM}\frac{\partial \hat{\mathbf{u}}}{\partial t} + \mathbf{S}\cdot\mathbf{F} = \hat{\mathbf{F}}\mathbf{n}\cdot(\mathbf{F}-\mathbf{F}^*)\,, \tag{4.2.4}$$

where $\hat{\mathbf{u}}$ is the vector of coefficients $\hat{\mathbf{u}}_j (1 \leq j \leq N)$, $\mathbf{M}$, $\mathbf{S}$, $\hat{\mathbf{F}}$ are the mass matrix, the stiffness matrix and the face matrix respectively and are defined as follows:

$$\begin{aligned}
\mathbf{M}_{ij} &= \int_{\Omega_k} \phi_i(r)\phi_j(r)\mathrm{d}r,\\
\mathbf{S}_{ij} &= \int_{\Omega_k} \phi_i(r)\nabla\phi_j(r)\mathrm{d}r,\\
\hat{\mathbf{F}}_{ij} &= \int_{\partial\Omega_k} \phi_i(r)\phi_j(r)\mathrm{d}r.
\end{aligned}$$

Finally, we define the numerical flux to complete the scheme([6], [15], [28]):

$$\mathbf{n}\cdot(\mathbf{F}-\mathbf{F}^*) = \begin{cases}
\frac{1}{2}\Big(-(E_z^+ + E_z^-) + \alpha[(H_y^+ - H_y^-) - (H_x^+ - H_x^-)]\Big)\\
\frac{1}{2}\Big((E_z^+ + E_z^-) + \alpha[(H_y^+ - H_y^-) - (H_x^+ - H_x^-)]\Big)\\
\frac{1}{2}\Big((H_y^+ + H_y^-) - (H_x^+ + H_x^-) - \alpha[E_z^+ - E_z^-]\Big)
\end{cases}$$

where α is a constant lying in the interval $[0,1]$. The two relevant cases that we are dealing with in this problem are $\alpha = 0$ and $\alpha = 1$. When $\alpha = 0$, one has a centered numerical flux and for $\alpha = 1$ an upwind numerical flux. To test the numerical scheme, the following exact values of H_x, H_y and E_z will be implemented in the code:

$$\begin{aligned}
H_x(x,y,t) &= -\frac{\pi n}{\omega}\sin(m\pi x)\cos(n\pi y)\sin(\omega t),\\
H_y(x,y,t) &= \frac{\pi m}{\omega}\cos(m\pi x)\sin(n\pi y)\sin(\omega t),\\
E_z(x,y,t) &= \sin(m\pi x)\sin(n\pi y)\cos(\omega t),
\end{aligned}$$

where $\omega = \pi\sqrt{m^2+n^2}$ is the resonance frequency and the system is subject to the following initial conditions

$$\begin{aligned}
H_x(x,y,0) &= 0,\\
H_y(x,y,0) &= 0,\\
E_z(x,y,0) &= \sin(m\pi x)\sin(n\pi y).
\end{aligned}$$

For the time marching, we use the fourth order Runge-Kutta method with a time step $\Delta t = 0.001$, number of steps NSteps=1000 and final time $T = 1$. To show the p-convergence of the method, the error is computed for several values of p while fixing the number of elements Ne=267. As for the h-convergence, the mesh is refined substantially, that is we increase the number of elements Ne. Thus, we compute the error for Ne=171, Ne=267, Ne=525 and Ne=1033 while firstly setting $p = 5$ and secondly $p = 6$.

4.2.2 p-convergence in the N_2 and N_∞ norms for the TM test problem

Table 4.7: Error computed with Ne=267 for H_x

Norm	p	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	3	0.00178033	199.256	0.0459058	199.862
	4	$9.96812\ 10^{-5}$	208.958	0.00299052	203.633
	5	$4.63224\ 10^{-6}$	216.313	0.000261836	213.242
	6	$1.78846\ 10^{-7}$	225.334	$9.30758\ 10^{-6}$	222.29
	7	$7.00582\ 10^{-9}$	238.682	$6.80478\ 10^{-7}$	234.134
	8	$1.99057\ 10^{-10}$	247.232	$1.51698\ 10^{-8}$	245.101
	9	$7.56529\ 10^{-12}$	264.043	$1.02506\ 10^{-9}$	267.027
	10	$2.78327\ 10^{-12}$	279.043	$9.10541\ 10^{-11}$	279.759
N_∞	3	0.00854869	199.256	0.33174	199.862
	4	0.000820097	208.958	0.0263105	203.633
	5	$6.93925\ 10^{-5}$	216.313	0.00473269	213.242
	6	$3.10325\ 10^{-6}$	225.334	0.000165144	222.29
	7	$2.2525\ 10^{-7}$	238.682	$2.70764\ 10^{-5}$	234.134
	8	$5.53935\ 10^{-9}$	247.232	$4.82133\ 10^{-7}$	245.101
	9	$3.69731\ 10^{-10}$	264.043	$7.20305\ 10^{-8}$	267.027
	10	$6.35093\ 10^{-12}$	279.043	$9.10541\ 10^{-10}$	279.759

Table 4.8: Error computed with Ne=267 for H_y

Norm	p	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	3	0.0018859	199.256	0.047271	199.862
	4	$9.87827\ 10^{-5}$	208.958	0.00308158	203.633
	5	$4.92867\ 10^{-6}$	216.313	0.000272208	213.242
	6	$1.78989\ 10^{-7}$	225.334	$9.60479\ 10^{-6}$	222.29
	7	$7.54635\ 10^{-9}$	238.682	$7.15433\ 10^{-7}$	234.134
	8	$2.02045\ 10^{-10}$	247.232	$1.57862\ 10^{-8}$	245.101
	9	$8.19374\ 10^{-12}$	264.043	$1.08484\ 10^{-9}$	267.027
	10	$2.78415\ 10^{-12}$	279.09	$1.66572\ 10^{-11}$	279.759
N_∞	3	0.0133793	199.256	0.398398	199.862
	4	0.00115719	208.958	0.0275281	203.633
	5	$9.80127\ 10^{-5}$	216.313	0.0054503	213.242
	6	$4.26263\ 10^{-6}$	225.334	0.000199659	222.29
	7	$2.96644\ 10^{-7}$	238.682	$2.70595\ 10^{-5}$	234.134
	8	$7.52092\ 10^{-9}$	247.232	$6.59275\ 10^{-7}$	245.101
	9	$4.76451\ 10^{-10}$	264.043	$6.7245\ 10^{-8}$	267.027
	10	$7.23024\ 10^{-12}$	279.09	$1.15485\ 10^{-9}$	279.759

Table 4.9: Error computed with Ne=267 for E_z

Norm	p	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	3	0.00186388	199.256	0.00347365	199.862
	4	$9.57025\ 10^{-5}$	208.958	0.000174732	203.633
	5	$4.82305\ 10^{-6}$	216.313	$1.14649\ 10^{-5}$	213.242
	6	$1.73451\ 10^{-7}$	225.334	$3.39719\ 10^{-7}$	222.29
	7	$7.40256\ 10^{-9}$	238.682	$2.04451\ 10^{-8}$	234.134
	8	$1.96167\ 10^{-10}$	247.232	$4.12978\ 10^{-10}$	245.101
	9	$1.61955\ 10^{-11}$	264.043	$2.69182\ 10^{-11}$	267.027
	10	$1.40159\ 10^{-11}$	279.09	$1.4019\ 10^{-11}$	279.759
N_∞	3	0.0103986	199.256	0.0146862	199.862
	4	0.000727188	208.958	0.00168659	203.633
	5	$6.6081\ 10^{-5}$	216.313	0.000124801	213.242
	6	$2.46883\ 10^{-6}$	225.334	$6.24637\ 10^{-6}$	222.29
	7	$1.7915\ 10^{-7}$	238.682	$5.02205\ 10^{-7}$	234.134
	8	$4.18117\ 10^{-9}$	247.232	$1.46446\ 10^{-8}$	245.101
	9	$2.67186\ 10^{-10}$	264.043	$1.05118\ 10^{-9}$	267.027
	10	$1.75535\ 10^{-11}$	279.09	$2.06606\ 10^{-11}$	279.759

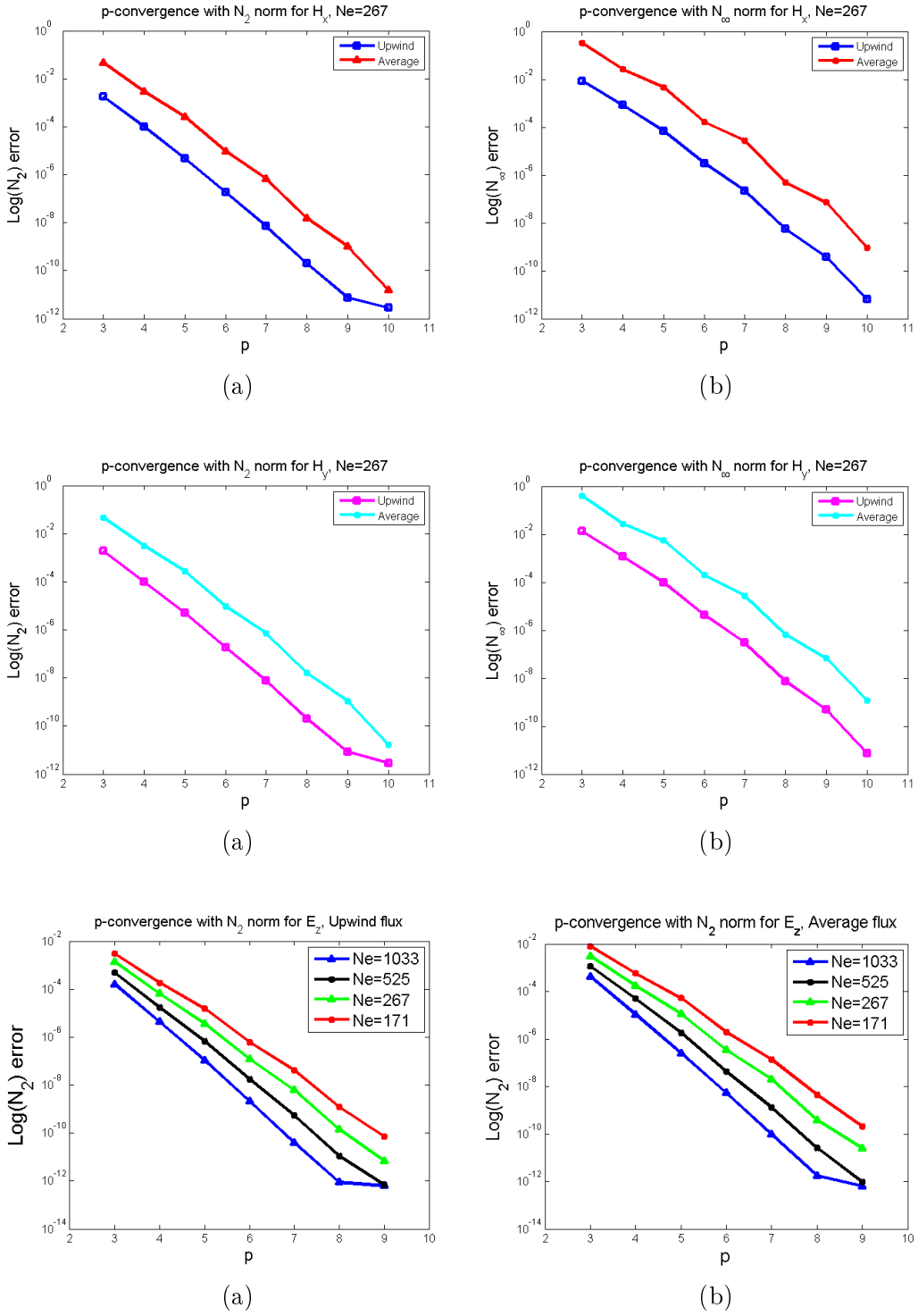


Figure 4.9: p-convergence with N_2 and N_∞ norms for H_x(top, Ne=267), H_y(middle, Ne=267) and E_z(bottom, Ne=1033,525,267,171) with upwind and centered numerical flux.

4.2.3 h-convergence fixing $p = 5$

Table 4.10: h-convergence by fixing $p = 5$ for H_x

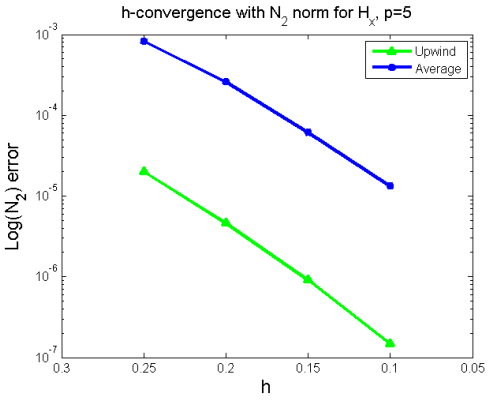
Norm	h	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	0.1	$1.4962\ 10^{-7}$	847.419	$1.33919\ 10^{-5}$	854.79
	0.15	$9.19025\ 10^{-7}$	423.052	$6.12766\ 10^{-5}$	423.751
	0.2	$4.63224\ 10^{-6}$	212.963	0.000261836	214.928
	0.25	$2.02728\ 10^{-5}$	136.58	0.000826628	135.267
N_∞	0.1	$2.25606\ 10^{-6}$	847.419	0.000255205	854.79
	0.15	$1.42849\ 10^{-5}$	423.052	0.00162483	423.751
	0.2	$6.93925\ 10^{-5}$	212.963	0.00473269	214.928
	0.25	0.000275211	136.58	0.0121255	135.267

Table 4.11: h-convergence by fixing $p = 5$ for H_y

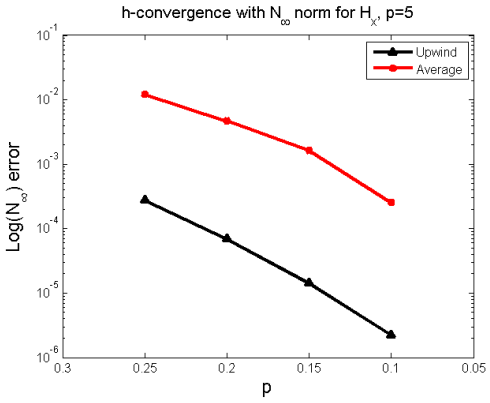
Norm	h	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	0.1	$1.54496\ 10^{-7}$	847.419	$1.33248\ 10^{-5}$	854.79
	0.15	$9.16501\ 10^{-7}$	423.052	$5.71486\ 10^{-5}$	423.751
	0.2	$4.92867\ 10^{-6}$	212.963	0.000272208	214.928
	0.25	$2.18573\ 10^{-5}$	136.58	0.000841322	135.267
N_∞	0.1	$3.02516\ 10^{-6}$	847.419	0.000285898	854.79
	0.15	$1.3911\ 10^{-5}$	423.052	0.00102574	423.751
	0.2	$9.80127\ 10^{-5}$	212.963	0.0054503	214.928
	0.25	0.000323558	136.58	0.0119075	135.267

Table 4.12: h-convergence by fixing $p = 5$ for E_z

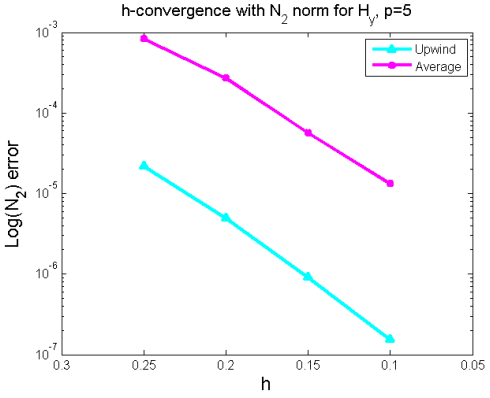
Norm	h	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	0.1	$1.5613\ 10^{-7}$	847.419	$3.04383\ 10^{-7}$	854.79
	0.15	$9.53162\ 10^{-7}$	423.052	$1.95057\ 10^{-6}$	423.751
	0.2	$4.82305\ 10^{-6}$	212.963	$1.14649\ 10^{-5}$	214.928
	0.25	$2.17718\ 10^{-5}$	136.58	$5.02423\ 10^{-5}$	135.267
N_∞	0.1	$1.90366\ 10^{-6}$	847.419	$4.58217\ 10^{-6}$	854.79
	0.15	$1.24004\ 10^{-5}$	423.052	$2.31349\ 10^{-5}$	423.751
	0.2	$6.6081\ 10^{-5}$	212.963	0.000124801	214.928
	0.25	0.000196871	136.58	0.000435994	135.267



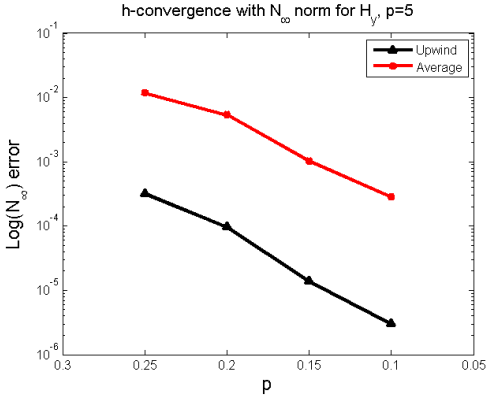
(a)



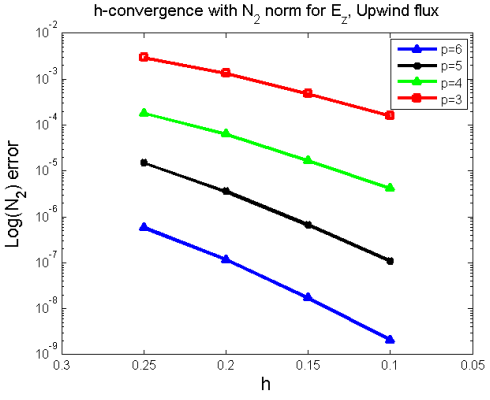
(b)



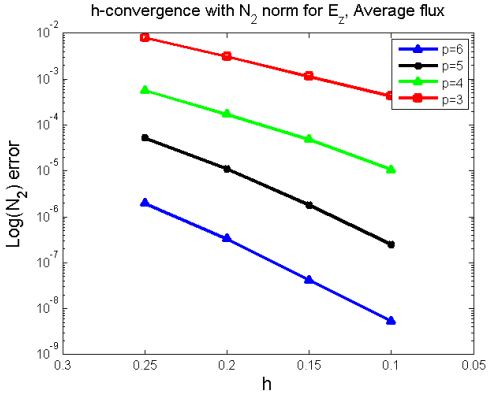
(a)



(b)



(a)



(b)

Figure 4.12: h-convergence with N_2 and N_∞ norms for H_x(top), H_y(middle) and E_z(bottom, $p = 3, 4, 5, 6$) with upwind and centered numerical flux.

4.2.4 h-convergence fixing $p = 6$

Table 4.13: h-convergence by fixing $p = 6$ for H_x

Norm	h	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	0.1	$3.01866\ 10^{-9}$	881.584	$3.37817\ 10^{-7}$	877.512
	0.15	$2.53931\ 10^{-8}$	439.447	$1.8815\ 10^{-6}$	440.447
	0.2	$1.78846\ 10^{-7}$	220.543	$9.30758\ 10^{-6}$	221.754
	0.25	$1.04853\ 10^{-6}$	141.695	$3.49961\ 10^{-5}$	140.97
N_∞	0.1	$6.57264\ 10^{-8}$	881.584	$1.56314\ 10^{-5}$	877.512
	0.15	$9.56357\ 10^{-7}$	439.447	$4.52458\ 10^{-5}$	440.447
	0.2	$3.10325\ 10^{-6}$	220.543	0.000165144	221.754
	0.25	$2.15748\ 10^{-5}$	141.695	0.00076807	140.97

Table 4.14: h-convergence by fixing $p = 6$ for H_y

Norm	h	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	0.1	$3.10169\ 10^{-9}$	881.584	$3.36292\ 10^{-7}$	877.512
	0.15	$2.51632\ 10^{-8}$	439.447	$1.90776\ 10^{-6}$	440.447
	0.2	$1.78989\ 10^{-7}$	220.543	$9.60479\ 10^{-6}$	221.754
	0.25	$1.07039\ 10^{-6}$	141.695	$3.29651\ 10^{-5}$	140.97
N_∞	0.1	$6.36626\ 10^{-8}$	881.584	$1.57281\ 10^{-5}$	877.512
	0.15	$7.94177\ 10^{-7}$	439.447	$5.3721\ 10^{-5}$	440.447
	0.2	$4.26263\ 10^{-6}$	220.543	0.000199659	221.754
	0.25	$2.66702\ 10^{-5}$	141.695	0.000795635	140.97

Table 4.15: h-convergence by fixing $p = 6$ for E_z

Norm	h	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	0.1	$3.13852\ 10^{-9}$	881.584	$6.28828\ 10^{-9}$	877.512
	0.15	$2.52203\ 10^{-8}$	439.447	$5.24594\ 10^{-8}$	440.447
	0.2	$1.73451\ 10^{-7}$	220.543	$3.39719\ 10^{-7}$	221.754
	0.25	$9.34455\ 10^{-7}$	141.695	$2.08063\ 10^{-6}$	140.97
N_∞	0.1	$5.33329\ 10^{-8}$	881.584	$1.3825\ 10^{-7}$	877.512
	0.15	$5.24593\ 10^{-7}$	439.447	$1.0968\ 10^{-6}$	440.447
	0.2	$2.46883\ 10^{-6}$	220.543	$6.24637\ 10^{-6}$	221.754
	0.25	$1.49239\ 10^{-5}$	141.695	$3.58266\ 10^{-5}$	140.97

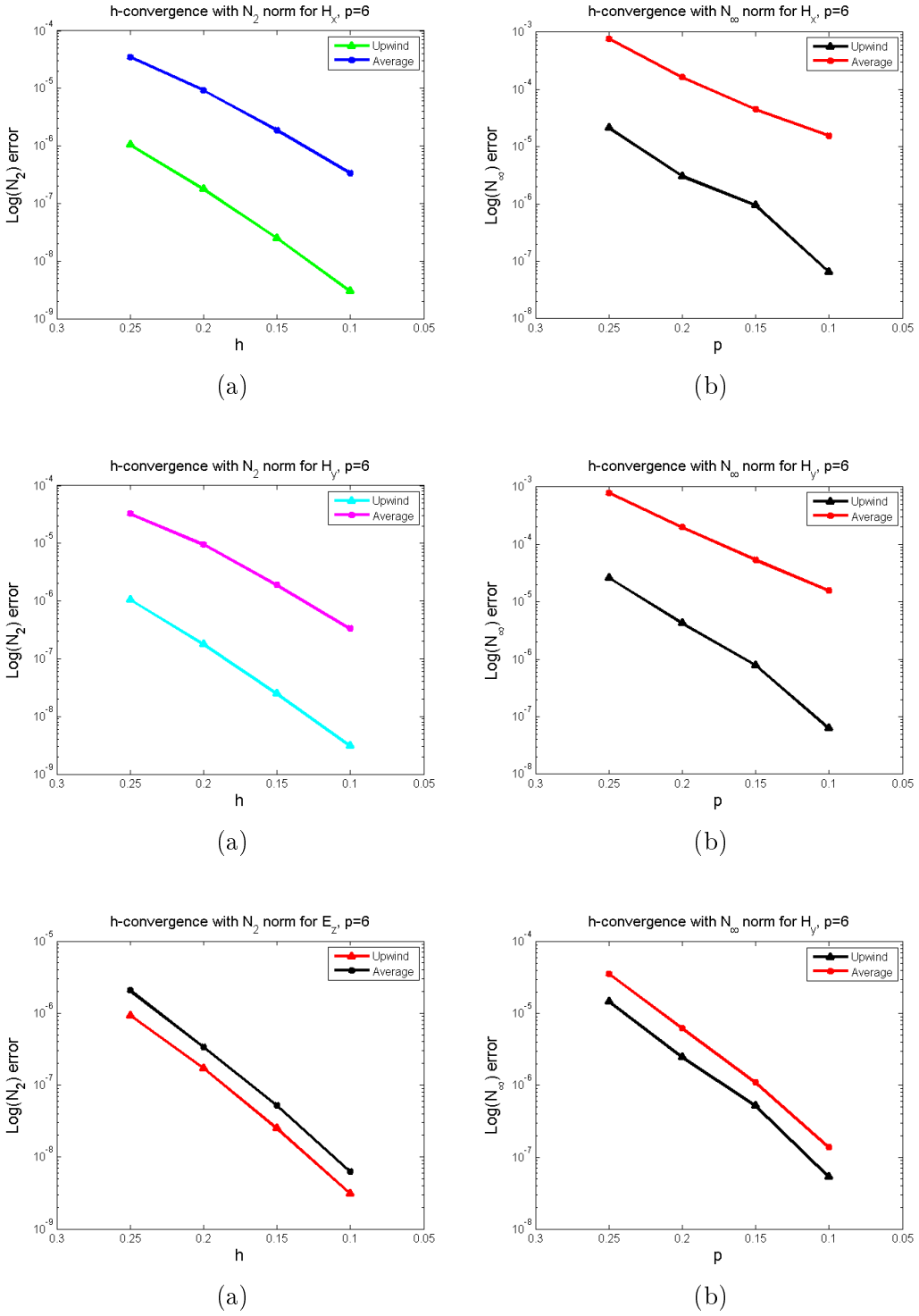


Figure 4.15: h-convergence with N_2 and N_∞ norms for E_x(top), E_y(middle) and H_z(bottom) with upwind and centered numerical flux, polynomial degree $p = 6$.

4.2.5 Test problem in TE polarization

We discuss the two dimensional Maxwell's equations, this time in tranverse electric mode. We recall the **TE** mode defined in section (2.3.2) involving E_x, E_y and H_z:

$$\begin{aligned}\epsilon\frac{\partial E_x}{\partial t} &= \frac{\partial H_z}{\partial y},\\ \epsilon\frac{\partial E_y}{\partial t} &= -\frac{\partial H_z}{\partial x},\\ \mu\frac{\partial H_z}{\partial t} &= \frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x}.\end{aligned}$$

We adopt the same notation as for the **TM** mode, the difference is that we have another set of electromagnetic components:

$$\mathbf{Q} = \begin{pmatrix}\epsilon & 0 & 0\\ 0 & \epsilon & 0\\ 0 & 0 & \mu\end{pmatrix},\ \mathbf{u} = \begin{pmatrix}E_x\\ E_y\\ H_z\end{pmatrix} \text{ and } \mathbf{F}(\mathbf{u}) = \begin{pmatrix}0 & -H_z\\ H_z & 0\\ E_y & -E_x\end{pmatrix}.$$

Proceeding as in the previous subsection, we obtain a numerical scheme similar to (4.2.4) subject to the initial conditions:

$$\begin{aligned}E_x(x,y,0) &= 0,\\ E_y(x,y,0) &= 0,\\ H_z(x,y,0) &= \cos(m\pi x)\cos(n\pi y).\end{aligned}$$

To test the accuracy of the numerical scheme, the analytical solutions are the following($\epsilon = \mu = 1$):

$$\begin{aligned}E_x(x,y,t) &= -\frac{\pi n}{\omega}\cos(m\pi x)\sin(n\pi y)\sin(\omega t),\\ E_y(x,y,t) &= \frac{\pi m}{\omega}\sin(m\pi x)\cos(n\pi y)\sin(\omega t),\\ H_z(x,y,t) &= \cos(m\pi x)\cos(n\pi y)\cos(\omega t).\end{aligned}$$

For the time-stepping, the parameters of integration are $\Delta t = 0.001$, NSteps=100, final time $T = 0.1$. To show the h-convergence and the p-convergence, the resolution and the polynomial expansion remain the same as for the transverse magnetic test problem.

4.2.6 p-convergence in the N_2 and N_∞ norms for the TE test problem

Table 4.16: Error computed with $h = 0.2$ for E_y

Norm	p	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	3	0.00126849	19.4428	0.00452301	19.2474
	4	$6.20719\ 10^{-5}$	20.0808	0.000300294	20.0557
	5	$3.61268\ 10^{-6}$	21.2662	$2.54375\ 10^{-5}$	21.0651
	6	$1.16766\ 10^{-7}$	21.9851	$9.37274\ 10^{-7}$	21.9694
	7	$6.07139\ 10^{-9}$	23.406	$6.6276\ 10^{-8}$	23.5286
	8	$1.39525\ 10^{-10}$	24.5424	$1.58856\ 10^{-9}$	24.3463
	9	$6.70659\ 10^{-12}$	26.5109	$9.99943\ 10^{-11}$	26.3017
	10	$9.39697\ 10^{-13}$	28.2437	$1.94314\ 10^{-12}$	28.1119
N_∞	3	0.00561739	19.4428	0.0289159	19.2474
	4	0.00056801	20.0808	0.00320146	20.0557
	5	$2.69487\ 10^{-5}$	21.2662	0.000337524	21.0651
	6	$2.43102\ 10^{-6}$	21.9851	$1.92287\ 10^{-5}$	21.9694
	7	$7.14385\ 10^{-8}$	23.406	$1.8528\ 10^{-6}$	23.5286
	8	$5.01003\ 10^{-9}$	24.5424	$6.09668\ 10^{-8}$	24.3463
	9	$1.19827\ 10^{-10}$	26.5109	$2.143\ 10^{-9}$	26.3017
	10	$6.31919\ 10^{-12}$	28.2437	$1.09976\ 10^{-10}$	28.1119

Table 4.17: Error computed with $h = 0.2$ for H_z

Norm	p	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	3	0.00127407	19.4428	0.00306108	19.2474
	4	$6.35211\ 10^{-5}$	20.0808	0.000149731	20.0557
	5	$3.50935\ 10^{-6}$	21.2662	$1.05539\ 10^{-5}$	21.0651
	6	$1.15128\ 10^{-7}$	21.9851	$2.56161\ 10^{-7}$	21.9694
	7	$5.99071\ 10^{-9}$	23.406	$1.89087\ 10^{-8}$	23.5286
	8	$1.40473\ 10^{-10}$	24.5424	$2.93125\ 10^{-10}$	24.3463
	9	$6.67037\ 10^{-12}$	26.5109	$9.143\ 10^{-11}$	26.3017
	10	$6.31703\ 10^{-13}$	28.2437	$6.68423\ 10^{-13}$	28.1119
N_∞	3	0.003852	19.4428	0.0299545	19.2474
	4	0.000285316	20.0808	0.000994325	20.0557
	5	$2.41909\ 10^{-5}$	21.2662	0.000209266	21.0651
	6	$6.65472\ 10^{-7}$	21.9851	$2.92547\ 10^{-6}$	21.9694
	7	$4.78902\ 10^{-8}$	23.406	$5.64491\ 10^{-7}$	23.5286
	8	$1.29621\ 10^{-9}$	24.5424	$7.63604\ 10^{-9}$	24.3463
	9	$7.14157\ 10^{-11}$	26.5109	$9.5876\ 10^{-10}$	26.3017
	10	$1.8171\ 10^{-12}$	28.2437	$1.38861\ 10^{-11}$	28.1119

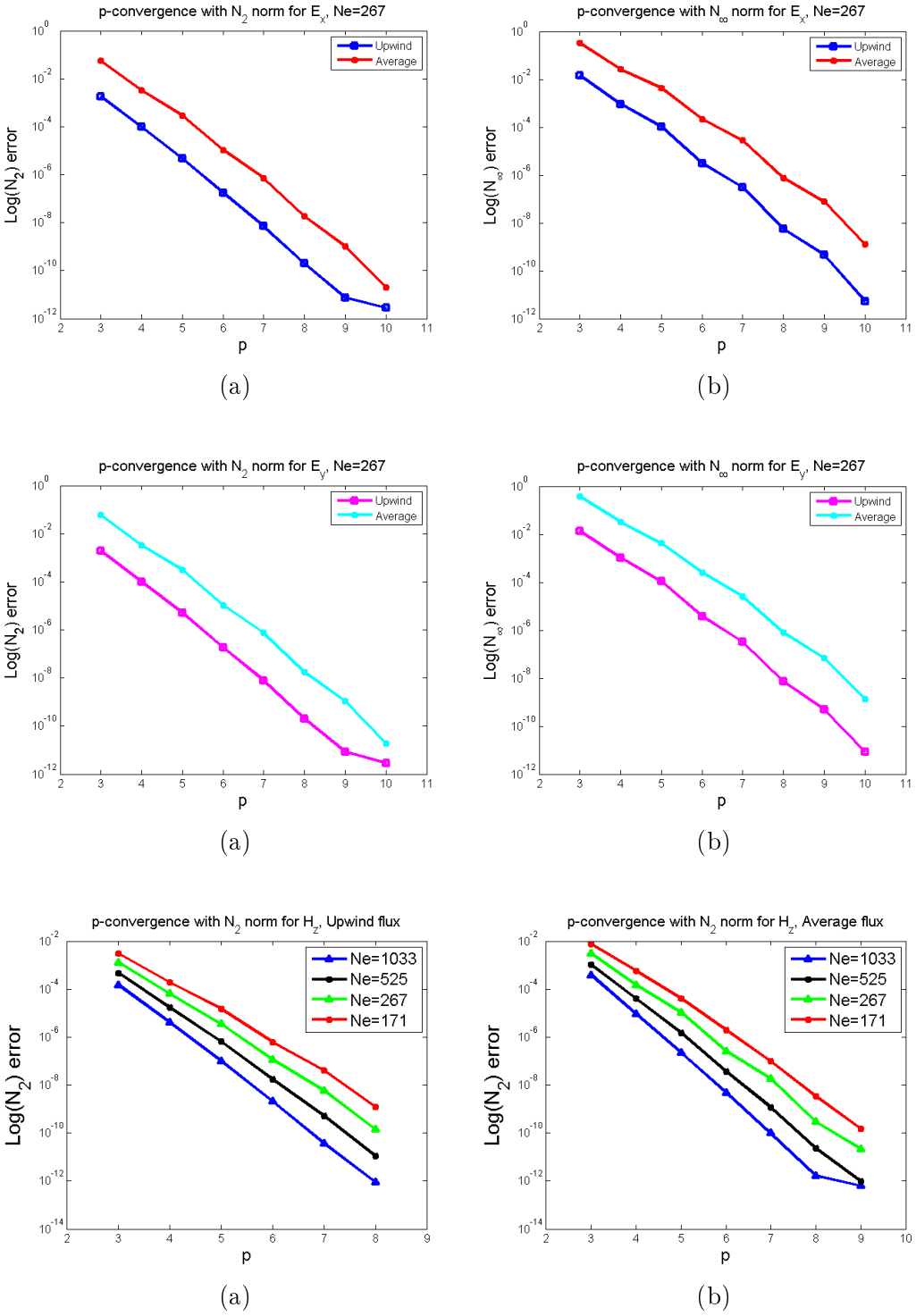


Figure 4.18: p-convergence with N_2 and N_∞ norms for E_x(top, Ne=267), E_y(middle, Ne=267) and H_z(bottom, Ne=171, Ne=267, Ne=525, Ne=1033) with upwind and centered numerical flux.

4.2.7 h-convergence in the N_2 and N_∞ norms fixing $p = 5$

Table 4.18: h-convergence by fixing $p = 5$ for E_x

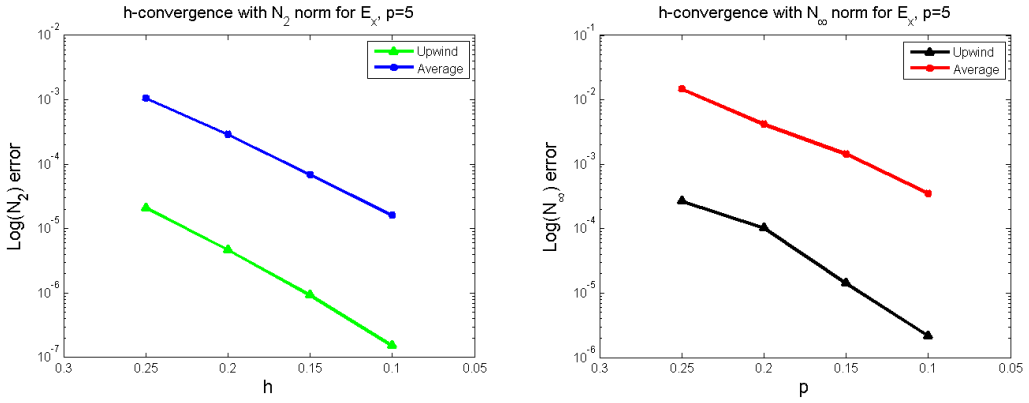
Norm	h	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	0.1	$1.53256\ 10^{-7}$	835.876	$1.61569\ 10^{-5}$	852.645
	0.15	$9.35109\ 10^{-7}$	424.997	$6.88687\ 10^{-5}$	433.417
	0.2	$4.71727\ 10^{-6}$	212.457	0.000290717	210.817
	0.25	$2.12594\ 10^{-5}$	134.391	0.014761	135.081
N_∞	0.1	$2.19661\ 10^{-6}$	835.876	0.00035279	852.645
	0.15	$1.43134\ 10^{-5}$	424.997	0.00144474	433.417
	0.2	0.000104023	212.457	0.00420882	210.817
	0.25	0.000271766	134.391	0.014761	135.081

Table 4.19: h-convergence by fixing $p = 5$ for E_y

Norm	h	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	0.1	$1.6025\ 10^{-7}$	835.876	$1.6472\ 10^{-5}$	852.645
	0.15	$9.50256\ 10^{-7}$	425.997	$7.03106\ 10^{-5}$	433.417
	0.2	$5.18499\ 10^{-6}$	212.457	0.000308706	210.817
	0.25	$2.17557\ 10^{-5}$	134.391	0.00099153	135.081
N_∞	0.1	$3.03414\ 10^{-6}$	835.876	0.000279942	852.645
	0.15	$1.24815\ 10^{-5}$	425.997	0.00106376	433.417
	0.2	0.000107649	212.457	0.00420882	210.817
	0.25	0.000309312	134.391	0.0108075	135.081

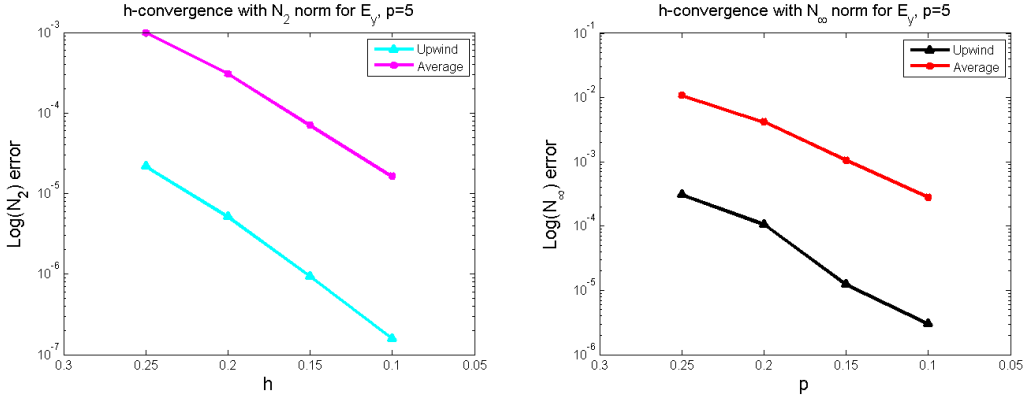
Table 4.20: h-convergence by fixing $p = 5$ for H_z

Norm	h	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	0.1	$1.5736\ 10^{-7}$	835.876	$3.04459\ 10^{-7}$	852.645
	0.15	$9.56684\ 10^{-7}$	424.997	$1.85036\ 10^{-6}$	433.417
	0.2	$4.83273\ 10^{-6}$	212.457	$1.02058\ 10^{-5}$	210.817
	0.25	$2.18347\ 10^{-5}$	134.391	$4.16963\ 10^{-5}$	135.081
N_∞	0.1	$1.84353\ 10^{-6}$	835.876	$4.4265\ 10^{-6}$	852.645
	0.15	$1.29824\ 10^{-5}$	424.997	$2.66094\ 10^{-5}$	433.417
	0.2	$6.41224\ 10^{-5}$	212.457	0.000119557	210.817
	0.25	0.000192599	134.391	0.000351508	135.081



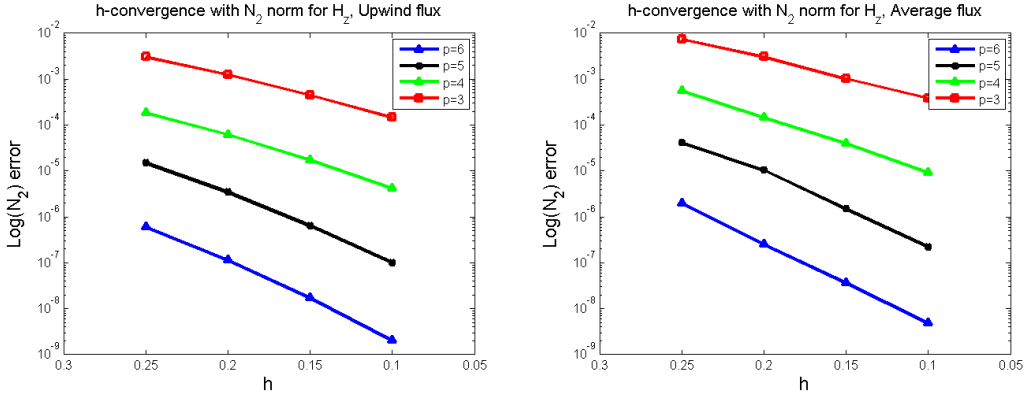
(a)

(b)



(a)

(b)



(a)

(b)

Figure 4.21: h-convergence with N_2 and N_∞ norms for E_x(top), E_y(middle) and H_z(bottom, $p = 3, 4, 5, 6$) with upwind and centered numerical flux.

4.2.8 h-convergence in the N_2 and N_∞ norms when $p = 6$

Norm	h	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	0.1	$3.04706\ 10^{-9}$	881.537	$3.8305\ 10^{-7}$	889.203
	0.15	$2.5315\ 10^{-8}$	449.47	$2.10083\ 10^{-6}$	452.433
	0.2	$1.78472\ 10^{-7}$	226.807	$1.06243\ 10^{-5}$	221.866
	0.25	$1.04276\ 10^{-6}$	143.911	$4.77486\ 10^{-5}$	146.395
N_∞	0.1	$7.53927\ 10^{-8}$	881.537	$1.74842\ 10^{-5}$	889.203
	0.15	$9.44688\ 10^{-7}$	449.47	$6.3454\ 10^{-5}$	452.433
	0.2	$3.15511\ 10^{-6}$	226.807	0.000216841	221.866
	0.25	$2.15257\ 10^{-5}$	143.911	0.00153756	146.395

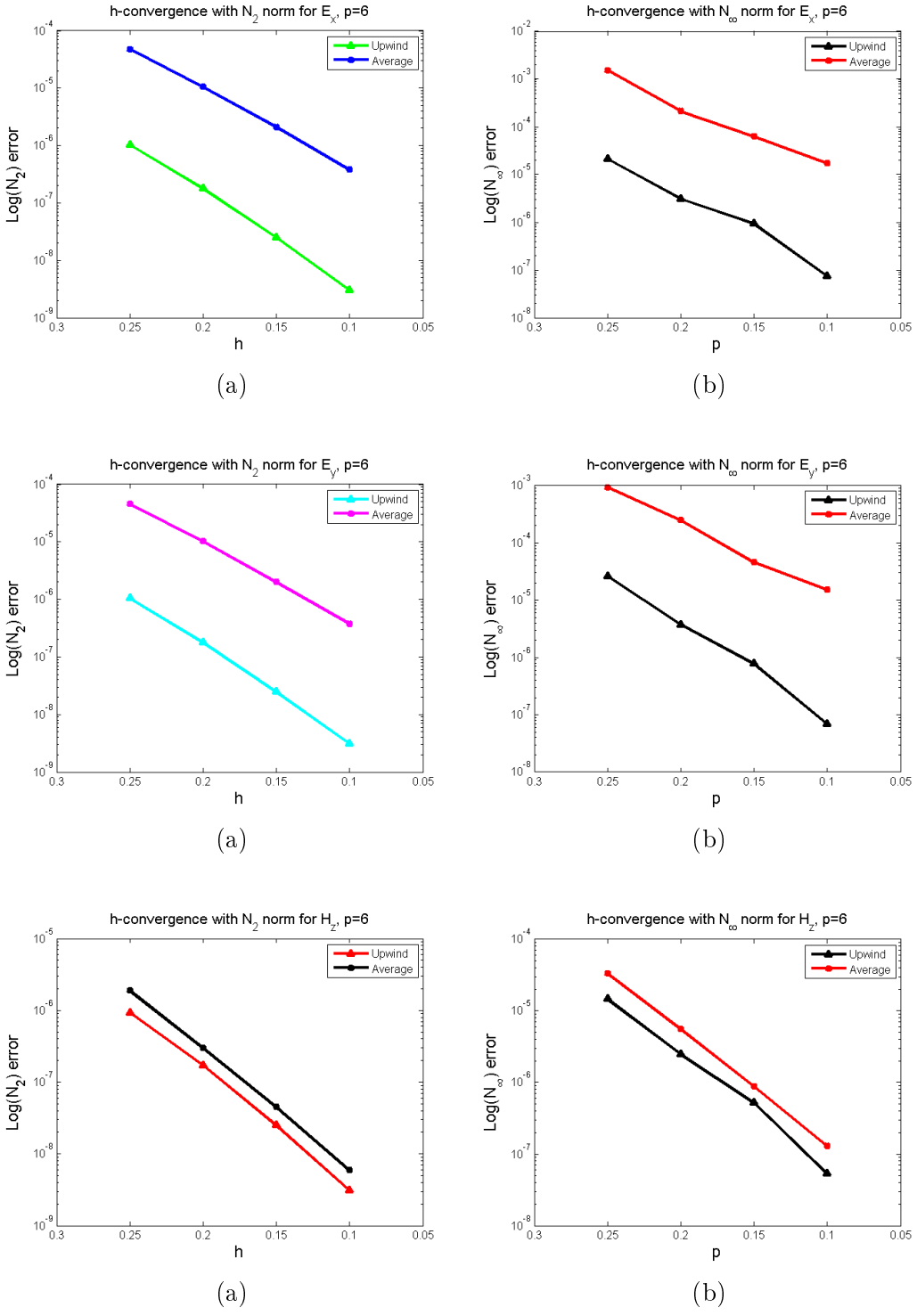
Table 4.21: Error computed with $p = 6$ for E_x

Norm	h	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	0.1	$3.14317\ 10^{-9}$	881.537	$3.78061\ 10^{-7}$	889.203
	0.15	$2.51605\ 10^{-8}$	449.47	$2.01321\ 10^{-6}$	452.433
	0.2	$1.80299\ 10^{-7}$	226.807	$1.03058\ 10^{-5}$	221.866
	0.25	$1.0647\ 10^{-6}$	143.911	$4.59796\ 10^{-5}$	146.395
N_∞	0.1	$6.94319\ 10^{-8}$	881.537	$1.52987\ 10^{-5}$	889.203
	0.15	$7.84627\ 10^{-7}$	449.47	$4.64402\ 10^{-5}$	452.433
	0.2	$3.76646\ 10^{-6}$	226.807	0.00025063	221.866
	0.25	$2.66893\ 10^{-5}$	143.911	0.000936095	146.395

Table 4.22: Error computed with $p = 6$ for E_y

Norm	h	Upwind	CPU Time(s)	Average	CPU Time(s)
N_2	0.1	$3.1452\ 10^{-9}$	881.537	$5.98045\ 10^{-9}$	889.203
	0.15	$2.52278\ 10^{-8}$	449.47	$4.54355\ 10^{-8}$	452.433
	0.2	$1.72199\ 10^{-7}$	226.807	$3.01223\ 10^{-7}$	221.866
	0.25	$9.31156\ 10^{-7}$	143.911	$1.90869\ 10^{-6}$	146.395
N_∞	0.1	$5.35009\ 10^{-8}$	881.537	$1.3057\ 10^{-7}$	889.203
	0.15	$5.25943\ 10^{-7}$	449.47	$8.84896\ 10^{-7}$	452.433
	0.2	$2.49775\ 10^{-6}$	226.807	$5.56985\ 10^{-6}$	221.866
	0.25	$1.48222\ 10^{-5}$	143.911	$3.34643\ 10^{-5}$	146.395

Table 4.23: Error computed with $p = 6$ for H_z



(a) (b)

(a) (b)

(a) (b)

Figure 4.24: h-convergence with N_2 and N_∞ norms for E_x(top), E_y(middle) and H_z(bottom) with upwind and centered numerical flux, polynomial degree $p = 6$.

4.2.9 Rate of convergence

For the space discretisation, the convergence rate of the method can be estimated for each degree of approximation. The degree of the polynomials p is fixed and for different values of the size of the mesh h, we compute the error δ. The convergence rate β is expressed by the following formula:

$$\beta = \frac{Log\left[\frac{\delta(h_{i+1})}{\delta(h_i)}\right]}{Log\left[\frac{h_i}{h_{i+1}}\right]}.$$

Where h_i and h_{i+1} are two different values of the size of the mesh and $\delta(h_i)$, $\delta(h_{i+1})$ the errors corresponding to a mesh of size h_i and h_{i+1} respectively. The tables (4.24) and (4.25) show the convergence rate β for different values of p for the **TM** and **TE** polarizations.

p	3	4	5	6	7	8
Upwind flux	3.26	4.23	5.56	6.44	7.71	8.37
Average flux	3.47	4.32	5.79	6.56	8.20	8.74

Table 4.24: Convergence rate of the method for the **TM** mode.

p	3	4	5	6	7	8
Upwind flux	3.39	4.23	5.57	6.43	7.76	8.39
Average flux	3.50	4.59	5.55	6.66	7.87	8.47

Table 4.25: Convergence rate of the method for the **TE** mode.

4.3 Analysis of the results

We have computed the numerical solution for different test problems in one and two dimensions. In one dimension, the computational domain consists of a medium with index of refraction $n = 1.0$. For different values of p and for fixed h, we evaluated the error between the exact solution and the numerical solution in N_2 and N_∞ norms. The temporal error is made negligible compared to the spatial error by choosing the time step as small as $\Delta t = 0.001$. The figures (4.1) and (4.2) show the p-convergence of the method for the centered numerical flux and the upwind numerical flux. As we increase the order of approximation, the error decreases exponentially for both norms as expected for spectral/hp methods. Moreover, we consider two media: one with index of refraction $n_1 = 1.0$ and one with index of refraction $n_2 = 1.5$. For different

hs and fixed $p = 6$, the error is computed with N_2 and N_∞ norms. The figures (4.3) and (4.5) show the h-convergence of the scheme for the centered numerical flux as well as for the upwind numerical flux. The more refined the mesh is, the less the error obtained for both norms. Similar computations can be done for the two-dimensional case with **TM** and **TE** modes. The p-convergence and the h-convergence are shown by the figures (4.9) and the figures (4.12) respectively for H_x, H_y and E_z . Similar convergence is achieved for E_x, E_y and H_z for the **TE** mode. We notice that increasing the order of approximation p or refining the grid increases the CPU time.

For all the test problems we have performed, the upwind numerical flux is more accurate than the centered numerical flux. Indeed, with the centered numerical flux there are spurious modes that are not adequately discarded which reduces the accuracy of the scheme([22]).

4.4 Application to the scattering of an electromagnetic wave

One of the principal goals of computational electromagnetics is the analysis of electromagnetic wave scattering which is well-known in radar technology. Firstly, we simulate the scattering of an electromagnetic wave by a circular cylinder with **PEC** boundary conditions by applying the code that we implemented for the two-dimensional Maxwell's equations. Secondly, we replace the circular cylinder by a rectangular perfect electric conductor. To do this, we use the total field/scattered field principle which consists of emitting an incident wave from a transmitter([8]). When the incident field reaches the scatterer, a so-called scattered field is generated and the main task is to approximate it. Since the electric field $\mathbf{E}$ and the magnetic field $\mathbf{H}$ are linear, they can be written in such a way that the total field($\mathbf{E}^{\text{tot}}$, $\mathbf{H}^{\text{tot}}$) is the sum of an incident field($\mathbf{E}^{\text{inc}}$, $\mathbf{H}^{\text{inc}}$) and a scattered field($\mathbf{E}^{\text{scat}}$, $\mathbf{H}^{\text{scat}}$):

$$\mathbf{E}^{\text{tot}} = \mathbf{E}^{\text{inc}} + \mathbf{E}^{\text{scat}}, \tag{4.4.1}$$

$$\mathbf{H}^{\text{tot}} = \mathbf{H}^{\text{inc}} + \mathbf{H}^{\text{scat}}. \tag{4.4.2}$$

We recall that for the **TM** mode the components of the electromagnetic field involved in the computation are H_x, H_y and E_z and the unknown vector $\mathbf{u}$ is $\mathbf{u} = \begin{pmatrix} H_x \\ H_y \\ E_z \end{pmatrix}$ while for the **TE** mode we have E_x, E_y, H_z and $\mathbf{u} = \begin{pmatrix} E_x \\ E_y \\ H_z \end{pmatrix}$. In the scattered form, $\mathbf{u}$ in both cases is written as follows:

$$\mathbf{u}^{\text{tot}} = \mathbf{u}^{\text{inc}} + \mathbf{u}^{\text{scat}}. \tag{4.4.3}$$

In order to perform the simulation, we need to send an electromagnetic wave to the scatterer(cylinder or rectangle), this is done by using the incident fields $\mathbf{E}^{\text{inc}}$ and $\mathbf{H}^{\text{inc}}$. Therefore, $\mathbf{E}^{\text{inc}}$ and $\mathbf{H}^{\text{inc}}$ are explicitly known.

For this problem, all the electromagnetic components are considered to be complex values. To generate a tapered wave(that is the amplitude of the wave decreases gradually as the wave progresses), we define a real factor $\alpha = \dfrac{2}{1+e^{\frac{1}{2}(|x-t|-1)}}$.

Thus, for the **TM** mode the incident fields are:

$$\begin{aligned} H_x^{\text{inc}} &= -\alpha\Big[\cos(f(x-t)) + i\sin(f(x-t))\Big], \\ H_y^{\text{inc}} &= -\alpha\Big[\cos(f(x-t)) + i\sin(f(x-t))\Big], \\ E_z^{\text{inc}} &= \alpha\Big[\cos(f(x-t)) + i\sin(f(x-t))\Big], \end{aligned}$$

and for the **TE** mode:

$$\begin{aligned} E_x^{\text{inc}} &= \alpha\Big[\cos(f(x-t)) + i\sin(f(x-t))\Big], \\ E_y^{\text{inc}} &= \alpha\Big[\cos(f(x-t)) + i\sin(f(x-t))\Big], \\ H_z^{\text{inc}} &= \alpha\Big[\cos(f(x-t)) + i\sin(f(x-t))\Big], \end{aligned}$$

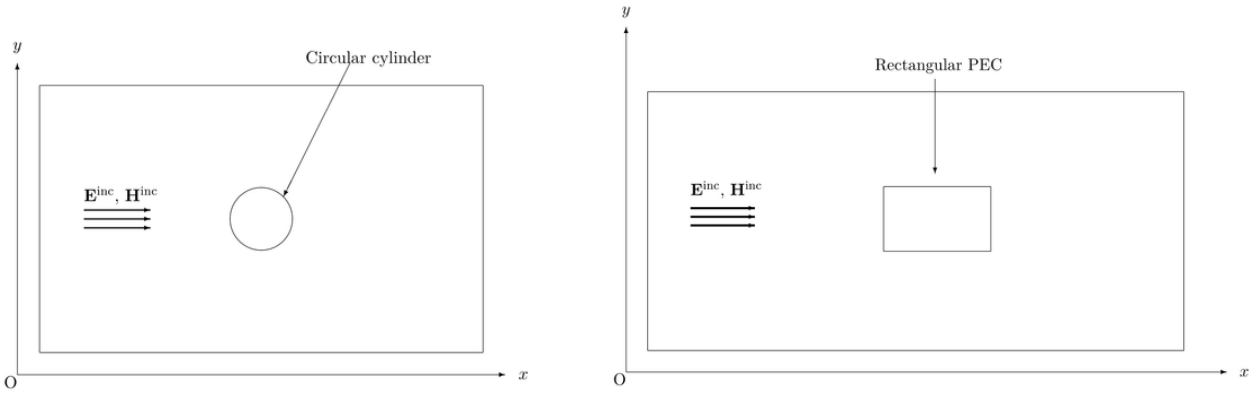
where f is the frequency of the wave in Hertz and i the complex imaginary number. The boundary condition in this problem is

$$\mathbf{n} \times \mathbf{E}^{\text{tot}} = 0. \tag{4.4.4}$$

Substituting $\mathbf{E}^{\text{tot}}$ by $\mathbf{E}^{\text{inc}} + \mathbf{E}^{\text{scat}}$ in (4.4.4), we obtain:

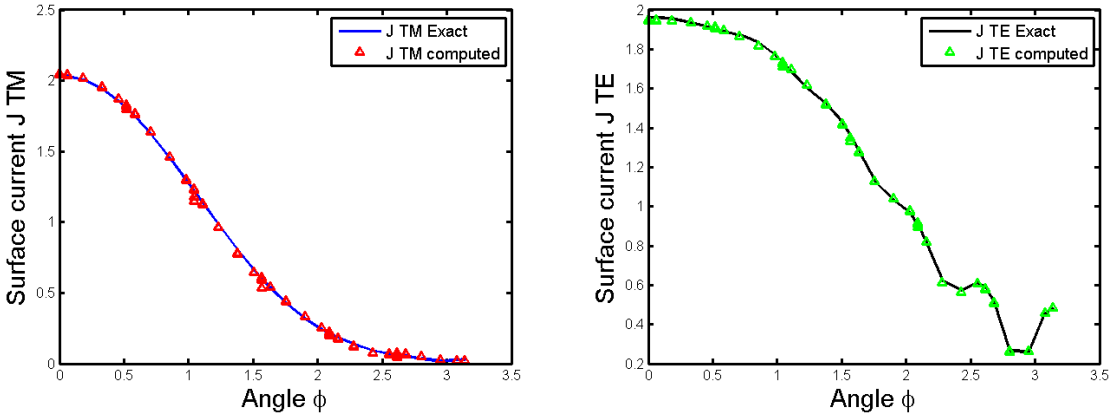
$$\begin{aligned} \mathbf{n} \times (\mathbf{E}^{\text{inc}} + \mathbf{E}^{\text{scat}}) &= 0, \\ \mathbf{n} \times \mathbf{E}^{\text{scat}} &= -\mathbf{n} \times \mathbf{E}^{\text{inc}}, \\ E^{\text{scat}} &= -E^{\text{inc}}. \end{aligned}$$

The parameters of integration are $\Delta t = 0.01$, number of steps Nsteps=3500 and final time $T = 35$. The propagation of the waves is assured by the time-stepping which carries on until the final time T is reached. When the waves encounter the cylinder, an induced current $\mathbf{J} = \mathbf{n} \times \mathbf{H}$ appears on the surface of the cylinder. The figures (4.26) and (4.30) show the exact and the approximate values of $\mathbf{J}$ for the **TM** and **TE** modes and E_z and E_x after the scattering. The red and blue colors highlight the propagation of the waves, and the grey color represents a shadowed area due to the reflection of waves.



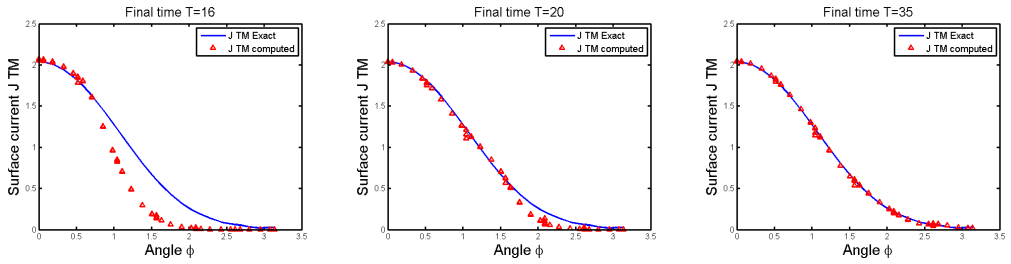
(a) Circular cylinder (b) Rectangular PEC

Figure 4.25: Domain of study for the scattering problem.



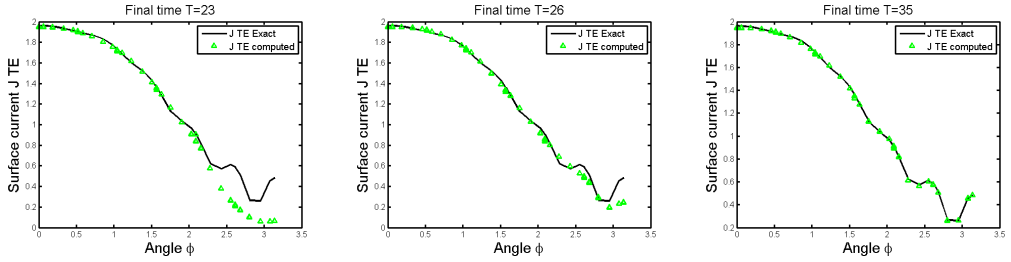
(a) TM mode (b) TE mode

Figure 4.26: Exact and approximate values of the surface current for **TM** and **TE** modes with the cylinder.



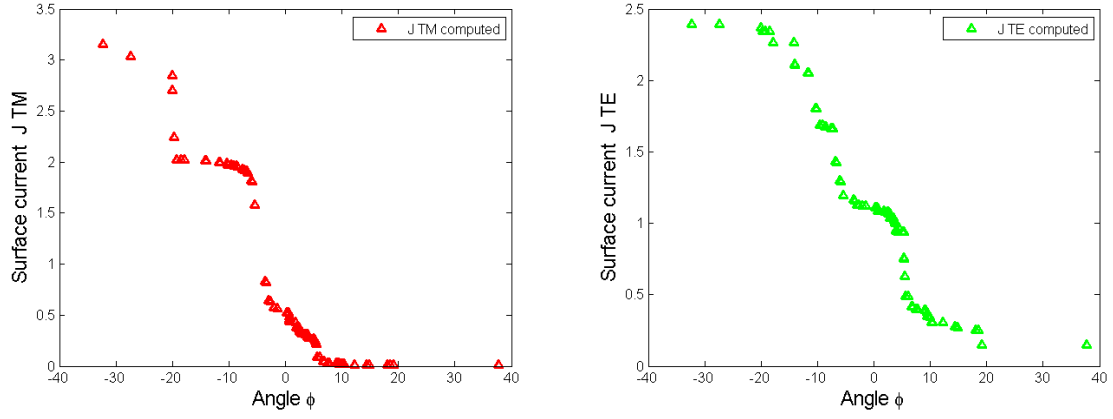
(a) T=16 (b) T=20 (c) T=35

Figure 4.27: Here we show how the approximate value approaches the exact value as T increases for the **TM** mode.



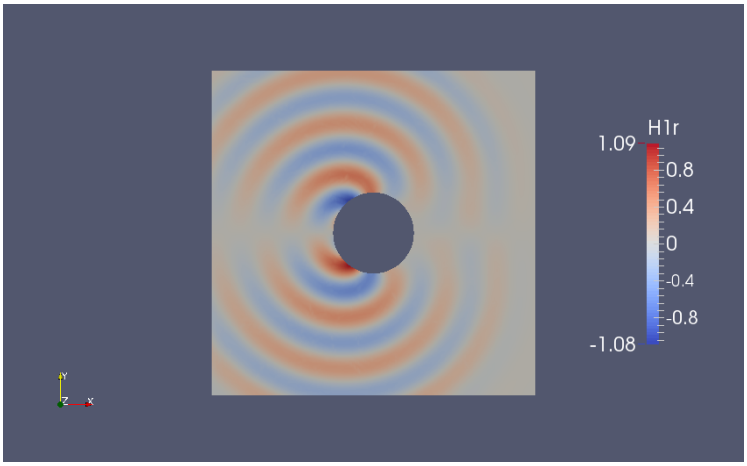
(a) T=23 (b) T=26 (c) T=35

Figure 4.28: Here we show how the approximate value approaches the exact value as T increases for the **TE** mode.

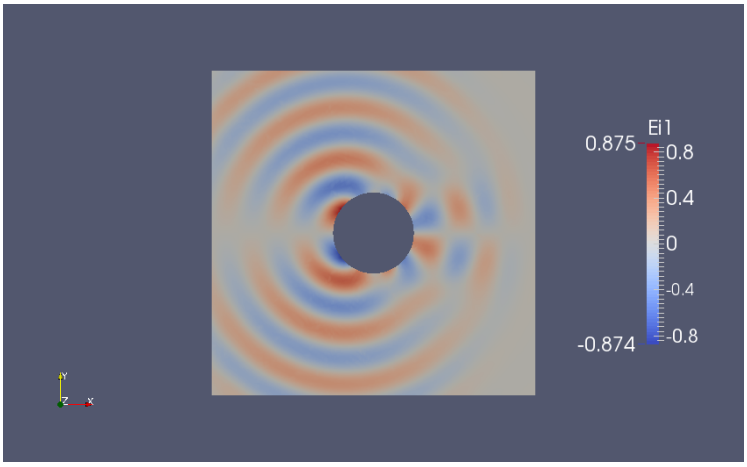


(a) TM mode (b) TE mode

Figure 4.29: Computed values of the surface current for **TM** and **TE** modes in the case of a rectangular PEC.



(a) H_x TM mode



(b) E_x TE mode

Figure 4.30: Using **Paraview** to visualize, the red and blue colors show the propagation of the waves and the grey color represents a shadowed area due to the reflection.

Chapter 5

Conclusion

Maxwell's equations outline the fundamental laws of electricity and magnetism which allow the electrification and the transportation of energy and information throughout the world. They consist of four complex equations and can be reduced to two dimensions and to one dimension in two modes: the transverse electric and the transverse magnetic modes. In addition to Maxwell's equations, we have the constitutive relations, which relate the electromagnetic fields **E** and **H** to the electric displacement field **D** and the magnetic flux **B** respectively. Along with the constitutive relations, Maxwell's equations describe the propagation of waves in a medium characterized by the electric permittivity ϵ, the magnetic permeability μ and the conductivity σ. The difficulty of solving these equations analytically prompted scientists to introduce numerical methods to be able to explore and analyse electromagnetic phenomena. Thus, after its introduction, the **FDTD** based on Yee's scheme became the most used method in computational electromagnetics. Years later, after realizing the weaknesses of the **FDTD**, scientists proposed new approaches in **CEM**, such as the finite element and the spectral methods. In this thesis, we used the spectral/hp discontinuous Galerkin method to solve Maxwell's equations in **TM** and **TE** polarisations. A brief definition of this method has been given in chapter(3).

For the space discretization, we used the discontinuous Galerkin method ending up with a semi-discrete scheme, which we integrated in time by using the Runge-Kutta fourth order method. Indeed, with the DG method, there are discontinuities at the interface between neighbouring elements in the mesh. Therefore, one must use a numerical flux to interconnect the local solutions. We have seen in chapter(3) that the numerical flux is a function of two states and is derived from the Rankine-Hugoniot condition. In the numerical scheme, we applied the two cases of Lax-Friedrichs numerical flux: the centered and the upwind numerical flux. It appears that the best approximation is achieved with the latter.

For all the test problems carried out in one dimension and in two dimensions, the results showed an exponential convergence for both numerical fluxes as

guaranteed by the spectral/*hp* discontinuous Galerkin method. We noticed that when we refine the grid or increase the degree of the approximating polynomials, the error evaluated with the N_2 and N_∞ norms decreases remarkably. With a fixed degree of polynomial p, the expected order of convergence of the method is $O(h^{p+1})$, h being a measure of the size of the elements in the mesh. This fact is confirmed by the tables (4.24) and (4.25) of the convergence rate in chapter(4).

We also noticed that when the frequency gets large, the difference between the exact solution and the approximate solution increases. This is due to the fact that the higher the frequency, the smaller the wavelength. Therefore, to obtain a better approximation, the mesh must be finer to resolve the smallest wavelength.

In the last part of the thesis, an application to the scattering of electromagnetic waves has been performed for both **TM** and **TE** modes. The tables (4.26a) and (4.26b) show a good approximation of the current flow induced on the scatterer. We observed that the longer the time integration, the better the approximation as shown on figures (4.27) and (4.28). When the dimension of the scatterer is very small, the effect of the scattering is almost unnoticeable. In contrast, when it is large, a shadowed region appears behind the scatterer, where the magnitude of the wave is tiny due to the reflection as depicted on the pictures (4.30a) and (4.30b).
For future work, one can study the accuracy of the scheme for electromagnetic scattering problems in cylindrical or spherical cavities or the application to waveguide analysis.

Appendices

Appendix A

Mathematical formulas

Given a vector field $\mathbf{A} = (A_x, A_y, A_z)$ in Cartesian coordinates $(\mathbf{i}, \mathbf{j}, \mathbf{k})$, we have:

1 $\nabla \cdot \mathbf{A} = \dfrac{\partial A_x}{\partial x} + \dfrac{\partial A_y}{\partial y} + \dfrac{\partial A_z}{\partial z}$.

2 $\nabla \times \mathbf{A} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ \dfrac{\partial}{\partial x} & \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial z} \\ A_x & A_y & A_z \end{vmatrix} = (\dfrac{\partial A_z}{\partial y} - \dfrac{\partial A_y}{\partial z})\mathbf{i} + (\dfrac{\partial A_x}{\partial z} - \dfrac{\partial A_z}{\partial x})\mathbf{j} + (\dfrac{\partial A_y}{\partial x} - \dfrac{\partial A_x}{\partial y})\mathbf{k}$.

Definition A.1 (Integration by parts) *Let $\Omega \subset \mathbb{R}^n$ be an open bounded set of $\mathbb{R}^n$ with a regular boundary $\partial\Omega$, and u, v two continuously differentiable functions on $\bar{\Omega}$. The formula for integrration by parts is*

$$\int_\Omega \frac{\partial u}{\partial x_i} v dx = \int_{\partial\Omega} uvn_i d\sigma - \int_\Omega u \frac{\partial v}{\partial x_i} dx.$$

Where n_i is the i-th component of the unit outward normal $\mathbf{n}$ to $\partial\Omega$.
Replacing v by v_i and summing over i in the above formula gives the vector formula

$$\int_\Omega \nabla u \cdot \mathbf{v} dx = \int_{\partial\Omega} u(\mathbf{v} \cdot \mathbf{n}) d\sigma - \int_\Omega u \nabla \cdot \mathbf{v} dx.$$

Where $\mathbf{v} = (v_1, \cdots, v_n)$ is a vector valued function.

Definition A.2 *Let $\mathbf{\Phi} : \mathbb{R}^m \longrightarrow \mathbb{R}^n$ be a matrix-valued function, the divergence $\nabla \cdot \mathbf{\Phi}$ is defined as:*

$$\nabla \cdot \mathbf{\Phi} = \begin{pmatrix} \dfrac{\partial \mathbf{\Phi}_{11}}{\partial x_1} + \cdots + \dfrac{\partial \mathbf{\Phi}_{1m}}{\partial x_m} \\ \vdots \\ \dfrac{\partial \mathbf{\Phi}_{n1}}{\partial x_1} + \cdots + \dfrac{\partial \mathbf{\Phi}_{nm}}{\partial x_m} \end{pmatrix}$$

Theorem A.3 (Stokes) *The circulation of a vector field over a closed path* $\mathcal{C}$ *is equal to the integral of the normal component of the curl of that field over a surface* $\mathbf{S}$ *enclosed by the contour* $\mathcal{C}$*. Mathematically this statement is written as follows:*

$$\oint_{\mathcal{C}} \mathbf{A} \cdot \mathbf{dl} = \int_{\mathbf{S}} (\nabla \times \mathbf{A}) \cdot \mathbf{dS}.$$

Theorem A.4 (Green divergence theorem) *The flux of a vector field through a closed surface* $\mathbf{S}$ *is equal to the integral of the divergence of that field over a volume* $\mathbf{V}$ *for which* $\mathbf{S}$ *is a boundary.*
Mathematically:

$$\oint_{\mathbf{S}} \mathbf{A} \cdot \mathbf{dS} = \int_{\mathbf{V}} (\nabla \cdot \mathbf{A})\, \mathbf{dV}.$$

Appendix B

The Nektar++ library

This appendix is intended for the installation and the use of `Nektar++` framework which can be utilized to solve partial differential equations, in particular hyperbolic conservation laws.

B.1 Installation of Nektar++

For the installation of `Nektar++`, one can visit the the website **http://www.nektar.info** which contains all the latest versions of the software.
To configure `Nektar++-3.4.0` version which we have used for our problem, the reader can follow the link **http://www.nektar.info/wiki/3.4/UserGuide/Compile/** for a list of directives for any system either `Linux`, `Windows` or `Mac`.

B.2 Use of Nektar++

After installing `Nektar++` on the system, the source code is contained in the folder named `Nektar++-3.4.0` in our case, which itself comprises several folders.
One of the significant folders is `Nektar++-3.4.0/builds` which, itself contains seven folders. Among them, `Nektar++-3.4.0/builds/solvers` and `Nektar++-3.4.0/builds/utilities`

- ⋆ **Nektar++-3.4.0/builds/solvers**:
  All the solvers currently implemented in `Nektar++` are located in this folder.
  On Linux system, to use one of these solvers, on the terminal one has to move into the directory containing the solver of interest and the **.xml** file and run it.
  For instance in our case the solver that we are using is **Maxwellsolver-g-3.4.0** and the command line is:
  *./Maxwellsolver-g-3.4.0 **.xml*

⋆ **Nektar++-3.4.0/builds/utilities**
This folder contains two subfolders that are relevant for the steps before running the **.xml** file and the process after the **.xml** file is run.
These folders are `Nektar++-3.4.0/builds/utilities/Preprocessing` and `Nektar++-3.4.0/builds/utilities/Postprocessing`

- **Nektar++-3.4.0/builds/utilities/Preprocessing**
To generate the mesh, we follow the following procedure: from the **.geo** file, we make the **.msh** file using **Gmsh**. Afterwards, we put the **.msh** file into `Meshconvert/` which contains an executable `Meshconvert-g-3.4.0`.
Using the following command line, we make the **.xml** file:
*./Meshconvert-g-3.4.0 **.msh **.xml*
- **Nektar++-3.4.0/builds/utilities/Postprocessing**
Running the ***.xml* file with a solver generates several ***.chk* files and one ***.fld* file.
For the visualization of the output by `Paraview` for instance, one needs to convert these files into the format ***.vtu*. The command lines for the conversion are:
*./FldToVtk **.xml **.fld*
*./FldToVtk **.xml **.chk*

B.3 Functions used in the code

Here are some functions that we used in the code:

- *GetExpsize()*
Get the number of points in the domain.
- *GetTotPoints()*
Get the quadrature points and weights required for integration.
- *GetNumPoints()*
Get the number of points in the expansion.
- *GetPhysNormals()*
Get the number of normals in the expansion.
- *FwdTrans()*
enables to pass from the physical space to the modal space.
- *BwdTrans()*
It does the opposite of *FwdTrans()*: going from the modal space back to the physical space.

- *CopyBoundaryTrace()*
  Copy the forward trace of the field to the backward trace.

- *m_traceNormals*
  Array holding the forward normals.

- *Physderiv()*
  evaluates the derivative of a function $u(x)$ with respect to x .

- *PhysIntegral()*
  This function integrates a function $u(x)$ over the domain consisting of all elements of the expansion:

$$\int_{\Omega} u(x)\mathrm{d}x = \sum_{k=1}^{N_{el}} \left\{ \int_{\Omega_k} u(x)\mathrm{d}x \right\}.$$

- *IProductWRTDerivBase()*
  Calculates the inner product of a function $u(x)$ with respect to the derivative of the modal function:

$$\int_{\Omega} u(x) \frac{\partial \phi}{\partial x}(x)\ \mathrm{d}x.$$

- *WeakDGAdvection()*
  Calculates the weak discontinuous Galerkin advection of the form:

$$(\mathbf{n} \cdot \hat{\mathbf{F}}, \phi) - (\mathbf{F} \cdot \nabla \phi)\,.$$

- *EvaluateMaxwell2DTM()*
  evaluates the exact solution for the **TM** polarization.

- *EvaluateMaxwell2DTE()*
  evaluates the exact solution for the **TE** polarization.

- *SetinitialMaxwell2D()*
  Sets the initial conditions for the 2D test problem.

- *Numericalflux1D()*
  Defines the numerical flux for the 1D test problem.

- *Numericalflux2DTM()*
  Defines the numerical flux for the 2D test problem in **TM** form according to the type, that is Upwind or Average.

- *Numericalflux2DTE()*
  Defines the numerical flux for the 2D test problem in **TE** form.

- *DoOdeRhs()*
  Computes the right hand side of the ODE which represents all the terms expect the one that contains $\frac{\partial u}{\partial t}$.

- *DoOdeProjection()*
  Carries out the projection for the unsteady diffusion problem, either Galerkin or Discontinuous Galerkin.

- *Getfluxvector()*
  Returns the flux vector for the unsteady diffusion problem.

- *ExtractTracePhys(A, B)*
  This function extracts the trace(on the edges) from the field A and puts the values in B.

Appendix C

Nektar++ code for Maxwell's equations

Here is the *cpp* file of the code: *LinearMaxwell.cpp*.

```
#include <iostream>
#include <iomanip>
#include <cstdio>
#include <cstdlib>
#include <ctime>

#include <MultiRegions/AssemblyMap/AssemblyMapDG.h>
#include <LibUtilities/TimeIntegration/TimeIntegrationScheme.h>
#include <MaxwellSolver/EquationSystems/LinearMaxwell.h>

namespace Nektar
{
  string LinearMaxwell::className =SolverUtils::GetEquationSystemFactory().RegisterCreatorFunction(
                                                              "LinearMaxwell", LinearMaxwell::create,
                                                    "Linear Maxwell equation in primitive variables.");

  LinearMaxwell::LinearMaxwell(const LibUtilities::SessionReaderSharedPtr& pSession): MaxwellSystem(pSession)
  {
  }

  void LinearMaxwell::v_InitObject()
  {
    m_PI = 3.14159265358979323846;

    MaxwellSystem::v_InitObject();

        m_session->LoadParameter("Nend", m_Nend, 1000);
        m_session->LoadParameter("alpha", m_alpha, 1.0);

        // Local index of refaction for 1D test
        m_session->LoadParameter("n1", m_n1, 1.0);
        m_session->LoadParameter("n2", m_n2, 1.0);

        m_session->LoadParameter("varepsilon1", m_varepsilon1, 1.0);
        m_session->LoadParameter("varepsilon2", m_varepsilon2, 1.0);

        m_session->LoadParameter("mu1", m_mu1, 1.0);
        m_session->LoadParameter("mu2", m_mu2, 1.0);


      if(m_session->DefinesSolverInfo("PROBLEMTYPE"))
```

```
        {
          int i;
          std::string ProblemTypeStr = m_session->GetSolverInfo("PROBLEMTYPE");
          for (i = 0; i < (int) SIZE_ProblemType; ++i)
            {
              if (boost::iequals(ProblemTypeMap[i], ProblemTypeStr))
                {
                  m_problemType = (ProblemType)i;
                  break;
                }
            }
        }
      else
        {
         m_problemType = (ProblemType)0;
        }

      int i;
      // Upwind type
      for (i = 0; i < (int)SIZE_UpwindType; ++i)
        {
          bool match;
          m_session->MatchSolverInfo("UPWINDTYPE", UpwindTypeMap[i], match, false);
          if (match)
           {
              m_upwindType = (UpwindType) i;
              break;
           }
        }


    if (m_explicitAdvection)
      {
        m_ode.DefineOdeRhs     (&LinearMaxwell::DoOdeRhs,        this);
        m_ode.DefineProjection (&LinearMaxwell::DoOdeProjection, this);
      }
    else
      {
       ASSERTL0(false, "Implicit not set up.");
      }

  }

  LinearMaxwell::~LinearMaxwell()
  {
  }

// Sets initial conditions
 void LinearMaxwell::v_SetInitialConditions(NekDouble initialtime, bool dumpInitialConditions)
 {
     switch (m_problemType)
      {
        case eMaxwell1D:
          {
            SetInitialMaxwell1D(initialtime);
          }
          break;
        case eMaxwell2DTM:
          {
            SetInitialMaxwell2DTM(initialtime);
          }
          break;
        case eMaxwell2DTE:
          {
            SetInitialMaxwell2DTE(initialtime);
          }
```

```
          break;

        default:
          {
            EquationSystem::v_SetInitialConditions(initialtime, false);
          break;
          }
      }

      if (dumpInitialConditions)
        {
          // Dump initial conditions to file
          Checkpoint_Output(0);
        }
 }
 //Set initial conditions for 1D test problem
 void LinearMaxwell::SetInitialMaxwell1D(NekDouble initialtime)
 {
      int nTotQuadPoints  = GetTotPoints();

      Array<OneD, NekDouble> E0(nTotQuadPoints);
      Array<OneD, NekDouble> H0(nTotQuadPoints);

      EvaluateMaxwell1D(initialtime,0,E0);
      m_fields[0]->SetPhys(E0);

      EvaluateMaxwell1D(initialtime,1,H0);
      m_fields[1]->SetPhys(H0);

      Vmath::Vcopy(nTotQuadPoints, E0, 1, m_fields[0]->UpdatePhys(), 1);
      Vmath::Vcopy(nTotQuadPoints, H0, 1, m_fields[1]->UpdatePhys(), 1);


      // Forward transform to fill the coefficient space
      for(int i = 0; i < m_fields.num_elements(); ++i)
      {
        m_fields[i]->SetPhysState(true);
        m_fields[i]->FwdTrans(m_fields[i]->GetPhys(), m_fields[i]->UpdateCoeffs());
      }
  }
 //Set initial conditions for 2D Transverse Magnetic test problem
void LinearMaxwell::SetInitialMaxwell2DTM(NekDouble initialtime)
 {
   int nq = GetTotPoints();
   int nTotQuadPoints = GetTotPoints();

   Array<OneD, NekDouble> Hx0(nq);
   Array<OneD, NekDouble> Hy0(nq);
   Array<OneD, NekDouble> Ez0(nq);

   EvaluateMaxwell2DTM(initialtime, 0, Hx0);
   m_fields[0]->SetPhys(Hx0);

   EvaluateMaxwell2DTM(initialtime, 1, Hy0);
   m_fields[1]->SetPhys(Hy0);

   EvaluateMaxwell2DTM(initialtime, 2, Ez0);
   m_fields[2]->SetPhys(Ez0);

   Vmath::Vcopy(nTotQuadPoints, Hx0, 1, m_fields[0]->UpdatePhys(), 1);
   Vmath::Vcopy(nTotQuadPoints, Hy0, 1, m_fields[1]->UpdatePhys(), 1);
   Vmath::Vcopy(nTotQuadPoints, Ez0, 1, m_fields[2]->UpdatePhys(), 1);

   // Forward transform to fill the coefficient space
   for(int i = 0; i < m_fields.num_elements(); ++i)
     {
```

```
        m_fields[i]->SetPhysState(true);
        m_fields[i]->FwdTrans(m_fields[i]->GetPhys(), m_fields[i]->UpdateCoeffs());
     }
 }
 //Set initial conditions for 2D Transverse Electric test problem
void LinearMaxwell::SetInitialMaxwell2DTE(NekDouble initialtime)
{
  int nq = GetTotPoints();
  int nTotQuadPoints = GetTotPoints();

  Array<OneD, NekDouble> Ex0(nq);
  Array<OneD, NekDouble> Ey0(nq);
  Array<OneD, NekDouble> Hz0(nq);

  EvaluateMaxwell2DTE(initialtime, 0, Ex0);
  m_fields[0]->SetPhys(Ex0);

  EvaluateMaxwell2DTE(initialtime, 1, Ey0);
  m_fields[1]->SetPhys(Ey0);

  EvaluateMaxwell2DTE(initialtime, 2, Hz0);
  m_fields[2]->SetPhys(Hz0);

  Vmath::Vcopy(nTotQuadPoints, Ex0, 1, m_fields[0]->UpdatePhys(), 1);
  Vmath::Vcopy(nTotQuadPoints, Ey0, 1, m_fields[1]->UpdatePhys(), 1);
  Vmath::Vcopy(nTotQuadPoints, Hz0, 1, m_fields[2]->UpdatePhys(), 1);

  // Forward transform to fill the coefficient space
  for(int i = 0; i < m_fields.num_elements(); ++i)
   {
     m_fields[i]->SetPhysState(true);
     m_fields[i]->FwdTrans(m_fields[i]->GetPhys(), m_fields[i]->UpdateCoeffs());
   }
}

// Evaluates exact solutions for test problems
 void LinearMaxwell::v_EvaluateExactSolution(unsigned int                        field,
                                             Array<OneD, NekDouble>             &outfield,
                                             const NekDouble                    time)
 {
   switch(m_problemType)
      {
       case eMaxwell1D:
        {
         EvaluateMaxwell1D(time, field, outfield);
        }
       break;

       case eMaxwell2DTM:
        {
          EvaluateMaxwell2DTM(time, field, outfield);
        }
       break;
       case eMaxwell2DTE:
        {
          EvaluateMaxwell2DTE(time, field, outfield);
        }
       break;

       default:
        {
         break;
        }
      }
 }
```

```
//Evaluates exact solution for 1D test problem
 void LinearMaxwell::EvaluateMaxwell1D(const NekDouble time, unsigned int field,
                                       Array<OneD, NekDouble> &outfield)
 {
     int nq = m_fields[0]->GetNpoints();
     int nTraceNumPoints = GetTraceNpoints();

     Array<OneD,NekDouble> x0(nq);
     Array<OneD,NekDouble> x1(nq);
     Array<OneD,NekDouble> x2(nq);

     m_fields[0]->GetCoords(x0,x1,x2);

     // generate Permittivity and Permeability tensor
     m_varepsilon = Array<OneD, NekDouble>(nq);
     m_mu = Array<OneD, NekDouble>(nq);

     NekDouble mu1, mu2;
     mu1=1.0;
     mu2=1.0;

     NekDouble eps1, eps2;
     eps1 = m_n1*m_n1;
     eps2 = m_n2*m_n2;

     for (int i=0; i<nq; ++i)
      {
       if(x0[i]>0.0)
         {
          m_varepsilon[i] = eps2;
          m_mu[i] = 1.0;
         }

       else
         {
          m_varepsilon[i] = eps1;
          m_mu[i] = 1.0;
         }
      }

     // Generate the impedance Zim and the conductance Yim
     Array<OneD, NekDouble> Fwdeps(nTraceNumPoints);
     Array<OneD, NekDouble> Bwdeps(nTraceNumPoints);

     Array<OneD, NekDouble> Fwdmu(nTraceNumPoints);
     Array<OneD, NekDouble> Bwdmu(nTraceNumPoints);

     Array<OneD, NekDouble> x0Fwd(nTraceNumPoints);
     Array<OneD, NekDouble> x0Bwd(nTraceNumPoints);

     Array<OneD, NekDouble> x1Fwd(nTraceNumPoints);
     Array<OneD, NekDouble> x1Bwd(nTraceNumPoints);

     m_ZimFwd = Array<OneD, NekDouble>(nTraceNumPoints);
     m_ZimBwd = Array<OneD, NekDouble>(nTraceNumPoints);

     m_YimFwd = Array<OneD, NekDouble>(nTraceNumPoints);
     m_YimBwd = Array<OneD, NekDouble>(nTraceNumPoints);

     m_fields[0]->GetFwdBwdTracePhys(0,m_Nend,eps1,eps2,Fwdeps,Bwdeps);
     m_fields[0]->GetFwdBwdTracePhys(0,m_Nend,mu1,mu2,Fwdmu,Bwdmu);

     m_fields[0]->GetFwdBwdTracePhys(x0, x0Fwd, x0Bwd);
     m_fields[0]->GetFwdBwdTracePhys(x1, x1Fwd, x1Bwd);
```

```
BoundaryFwdBwdConditions(Fwdeps,Bwdeps,SpatialDomains::ePEC,eFwdEQBwd);
BoundaryFwdBwdConditions(Fwdmu,Bwdmu,SpatialDomains::ePEC,eFwdEQBwd);
BoundaryFwdBwdConditions(Fwdeps,Bwdeps,SpatialDomains::eTransParent,eFwdEQBwd);
BoundaryFwdBwdConditions(Fwdmu,Bwdmu,SpatialDomains::eTransParent,eFwdEQBwd);

// ZimFwd = sqrt( muFwd / epsFwd),  ZimBwd = sqrt( muBwd / epsBwd)
Vmath::Vdiv(nTraceNumPoints, Fwdmu, 1, Fwdeps, 1, m_ZimFwd, 1);
Vmath::Vdiv(nTraceNumPoints, Bwdmu, 1, Bwdeps, 1, m_ZimBwd, 1);
Vmath::Vsqrt(nTraceNumPoints, m_ZimFwd, 1, m_ZimFwd, 1);
Vmath::Vsqrt(nTraceNumPoints, m_ZimBwd, 1, m_ZimBwd, 1);

// YimFwd = sqrt( epsFwd / muFwd),   YimBwd = sqrt( epsBwd / muBwd)
Vmath::Vdiv(nTraceNumPoints, Fwdeps, 1, Fwdmu, 1, m_YimFwd, 1);
Vmath::Vdiv(nTraceNumPoints, Bwdeps, 1, Bwdmu, 1, m_YimBwd, 1);
Vmath::Vsqrt(nTraceNumPoints, m_YimFwd, 1, m_YimFwd, 1);
Vmath::Vsqrt(nTraceNumPoints, m_YimBwd, 1, m_YimBwd, 1);

Array<OneD, NekDouble> E(nq);
Array<OneD, NekDouble> H(nq);
// Derive the frequency \omega
NekDouble omega;
NekDouble Tol = 0.000000001;
if(fabs(m_n1-m_n2)<Tol)
  {
   omega = m_PI/m_n1;
  }

else
  {
    omega = 2.0*m_PI/m_n2;

    NekDouble newomega, F, Fprime;
    for (int i=0; i<10000; ++i)
      {
        F = m_n1*tan(m_n2*omega)+m_n2*tan(m_n1*omega);
        Fprime = m_n1*m_n2*( 1.0/cos(m_n2*omega)/cos(m_n2*omega) + 1.0/cos(m_n1*omega)/cos(m_n1*omega));

        newomega = omega - F/Fprime;

        if(fabs(newomega - omega)> Tol)
          {
           omega = newomega;
          }

        else
          {
           break;
          }
       }
  }

// Generate A^k and B^k
std::complex<double> im = sqrt( complex<double>(-1) );
std::complex<double> A1, A2, B1, B2;
std::complex<double> Ak, Bk, nk;
std::complex<double> Ec, Hc;

A1 = m_n2*cos(m_n2*omega)/(m_n1*cos(m_n1*omega));
A2 = exp(-1.0*im*omega*(m_n1+m_n2));
B1 = A1*exp(-2.0*im*m_n1*omega);
B2 = A2*exp(2.0*im*m_n2*omega);

for (int i=0; i<nq; ++i)
 {
  if(x0[i]>0)
   {
```

```
            Ak = A2;
            Bk = B2;
            nk = m_n2;
          }

         else
          {
           Ak = A1;
           Bk = B1;
           nk = m_n1;
          }

         Ec = (Ak*exp(im*nk*omega*x0[i]) - Bk*exp(-im*nk*omega*x0[i]))*exp(im*omega*time);
         Hc = nk*(Ak*exp(im*nk*omega*x0[i]) + Bk*exp(-im*nk*omega*x0[i]))*exp(im*omega*time);

         E[i] = Ec.real();
         H[i] = Hc.real();
        }

       switch(field)
        {
         case(0):
          {
           outfield = E;
          }
          break;

         case(1):
          {
          outfield = H;
          }
          break;
        }
  }
 //Evaluates exact solution for 2D TM test problem
void LinearMaxwell::EvaluateMaxwell2DTM(const NekDouble time, unsigned int field,
                                        Array<OneD, NekDouble> &outfield)
  {
      int nq = m_fields[0]->GetNpoints();
      int nTraceNumPoints = GetTraceNpoints();

      Array<OneD,NekDouble> x0(nq);
      Array<OneD,NekDouble> x1(nq);
      Array<OneD,NekDouble> x2(nq);

      m_fields[0]->GetCoords(x0,x1,x2);

      NekDouble eps1,eps2,mu1,mu2;
      eps1=1.0;
      eps2=1.0;

      mu1=1.0;
      mu2=1.0;

      // Generate the impedance Zim and the conductance Yim
      Array<OneD, NekDouble> Fwdeps(nTraceNumPoints);
      Array<OneD, NekDouble> Bwdeps(nTraceNumPoints);

      Array<OneD, NekDouble> Fwdmu(nTraceNumPoints);
      Array<OneD, NekDouble> Bwdmu(nTraceNumPoints);

      Array<OneD, NekDouble> x0Fwd(nTraceNumPoints);
      Array<OneD, NekDouble> x0Bwd(nTraceNumPoints);

      Array<OneD, NekDouble> x1Fwd(nTraceNumPoints);
      Array<OneD, NekDouble> x1Bwd(nTraceNumPoints);
```

```
Array<OneD, NekDouble> x2Fwd(nTraceNumPoints);
Array<OneD, NekDouble> x2Bwd(nTraceNumPoints);

m_ZimFwd = Array<OneD, NekDouble>(nTraceNumPoints);
m_ZimBwd = Array<OneD, NekDouble>(nTraceNumPoints);

m_YimFwd = Array<OneD, NekDouble>(nTraceNumPoints);
m_YimBwd = Array<OneD, NekDouble>(nTraceNumPoints);

m_fields[0]->GetFwdBwdTracePhys(0,m_Nend,eps1,eps2,Fwdeps,Bwdeps);
m_fields[0]->GetFwdBwdTracePhys(0,m_Nend,mu1,mu2,Fwdmu,Bwdmu);

m_fields[0]->GetFwdBwdTracePhys(x0, x0Fwd, x0Bwd);
m_fields[0]->GetFwdBwdTracePhys(x1, x1Fwd, x1Bwd);
m_fields[0]->GetFwdBwdTracePhys(x2, x2Fwd, x2Bwd);

BoundaryFwdBwdConditions(Fwdeps,Bwdeps,SpatialDomains::ePEC,eFwdEQBwd);
BoundaryFwdBwdConditions(Fwdmu,Bwdmu,SpatialDomains::ePEC,eFwdEQBwd);

// ZimFwd = sqrt( muFwd / epsFwd),  ZimBwd = sqrt( muBwd / epsBwd)
Vmath::Vdiv(nTraceNumPoints, Fwdmu, 1, Fwdeps, 1, m_ZimFwd, 1);
Vmath::Vdiv(nTraceNumPoints, Bwdmu, 1, Bwdeps, 1, m_ZimBwd, 1);
Vmath::Vsqrt(nTraceNumPoints, m_ZimFwd, 1, m_ZimFwd, 1);
Vmath::Vsqrt(nTraceNumPoints, m_ZimBwd, 1, m_ZimBwd, 1);

// YimFwd = sqrt( epsFwd / muFwd),   YimBwd = sqrt( epsBwd / muBwd)
Vmath::Vdiv(nTraceNumPoints, Fwdeps, 1, Fwdmu, 1, m_YimFwd, 1);
Vmath::Vdiv(nTraceNumPoints, Bwdeps, 1, Bwdmu, 1, m_YimBwd, 1);
Vmath::Vsqrt(nTraceNumPoints, m_YimFwd, 1, m_YimFwd, 1);
Vmath::Vsqrt(nTraceNumPoints, m_YimBwd, 1, m_YimBwd, 1);

NekDouble freqm, freqn, omega;

freqm = 1.0;
freqn = 1.0;

omega = m_PI*sqrt(freqm*freqm+freqn*freqn);

Array<OneD, NekDouble> Hx(nq);
Array<OneD, NekDouble> Hy(nq);
Array<OneD, NekDouble> Ez(nq);

NekDouble mpi, npi;
mpi = freqm*m_PI;
npi = freqn*m_PI;

for (int i=0; i<nq; ++i)
  {

    Hx[i] = -1.0*(npi/omega)*sin(mpi*x0[i])*cos(npi*x1[i])*sin(omega*time);
    Hy[i] = (mpi/omega)*cos(mpi*x0[i])*sin(npi*x1[i])*sin(omega*time);
    Ez[i] = sin(mpi*x0[i])*sin(npi*x1[i])*cos(omega*time);
  }

switch(field)
 {
   case(0):
    {
     outfield = Hx;
    }
    break;

   case(1):
    {
     outfield = Hy;
```

```
        }
        break;

        case(2):
        {
         outfield = Ez;
        }
        break;
      }
}
 //Evaluates exact solution for 2D TE test problem
void LinearMaxwell::EvaluateMaxwell2DTE(const NekDouble time, unsigned int field,
                                        Array<OneD, NekDouble> &outfield)
{
    int nq = m_fields[0]->GetNpoints();
    int nTraceNumPoints = GetTraceNpoints();

    Array<OneD,NekDouble> x0(nq);
    Array<OneD,NekDouble> x1(nq);
    Array<OneD,NekDouble> x2(nq);

    m_fields[0]->GetCoords(x0,x1,x2);

    NekDouble eps1,eps2,mu1,mu2;
    eps1=1.0;
    eps2=1.0;

    mu1=1.0;
    mu2=1.0;

    // Generate the impedance Zim and the conductance Yim
    Array<OneD, NekDouble> Fwdeps(nTraceNumPoints);
    Array<OneD, NekDouble> Bwdeps(nTraceNumPoints);

    Array<OneD, NekDouble> Fwdmu(nTraceNumPoints);
    Array<OneD, NekDouble> Bwdmu(nTraceNumPoints);

    Array<OneD, NekDouble> x0Fwd(nTraceNumPoints);
    Array<OneD, NekDouble> x0Bwd(nTraceNumPoints);

    Array<OneD, NekDouble> x1Fwd(nTraceNumPoints);
    Array<OneD, NekDouble> x1Bwd(nTraceNumPoints);

    Array<OneD, NekDouble> x2Fwd(nTraceNumPoints);
    Array<OneD, NekDouble> x2Bwd(nTraceNumPoints);

    m_ZimFwd = Array<OneD, NekDouble>(nTraceNumPoints);
    m_ZimBwd = Array<OneD, NekDouble>(nTraceNumPoints);

    m_YimFwd = Array<OneD, NekDouble>(nTraceNumPoints);
    m_YimBwd = Array<OneD, NekDouble>(nTraceNumPoints);

    m_fields[0]->GetFwdBwdTracePhys(0,m_Nend,eps1,eps2,Fwdeps,Bwdeps);
    m_fields[0]->GetFwdBwdTracePhys(0,m_Nend,mu1,mu2,Fwdmu,Bwdmu);

    m_fields[0]->GetFwdBwdTracePhys(x0, x0Fwd, x0Bwd);
    m_fields[0]->GetFwdBwdTracePhys(x1, x1Fwd, x1Bwd);
    m_fields[0]->GetFwdBwdTracePhys(x2, x2Fwd, x2Bwd);

    BoundaryFwdBwdConditions(Fwdeps,Bwdeps,SpatialDomains::ePEC,eFwdEQBwd);
    BoundaryFwdBwdConditions(Fwdmu,Bwdmu,SpatialDomains::ePEC,eFwdEQBwd);

    // ZimFwd = sqrt( muFwd / epsFwd),  ZimBwd = sqrt( muBwd / epsBwd)
    Vmath::Vdiv(nTraceNumPoints, Fwdmu, 1, Fwdeps, 1, m_ZimFwd, 1);
    Vmath::Vdiv(nTraceNumPoints, Bwdmu, 1, Bwdeps, 1, m_ZimBwd, 1);
    Vmath::Vsqrt(nTraceNumPoints, m_ZimFwd, 1, m_ZimFwd, 1);
```

```
      Vmath::Vsqrt(nTraceNumPoints, m_ZimBwd, 1, m_ZimBwd, 1);

      // YimFwd = sqrt( epsFwd / muFwd),   YimBwd = sqrt( epsBwd / muBwd)
      Vmath::Vdiv(nTraceNumPoints, Fwdeps, 1, Fwdmu, 1, m_YimFwd, 1);
      Vmath::Vdiv(nTraceNumPoints, Bwdeps, 1, Bwdmu, 1, m_YimBwd, 1);
      Vmath::Vsqrt(nTraceNumPoints, m_YimFwd, 1, m_YimFwd, 1);
      Vmath::Vsqrt(nTraceNumPoints, m_YimBwd, 1, m_YimBwd, 1);


      NekDouble freqm, freqn, omega;

      freqm = 1.0;
      freqn = 1.0;

      omega = m_PI*sqrt(freqm*freqm+freqn*freqn);

      Array<OneD, NekDouble> Ex(nq);
      Array<OneD, NekDouble> Ey(nq);
      Array<OneD, NekDouble> Hz(nq);

      NekDouble mpi, npi;
      for (int i=0; i<nq; ++i)
        {
          mpi = freqm*m_PI;
          npi = freqn*m_PI;

          Ex[i] = -1.0*(npi/omega)*cos(mpi*x0[i])*sin(npi*x1[i])*sin(omega*time);
          Ey[i] = (mpi/omega)*sin(mpi*x0[i])*cos(npi*x1[i])*sin(omega*time);
          Hz[i] = cos(mpi*x0[i])*cos(npi*x1[i])*cos(omega*time);
        }

      switch(field)
       {
        case(0):
          {
           outfield = Ex;
          }
          break;

        case(1):
          {
           outfield = Ey;
          }
          break;

        case(2):
          {
           outfield = Hz;
          }
          break;
       }
  }


// Defines the boundary conditions
void  LinearMaxwell::BoundaryFwdBwdConditions(Array<OneD, NekDouble> &Fwd,
                                              Array<OneD, NekDouble> &Bwd,
                                              SpatialDomains::BndUserDefinedType BDtype,
                                              BoundaryCopyType BDCopytype)
{
 int id2, npts, cnt = 0;

 // loop over Boundary Regions
 for(int n = 0; n < m_fields[0]->GetBndConditions().num_elements(); ++n)
  {
   for(int e = 0; e < m_fields[0]->GetBndCondExpansions()[n]->GetExpSize(); ++e)
```

```
   {
   npts=m_fields[0]->GetBndCondExpansions()[n]->GetExp(e)->GetNumPoints(0);
   id2=m_fields[0]->GetTrace()->GetPhys_Offset(m_fields[0]->GetTraceMap()->GetBndCondCoeffsToGlobalCoeffsMap(cnt+e));

       if (m_fields[0]->GetBndConditions()[n]->GetUserDefined() == BDtype)
        {
          // Let du/dn = 0 at the boundaries
          switch(BDCopytype)
            {
               case(eFwdEQBwd):
                  {
                   Vmath::Vcopy(npts, &Fwd[id2], 1, &Bwd[id2], 1);
                  }
                  break;

               case(eFwdEQNegBwd):
                  {
                   Vmath::Vcopy(npts, &Fwd[id2], 1, &Bwd[id2], 1);
                   Vmath::Neg(npts, &Bwd[id2], 1);
                  }
                  break;

                  default:
                  break;
            }
        }
      }

       cnt +=m_fields[0]->GetBndCondExpansions()[n]->GetExpSize();
   }
}

// Defines the right hand side of the ODE to be integrated
void LinearMaxwell::DoOdeRhs(const Array<OneD, const Array<OneD, NekDouble> >&inarray,
                             Array<OneD,       Array<OneD, NekDouble> >&outarray,
                             const NekDouble time)
{
    int i;
    int nvariables = inarray.num_elements();
    int ncoeffs    = GetNcoeffs();
    int nq         = GetTotPoints();


    switch(m_projectionType)
      {
       case MultiRegions::eDiscontinuous:
         {
          // m_advection->Advect(nvariables, m_fields, advVel, inarray, outarray);
          Array<OneD, Array<OneD, NekDouble> > modarray(nvariables);
          Array<OneD, Array<OneD, NekDouble> > physarray(nvariables);
          for (i = 0; i < nvariables; ++i)
            {
              modarray[i]  = Array<OneD, NekDouble>(ncoeffs);
              physarray[i] = Array<OneD, NekDouble>(nq);
            }

          // straightforward discontinuous Galerkin
          WeakDGAdvection(inarray, modarray, true, true, nvariables);

          for(i = 0; i < nvariables; ++i)
           {
             m_fields[i]->MultiplyByElmtInvMass(modarray[i],modarray[i]);
             m_fields[i]->BwdTrans(modarray[i],outarray[i]);
           }

          // Multiply \varepsilon^{-1} and \mu^{-1}
```

```
          switch(m_TestType)
          case eMaxwell1D:
              {
                m_fields[0]->ElemtWiseDiv(0, m_Nend, m_n1*m_n1, m_n2*m_n2, outarray[0], outarray[0]);
                m_fields[1]->ElemtWiseDiv(0, m_Nend, 1.0, 1.0, outarray[1], outarray[1]);
              }
           break;
           case eMaxwell2DTM:
              {
               m_fields[0]->ElemtWiseDiv(0, m_Nend, m_mu1, m_mu1, outarray[0], outarray[0]);
               m_fields[1]->ElemtWiseDiv(0, m_Nend, m_mu2, m_mu2, outarray[1], outarray[1]);
               m_fields[2]->ElemtWiseDiv(0, m_Nend, 1.0, 1.0, outarray[2], outarray[2]);
              }
            break;
           case eMaxwell2DTE:
              {
                m_fields[0]->ElemtWiseDiv(0, m_Nend, m_varepsilon1, m_varepsilon1, outarray[0], outarray[0]);
                m_fields[1]->ElemtWiseDiv(0, m_Nend, m_varepsilon2, m_varepsilon2, outarray[1], outarray[1]);
                m_fields[2]->ElemtWiseDiv(0, m_Nend, 1.0, 1.0, outarray[2], outarray[2]);
              }
           break;

           default:
           break;
         }
     }
      break;
      default:
      ASSERTL0(false,"Unknown projection scheme for this case");
      break;
      }
}

// Define the discontinuous Galerkin projection
void LinearMaxwell::DoOdeProjection(const Array<OneD, const Array<OneD, NekDouble> >&inarray,
                                    Array<OneD, Array<OneD, NekDouble> >&outarray,
                                    const NekDouble time)
{
    int i;
    int nvariables = inarray.num_elements();


    switch(m_projectionType)
      {
        case MultiRegions::eDiscontinuous:
         {
           // Just copy over array
           int npoints = GetNpoints();

           for(i = 0; i < nvariables; ++i)
            {
              Vmath::Vcopy(npoints, inarray[i], 1, outarray[i], 1);
            }
           break;
         }
        default:
        ASSERTL0(false,"Unknown projection scheme");
        break;
      }
}


// Define the boundary conditions to be used
void LinearMaxwell::SetBoundaryConditions(Array<OneD, Array<OneD, NekDouble> > &inarray, NekDouble time)
{
```

```
    int nvariables = m_fields.num_elements();
    int cnt = 0;

    // loop over Boundary Regions
    for(int n = 0; n < m_fields[0]->GetBndConditions().num_elements(); ++n)
      {

        // PEC Boundary Condition
        if (m_fields[0]->GetBndConditions()[n]->GetUserDefined() == SpatialDomains::ePEC)
          {
            PECBoundary2D(n,cnt,inarray);
          }

        // Time Dependent Boundary Condition (specified in meshfile)
        if (m_fields[0]->GetBndConditions()[n]->GetUserDefined() == SpatialDomains::eTimeDependent)
          {
           for (int i = 0; i < nvariables; ++i)
              {
                m_fields[i]->EvaluateBoundaryConditions(time);
              }
          }
          cnt +=m_fields[0]->GetBndCondExpansions()[n]->GetExpSize();
      }
}

// implement the perfect electric conductor boundary condition
void LinearMaxwell::PECBoundary2D(int bcRegion, int cnt, Array<OneD, Array<OneD, NekDouble> > &physarray)
{
 int i;
 int nTraceNumPoints = GetTraceTotPoints();
 int nvariables      = physarray.num_elements();

 // get physical values of the forward trace
 Array<OneD, Array<OneD, NekDouble> > Fwd(nvariables);
 for (i = 0; i < nvariables; ++i)
 {
   Fwd[i] = Array<OneD, NekDouble>(nTraceNumPoints);
   m_fields[i]->ExtractTracePhys(physarray[i],Fwd[i]);
 }

 // Adjust the physical values of the trace to take
 // user defined boundaries into account
 int e, id1, id2, npts;

 for(e = 0; e < m_fields[0]->GetBndCondExpansions()[bcRegion]->GetExpSize(); ++e)
 {
  npts=m_fields[0]->GetBndCondExpansions()[bcRegion]->GetExp(e)->GetNumPoints(0);
  id1 =m_fields[0]->GetBndCondExpansions()[bcRegion]->GetPhys_Offset(e) ;
  id2 =m_fields[0]->GetTrace()->GetPhys_Offset(m_fields[0]->GetTraceMap()->GetBndCondCoeffsToGlobalCoeffsMap(cnt+e));

   switch(m_expdim)
   {
     case 1:
      {
        // negate the forward flux
        Vmath::Neg(npts,&Fwd[1][id2],1);
      }
      break;
     case 2:
      {
      }
      break;

      default:
         ASSERTL0(false,"Illegal expansion dimension");
   }
```

```
   Array<OneD, NekDouble> Zeros(npts, 0.0);
   switch(m_TestType)
   {
     case eMaxwell1D:
       {
         Vmath::Vcopy(npts, &Zeros[0], 1, &(m_fields[0]->GetBndCondExpansions()[bcRegion]->UpdatePhys())[id1], 1);
       }
       break;

     case eMaxwell2DTM:
     case eMaxwell2DTE:
       {
         Vmath::Vcopy(npts, &Zeros[0], 1, &(m_fields[2]->GetBndCondExpansions()[bcRegion]->UpdatePhys())[id1], 1);
       }
       break;

       default:
          break;
   }

 }
}
 // Flux vector for 1D test problem
void LinearMaxwell::GetFluxVector1D(const int i,
                                    const Array<OneD, const Array<OneD, NekDouble> > &physfield,
                                    Array<OneD, Array<OneD, NekDouble> > &flux)
{
    int nq = m_fields[0]->GetTotPoints();

    switch(i)
      {
       case 0:
        {
         // H in flux 0
         Vmath::Vcopy(nq, physfield[1], 1, flux[0], 1);

         // E in flux 1
         Vmath::Smul(nq, 0.0, physfield[0], 1, flux[1], 1);
        }
        break;

       case 1:
        {
          // E in flux 0
          Vmath::Vcopy(nq, physfield[0], 1, flux[0], 1);

          // H in flux 1
          Vmath::Smul(nq, 0.0, physfield[1], 1, flux[1], 1);
        }
        break;

        default:
        ASSERTL0(false,"GetFluxVector2D: illegal vector index");
      }
}
 // Flux vector for 2D TM test problem
void LinearMaxwell::GetFluxVector2DTM(const int i,
                                      const Array<OneD, const Array<OneD, NekDouble> > &physfield,
                                      Array<OneD, Array<OneD, NekDouble> > &flux)
{
    int nq = m_fields[0]->GetTotPoints();

    // [Hx, Hy, Ez]
    switch(i)
     {
```

```
       case 0:
         {
          // -Ez in flux 1
          Vmath::Zero(nq, flux[0], 1);
          Vmath::Smul(nq, -1.0, physfield[2], 1, flux[1], 1);
         }
        break;

       case 1:
         {
          // Ez in flux 0
          Vmath::Smul(nq, 1.0, physfield[2], 1, flux[0], 1);
          Vmath::Zero(nq, flux[1], 1);
         }
         break;

       case 2:
         {
          Vmath::Smul(nq, 1.0, physfield[1], 1, flux[0], 1);
          Vmath::Smul(nq, -1.0, physfield[0], 1, flux[1], 1);
         }
         break;

         default:
         ASSERTL0(false,"GetFluxVector2D: illegal vector index");
      }
}
 // Flux vector for 2D TE test problem
void LinearMaxwell::GetFluxVector2DTE(const int i,
                                      const Array<OneD, const Array<OneD, NekDouble> > &physfield,
                                      Array<OneD, Array<OneD, NekDouble> > &flux)
{
    int nq = m_fields[0]->GetTotPoints();

    // [Ex, Ey, Hz]
    switch(i)
      {
      case 0:
        {
          // Hz in flux 1
          Vmath::Zero(nq, flux[0], 1);
          Vmath::Smul(nq, 1.0, physfield[2], 1, flux[1], 1);
        }
        break;

      case 1:
        {
          // -Hz in flux 0
          Vmath::Smul(nq, -1.0, physfield[2], 1, flux[0], 1);
          Vmath::Zero(nq, flux[1], 1);
        }
        break;

      case 2:
        {
          Vmath::Smul(nq, -1.0, physfield[1], 1, flux[0], 1);
          Vmath::Smul(nq, 1.0, physfield[0], 1, flux[1], 1);
        }
        break;

        default:
        ASSERTL0(false,"GetFluxVector2D: illegal vector index");
      }
}
//Numerical flux for 1D test problem
void LinearMaxwell::NumericalFlux1D(Array<OneD, Array<OneD, NekDouble> > &physfield,
```

```
                                    Array<OneD, Array<OneD, NekDouble> > &numflux)
{
  switch(m_upwindType)
     {
       case eAverage:
         {
           AverageNumericalFlux1D(physfield, numflux);
         }
        break;

       case eUpwind:
         {
          UpwindNumericalFlux1D(physfield, numflux);
         }
         break;

       default:
         {
           ASSERTL0(false,"populate switch statement for upwind flux");
         }
       break;
     }
}
//upwind flux for 1D
void LinearMaxwell::UpwindNumericalFlux1D(Array<OneD, Array<OneD, NekDouble> > &physfield,
                                          Array<OneD, Array<OneD, NekDouble> > &numflux)
{
    int i;
    int nTraceNumPoints = GetTraceTotPoints();
    int nvariables      = 2;

    // get temporary arrays
    Array<OneD, Array<OneD, NekDouble> > Fwd(nvariables);
    Array<OneD, Array<OneD, NekDouble> > Bwd(nvariables);

    for (i = 0; i < nvariables; ++i)
      {
        Fwd[i] = Array<OneD, NekDouble>(nTraceNumPoints);
        Bwd[i] = Array<OneD, NekDouble>(nTraceNumPoints);
      }

    // get the physical values at the trace from the dependent variables
    for (i = 0; i < nvariables; ++i)
      {
        m_fields[i]->GetFwdBwdTracePhys(physfield[i],Fwd[i],Bwd[i]);
      }


    // E = 0 at the boundaries
    BoundaryFwdBwdConditions(Fwd[0], Bwd[0], SpatialDomains::ePEC, eFwdEQNegBwd);

    // d H / d n = 0 at the boundaries
    BoundaryFwdBwdConditions(Fwd[1], Bwd[1], SpatialDomains::ePEC, eFwdEQBwd);

    Array<OneD, NekDouble> dE(nTraceNumPoints);
    Array<OneD, NekDouble> dH(nTraceNumPoints);

    Vmath::Vsub(nTraceNumPoints, &Fwd[0][0], 1, &Bwd[0][0], 1, &dE[0], 1);
    Vmath::Vsub(nTraceNumPoints, &Fwd[1][0], 1, &Bwd[1][0], 1, &dH[0], 1);

    NekDouble nx, AverZ, AverY, AverZH, AverYE;
    for (i = 0; i < nTraceNumPoints; ++i)
      {
        nx = m_traceNormals[0][i];
        AverZ = m_ZimFwd[i]+m_ZimBwd[i];
        AverY = m_YimFwd[i]+m_YimBwd[i];
```

```
        AverZH = m_ZimFwd[i]*Fwd[1][i]+m_ZimBwd[i]*Bwd[1][i];
        AverYE = m_YimFwd[i]*Fwd[0][i]+m_YimBwd[i]*Bwd[0][i];

        numflux[0][i] = nx/AverZ*(AverZH - nx*dE[i]);
        numflux[1][i] = nx/AverY*(AverYE - nx*dH[i]);
      }
}
//centered flux for 1D
void LinearMaxwell::AverageNumericalFlux1D(Array<OneD, Array<OneD, NekDouble> > &physfield,
                                           Array<OneD, Array<OneD, NekDouble> > &numflux)
{
    int i;
    int nTraceNumPoints = GetTraceTotPoints();
    int nvariables      = 2;

    // get temporary arrays
    Array<OneD, Array<OneD, NekDouble> > Fwd(nvariables);
    Array<OneD, Array<OneD, NekDouble> > Bwd(nvariables);

    for (i = 0; i < nvariables; ++i)
      {
        Fwd[i] = Array<OneD, NekDouble>(nTraceNumPoints);
        Bwd[i] = Array<OneD, NekDouble>(nTraceNumPoints);
      }

    // get the physical values at the trace from the dependent variables
    for (i = 0; i < nvariables; ++i)
      {
        m_fields[i]->GetFwdBwdTracePhys(physfield[i],Fwd[i],Bwd[i]);
      }

    // E = 0 at the boundaries
    BoundaryFwdBwdConditions(Fwd[0], Bwd[0], SpatialDomains::ePEC, eFwdEQNegBwd);

    // d H / d n = 0 at the boundaries
    BoundaryFwdBwdConditions(Fwd[1], Bwd[1], SpatialDomains::ePEC, eFwdEQBwd);

    for (i = 0; i < nTraceNumPoints; ++i)
      {
        numflux[0][i]  = 0.5*m_traceNormals[0][i]*(Fwd[1][i] + Bwd[1][i]);
        numflux[1][i]  = 0.5*m_traceNormals[0][i]*(Fwd[0][i] + Bwd[0][i]);
      }
}

//numerical flux for 2D TM
void LinearMaxwell::NumericalFlux2DTM(Array<OneD, Array<OneD, NekDouble> > &physfield,
                                      Array<OneD, Array<OneD, NekDouble> > &numflux)
{
    int i;
    int nTraceNumPoints = GetTraceTotPoints();
    int nvar = m_fields.num_elements();

    // get temporary arrays
    Array<OneD, Array<OneD, NekDouble> > Fwd(nvar);
    Array<OneD, Array<OneD, NekDouble> > Bwd(nvar);

    for (i = 0; i < nvar; ++i)
      {
        Fwd[i] = Array<OneD, NekDouble>(nTraceNumPoints);
        Bwd[i] = Array<OneD, NekDouble>(nTraceNumPoints);
      }

    // get the physical values at the trace from the dependent variables
    for (i = 0; i < nvar; ++i)
      {
        m_fields[i]->GetFwdBwdTracePhys(physfield[i],Fwd[i],Bwd[i]);
```

```
      }

    // E = 0 at the boundaries
    BoundaryFwdBwdConditions(Fwd[0], Bwd[0], SpatialDomains::ePEC, eFwdEQBwd);
    BoundaryFwdBwdConditions(Fwd[1], Bwd[1], SpatialDomains::ePEC, eFwdEQBwd);
    BoundaryFwdBwdConditions(Fwd[2], Bwd[2], SpatialDomains::ePEC, eFwdEQNegBwd);

    Array<OneD, NekDouble> dHx(nTraceNumPoints);
    Array<OneD, NekDouble> dHy(nTraceNumPoints);
    Array<OneD, NekDouble> dEz(nTraceNumPoints);

    Vmath::Vsub(nTraceNumPoints, &Fwd[0][0], 1, &Bwd[0][0], 1, &dHx[0], 1);
    Vmath::Vsub(nTraceNumPoints, &Fwd[1][0], 1, &Bwd[1][0], 1, &dHy[0], 1);
    Vmath::Vsub(nTraceNumPoints, &Fwd[2][0], 1, &Bwd[2][0], 1, &dEz[0], 1);

    NekDouble nx, ny, AverZ, AverY, AverZHx, AverZHy, AverYEz, ncrossdH;
    for (i = 0; i < nTraceNumPoints; ++i)
      {
        nx = m_traceNormals[0][i];
        ny = m_traceNormals[1][i];

        AverZ = 0.5*(m_ZimFwd[i]+m_ZimBwd[i]);
        AverY = 0.5*(m_YimFwd[i]+m_YimBwd[i]);
        AverZHx = 0.5*(m_ZimFwd[i]*Fwd[0][i]+m_ZimBwd[i]*Bwd[0][i]);
        AverZHy = 0.5*(m_ZimFwd[i]*Fwd[1][i]+m_ZimBwd[i]*Bwd[1][i]);
        AverYEz = 0.5*(m_YimFwd[i]*Fwd[2][i]+m_YimBwd[i]*Bwd[2][i]);

        ncrossdH = nx*dHy[i] - ny*dHx[i];

        numflux[0][i]  = (ny/AverY)*( -1.0*AverYEz + 0.5*m_alpha*ncrossdH );
        numflux[1][i]  = (nx/AverY)*( AverYEz - 0.5*m_alpha*ncrossdH );
        numflux[2][i]  = (1.0/AverZ)*( nx*AverZHy - ny*AverZHx - 0.5*m_alpha*dEz[i] );
      }
}
//numerical flux for 2D TE
void LinearMaxwell::NumericalFlux2DTE(Array<OneD, Array<OneD, NekDouble> > &physfield,
                                      Array<OneD, Array<OneD, NekDouble> > &numflux)
{
    int i;
    int nTraceNumPoints = GetTraceTotPoints();
    int nvar = m_fields.num_elements();

    // get temporary arrays
    Array<OneD, Array<OneD, NekDouble> > Fwd(nvar);
    Array<OneD, Array<OneD, NekDouble> > Bwd(nvar);

    for (i = 0; i < nvar; ++i)
      {
       Fwd[i] = Array<OneD, NekDouble>(nTraceNumPoints);
       Bwd[i] = Array<OneD, NekDouble>(nTraceNumPoints);
      }

    // get the physical values at the trace from the dependent variables
    for (i = 0; i < nvar; ++i)
      {
        m_fields[i]->GetFwdBwdTracePhys(physfield[i],Fwd[i],Bwd[i]);
      }

    // E = 0 at the boundaries
    BoundaryFwdBwdConditions(Fwd[0], Bwd[0], SpatialDomains::ePEC, eFwdEQNegBwd);
    BoundaryFwdBwdConditions(Fwd[1], Bwd[1], SpatialDomains::ePEC, eFwdEQNegBwd);
    BoundaryFwdBwdConditions(Fwd[2], Bwd[2], SpatialDomains::ePEC, eFwdEQBwd);

    Array<OneD, NekDouble> dEx(nTraceNumPoints);
    Array<OneD, NekDouble> dEy(nTraceNumPoints);
    Array<OneD, NekDouble> dHz(nTraceNumPoints);
```

```
    Vmath::Vsub(nTraceNumPoints, &Fwd[0][0], 1, &Bwd[0][0], 1, &dEx[0], 1);
    Vmath::Vsub(nTraceNumPoints, &Fwd[1][0], 1, &Bwd[1][0], 1, &dEy[0], 1);
    Vmath::Vsub(nTraceNumPoints, &Fwd[2][0], 1, &Bwd[2][0], 1, &dHz[0], 1);

    NekDouble nx, ny, AverZ, AverY, AverYEx, AverYEy, AverZHz, ncrossdE;
    for (i = 0; i < nTraceNumPoints; ++i)
      {
        nx = m_traceNormals[0][i];
        ny = m_traceNormals[1][i];

        AverZ = 0.5*(m_ZimFwd[i]+m_ZimBwd[i]);
        AverY = 0.5*(m_YimFwd[i]+m_YimBwd[i]);
        AverYEx = 0.5*(m_YimFwd[i]*Fwd[0][i]+m_YimBwd[i]*Bwd[0][i]);
        AverYEy = 0.5*(m_YimFwd[i]*Fwd[1][i]+m_YimBwd[i]*Bwd[1][i]);
        AverZHz = 0.5*(m_ZimFwd[i]*Fwd[2][i]+m_ZimBwd[i]*Bwd[2][i]);

        ncrossdE = nx*dEy[i] - ny*dEx[i];

        numflux[0][i]  = (ny/AverZ)*( 1.0*AverZHz + 0.5*m_alpha*ncrossdE );
        numflux[1][i]  = (nx/AverZ)*( -AverZHz - 0.5*m_alpha*ncrossdE );
        numflux[2][i]  = (1.0/AverY)*( -nx*AverYEy + ny*AverYEx - 0.5*m_alpha*dHz[i] );
      }
}
 //Evaluates L^2 error
NekDouble LinearMaxwell::v_L2Error(unsigned int field,
                                   const    Array<OneD, NekDouble> &exactsoln,
                                   bool     Normalised)
{
   int nq = m_fields[field]->GetNpoints();
   NekDouble L2error = -1.0;

   if(m_fields[field]->GetPhysState() == false)
     {
        m_fields[field]->BwdTrans(m_fields[field]->GetCoeffs(),
        m_fields[field]->UpdatePhys());
     }

    Array<OneD, NekDouble> exactsolution(nq);
    EvaluateExactSolution(field, exactsolution, m_time);
    L2error = m_fields[field]->L2(m_fields[field]->GetPhys(), exactsolution);

    return L2error;
}
  //Evaluates L\inf error
NekDouble LinearMaxwell::v_LinfError(unsigned int field,
                                     const  Array<OneD, NekDouble> &exactsoln)
{
   int nq = m_fields[field]->GetNpoints();
   NekDouble LinfError = -1.0;

   if(m_fields[field]->GetPhysState() == false)
     {
       m_fields[field]->BwdTrans(m_fields[field]->GetCoeffs(),
       m_fields[field]->UpdatePhys());
     }

    Array<OneD, NekDouble> exactsolution(nq);
    EvaluateExactSolution(field, exactsolution, m_time);
    LinfError = m_fields[field]->Linf(m_fields[field]->GetPhys(), exactsolution);

    return LinfError;
}

void LinearMaxwell::v_GenerateSummary(SolverUtils::SummaryList& s)
{
```

```
    MaxwellSystem::v_GenerateSummary(s);

    cout << "\tn1              : " << m_n1 << endl;
    cout << "\tn2              : " << m_n2 << endl;
    cout << "\tNend            : " << m_Nend << endl;
}

} //end of namespace
```

List of References

[1] Yee, K.: Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media. *IEEE Trans. Antennas Propag*, vol. 14, no. 3, pp. 302–307, 1966.

[2] Kopriva, D.A.: *Implementing Spectral Methods for Partial Differential Equations: Algorithms for Scientists and Engineers*. Springer, 2009.

[3] Trefethen, L.N.: *Spectral Methods in MATLAB*, vol. 10. Siam, 2000.

[4] Gottlieb, D., Orszag, S.A. and MA, C.H.I.: *Numerical Analysis of Spectral Methods: Theory and Applications*. SIAM, 1977.

[5] Lonngren, K.E. and Savov, S.V.: *Fundamentals of electromagnetics with MATLAB*. SciTech publishing, Taiwan, 2007.

[6] Hesthaven, J.S. and Warburton, T.: *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*, vol. 54. Springer, 2007.

[7] Garg, R.: *Analytical and Computational Methods in Electromagnetics*. Artech house, 2008.

[8] Taflove, A. and Hagness, S.C.: Computational Electrodynamics: The Finite-Difference Time Domain Method. 2000.

[9] Karniadakis, G. and Sherwin, S.: *Spectral/hp Element Methods for Computational Fluid Dynamics*. Oxford University Press, 2005.

[10] Toro, E.: *Riemann Solvers and Numerical Methods for Fluids Dynamics*. Springer, 2009.

[11] Bondeson, A., Rylander, T. and Ingelström, P.: *Computational Electromagnetics*, vol. 51. Springer, 2005.

[12] Sengupta, D.L. and Liepa, V.V.: *Applied electromagnetics and electromagnetic compatibility*, vol. 136. John Wiley & Sons, 2005.

[13] Ditkowski, A., Dridi, K. and Hesthaven, J.S.: Convergent Cartesian Grid Methods for Maxwell's Equations in Complex Geometries. *Journal of Computational Physics*, vol. 170, no. 1, pp. 39–80, 2001.

[14] Hesthaven, J. and Warburton, T.: Discontinuous Galerkin Methods for the Time-Domain Maxwell's Equations. *Applied Computational Electromagnetics Society Newsletter*, vol. 19, no. 1, p. 4, 2004.

[15] Hesthaven, S. and Warburton, T.: High-Order Accurate Methods for Time-domain Electromagnetics. *Computer Modeling in Engineering and Sciences*, vol. 5, no. 5, pp. 395–408, 2000.

[16] Hesthaven, J.S., Warburton, T., Chauviere, C. and Wilcox, L.: *High-order discontinuous Galerkin methods for computational electromagnetics and uncertainty quantification.* Springer, 2010.

[17] Rao, N.N.: *Fundamentals of electromagnetics for electrical and computer engineering.* Pearson Prentice Hall, 2009.

[18] Monk, P.: *Finite element methods for Maxwell's equations.* Clarendon Press Oxford, 2003.

[19] Barth, T.J. and Deconinck, H.: *High-order methods for computational physics*, vol. 9. Springer Science & Business Media, 1999.

[20] Mohammadian, A.H., Shankar, V. and Hall, W.F.: Computation of electromagnetic scattering and radiation using a time-domain finite-volume discretization procedure. *Computer Physics Communications*, vol. 68, no. 1, pp. 175–196, 1991.

[21] Fahs, H.: Numerical evaluation of a non-conforming discontinuous Galerkin method on triangular meshes for solving the time-domain Maxwell equations. *Unité de recherche INRIA*, 2007.

[22] König, M., Busch, K. and Niegemann, J.: The Discontinuous Galerkin Time-Domain Method for Maxwell's equations with anisotropic materials. *Photonics and Nanostructures-Fundamentals and Applications*, vol. 8, no. 4, pp. 303–309, 2010.

[23] Chun, S. and Hesthaven, J.S.: Modeling of the Frozen Mode Phenomenon and its Sensitivity Using Discontinuous Galerkin Methods. *Communications in Computational Physics*, vol. 2, no. 611-639, 2007.

[24] JURGENS, T., Taflove, A., Umashankar, K. and MOORE, T.: Finite-Difference Time-Domain Modeling of Curved Surfaces. *IEEE Transactions on Antennas and Propagation*, vol. 40, no. 4, 1992.

[25] Taflove, A. and Brodwin, M.E.: Numerical solution of steady-state electromagnetic scattering problems using the time-dependent maxwell's equations. *Microwave Theory and Techniques, IEEE Transactions*, vol. 23, no. 8, pp. 623–630, 1975.

[26] Taflove, A. and Brodwin, M.E.: Computation of the electromagnetic fields and induced temperatures within a model of the microwave-irradiated human eye. *Microwave Theory and Techniques, IEEE Transactions*, vol. 23, no. 11, pp. 888–896, 1975.

[27] Malika Remaki, L.F.: Une méthode de Galerkin Discontinu pour la résolution des équations de Maxwell en milieu hétérogène. *Unité de recherche INRIA*, 1998.

[28] Min, M.: Discontinuous Galerkin Method Based on Quadrilateral Mesh for Maxwell's Equations. *Wireless Communications and Applied Computational Electromagnetics, 2005. IEEE/ACES International Conference*, pp. 719–723, 2005.

[29] Vos, P.E., Eskilsson, C., Bolis, A., Chun, S., Kirby, R.M. and Sherwin, S.J.: A generic framework for time-stepping partial differential equations (PDEs): general linear methods, object-oriented implementation and application to fluid problems. *International Journal of Computational Fluid Dynamics*, vol. 25, no. 3, pp. 107–125, 2011.

[30] Blank, E.: *The Discontinuous Galerkin method for Maxwell's equations*. Ph.D. thesis, Karlsruhe institute of technology, 2013.

[31] Jesus Alvarez, G.: *A discontinuous Galerkin Finite element method for the Time-Domain Solution of Maxwell Equations*. Ph.D. thesis, Granada University, 2012.

[32] Vos, P.E.J.: *From h to p efficiently: Optimizing the implementation of spectral/hp element methods*. Ph.D. thesis, University of London, 2011.

[33] Faraniaina, R.: *A comparative study on the impact of different numerical fluxes in a discontinuous Galerkin scheme for the 2D shallow water equations*. Master's thesis, Stellenbosch University, South Africa, 2013.

[34] Nektar++. http://www.nektar.info/.

[35] FDTD. http://en.wikipedia.org/wiki/Finite-difference_time-domain_method.

[36] Spectral Element Method. http://en.wikipedia.org/wiki/Spectral_element_method.