Multiphysics integrated computational project

DISCONTINUOUS GALERKIN METHOD FOR MAXWELL'S EQUATIONS

Louis Bonhomme, Quentin Elias, Chloé Halbach

under the supervision of

Christophe Geuzaine and Romain Boman

academic year 2019 - 2020

master degree in engineering physics - university of liège

Contents

List of Figures

List of Tables

1 Introduction

Multiphysics computing plays a central role in scientific progress and breakthroughs, offering a bridge between theoretical and experimental studies. It is a cost-efficient and adaptive way to check the validity of a theory or to better understand experimental phenomena. One important goal of computational modeling is to develop a general, accurate, and flexible method for a specific class of problems, so that problems from different fields (solid mechanics, fluid mechanics, electromagnetism, acoustics, ...) can be solved with the same code, provided that governing equations and boundary conditions are adapted.

Partial differential equations (PDE) can be divided into elliptic, parabolic and hyperbolic types, depending on their mathematical properties. These classes determine for example which type of physical phenomena can be modeled, or which numerical method should be chosen to approximate the solution [1]. In particular, this project consists in studying a hyperbolic system of (un)coupled equations, that will be applied to an advection problem and to the Maxwell equations. On the one hand, spatial discretization is performed using the discontinuous Galerkin (DG) method. Its advantages for solving PDEs are the following: geometric flexibility, high-order accuracy, hp-adaptivity with respect to mesh size and order of approximation, and local formulation [2]. Hence, the DG method allows to treat problems with complex geometries and discontinuities in the solution. Moreover, the local formulation simplifies parallel computing on graphics processing units (GPU). On the other hand, an explicit time-marching method has been chosen to discretize time since it is easier to implement. The state of the system at the next time step is directly obtained from the state at the current time step.

The report is structured as follows. At first, we summarize how to identify hyperbolic equations and how to obtain a DG-scheme from the PDE. A considerable part is dedicated to the numerical implementation of the DG-scheme. We explain how to compute the mass matrix and stiffness matrix, how to deal with fluxes to reconstruct the solution at the element interfaces, how to implement the time-marching method, and how to take boundary conditions into account. In the first instance, the DG-scheme is developed, tested and validated for the solution of three uncoupled scalar transport equations. The convergence and stability of the scheme are studied, along with the speedup when the code is parallelized on CPU with OpenMP, accelerated on a multicore CPU, and accelerated on a GPU. Then, the code is applied to Maxwell's equations. Several problems are simulated: a 2D resonant cavity, a parallel plate waveguide, a punctual oscillating charge in a unit-square geometry, a periodic grating constituted of parallel plate waveguides, an S-shaped waveguide, and a similar S-shaped waveguide cut into a photonic crystal. The convergence, stability, and speedup studies are carried out on the resonant cavity problem.

2 Hyperbolic partial differential equations

Since this form of equations is common in physics, let us consider a second order partial differential equation for the scalar solution u depending on n independent variables $\boldsymbol{x}$:

$$\sum_{i,j=1}^{n} a_{ij}(\boldsymbol{x})\frac{\partial^2 u(\boldsymbol{x})}{\partial x_i \partial x_j} + \sum_{i,j=1}^{n} b_i(\boldsymbol{x})\frac{\partial u(\boldsymbol{x})}{\partial x_i} + c(\boldsymbol{x})u(\boldsymbol{x}) + g(\boldsymbol{x}) = 0, \tag{1}$$

with the given coefficient functions a_{ij}, b_i, c, and g. The criterion to classify partial differential equations into parabolic, elliptic or hyperbolic types depends on the coefficients of the second order terms [3]. More precisely, the classification is based on the eigenvalues of the coefficient matrix a_{ij}:

- if any eigenvalue is zero, the PDE is said parabolic (e.g. heat equation);
- if all eigenvalues are non-zero and have the same sign, the PDE is said elliptic (e.g. Laplace equation);
- if all eigenvalues are non-zero and all but one have the same sign, the PDE is said hyperbolic (e.g. wave equation).

However, parabolic, elliptic and hyperbolic types can also refer to first order and higher order equations. One example of a first order hyperbolic equations is the advection equation:

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0. \tag{2}$$

Essential properties which distinguish hyperbolic equations from elliptic and parabolic equations are the following: hyperbolic equations entail a finite propagation speed, a finite domain of dependence, and a finite range of influence [1]. Data is transported along the characteristic lines of the equation.

3 Discontinuous Galerkin method

To introduce the discontinuous Galerkin method [2], let us consider the generalized two-dimensional advection equation for the scalar solution $u(\boldsymbol{x}, t)$:

$$\frac{\partial u}{\partial t} + \boldsymbol{\nabla} \cdot \boldsymbol{f} = g, \quad \boldsymbol{x} \in \Omega \tag{3}$$

where $\boldsymbol{f}(u)$ is the flux function, $g(\boldsymbol{x}, t)$ is a source function, and Ω is the domain delimited by a boundary $\partial\Omega$.

At first, a mesh must be generated by subdividing the domain Ω into a union of non-overlapping elements $\mathcal{D}^e$:

$$\Omega \simeq \Omega_h = \bigcup_{e=1}^{N_e} \mathcal{D}^e, \tag{4}$$

with N_e the number of mesh elements. Then, we must choose how to represent the exact solution $u(\boldsymbol{x}, t)$ by an approximate solution $u_h(\boldsymbol{x}, t)$. A common practice is to express the approximate solution as a linear combination of user-defined local polynomials $l_j^e(\boldsymbol{x})$, such that:

$$u(\boldsymbol{x}, t) \simeq u_h(\boldsymbol{x}, t) = \bigoplus_{e=1}^{N_e} u_h^e(\boldsymbol{x}, t) \quad \text{with} \quad u_h^e(\boldsymbol{x}, t) = \sum_{j=1}^{N_p} u_h^e\left(\boldsymbol{x_j^e}, t\right) l_j^e(\boldsymbol{x}), \tag{5}$$

whit N_p the number of nodes per element and $u_h^e\left(\boldsymbol{x}_j^e, t\right)$ the unknown nodal values. The flux and source functions are approximated as:

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x}, t) \simeq \boldsymbol{f}_h(\boldsymbol{x}, t) = \bigoplus_{e=1}^{N_e} \boldsymbol{f_h^e}(\boldsymbol{x}, t) \quad \text{with} \quad \boldsymbol{f_h^e}(\boldsymbol{x}, t) = \sum_{j=1}^{N_p} \boldsymbol{f_h^e}\left(\boldsymbol{x_j^e}, t\right) l_j^e(\boldsymbol{x}), \\ g(\boldsymbol{x}, t) \simeq g_h(\boldsymbol{x}, t) = \bigoplus_{e=1}^{N_e} g_h^e(\boldsymbol{x}, t) \quad \text{with} \quad g_h^e(\boldsymbol{x}, t) = \sum_{j=1}^{N_p} g_h^e\left(\boldsymbol{x_j^e}, t\right) l_j^e(\boldsymbol{x}). \end{aligned} \tag{6}$$

To ensure that the approximate solution satisfies (3), we define the local residual

$$\boldsymbol{x} \in \mathcal{D}^e : \mathcal{R}_h^e(\boldsymbol{x}, t) = \partial_t u_h^e + \boldsymbol{\nabla} \cdot \boldsymbol{f_h^e} - g_h^e, \tag{7}$$

and require this to vanish on each element in a Galerkin sense:

$$\int_{\mathcal{D}^e} \mathcal{R}_h^e(\boldsymbol{x}, t) l_i^e(\boldsymbol{x}) dS_{\boldsymbol{x}} = 0, \quad i = 1, \ldots, N_p, \quad e = 1, \ldots, N_e. \tag{8}$$

Nevertheless, up to now, all elements are disconnected due to the local statement of the residual. Therefore, to connect elements, we apply Gauss's theorem:

$$\begin{aligned} &\int_{\mathcal{D}^e} \left[\partial_t u_h^e l_i^e + \boldsymbol{\nabla} \cdot \boldsymbol{f_h^e}\, l_i^e - g_h^e l_i^e\right] dS_{\boldsymbol{x}} = 0 \\ \Leftrightarrow &\int_{\mathcal{D}^e} \left[\partial_t u_h^e l_i^e - \boldsymbol{f_h^e} \cdot \boldsymbol{\nabla} l_i^e - g_h^e l_i^e\right] dS_{\boldsymbol{x}} = -\oint_{\partial \mathcal{D}^e} \boldsymbol{f_h^e} \cdot \boldsymbol{n^e}\, l_i^e\, ds_{\boldsymbol{x}}, \end{aligned} \tag{9}$$

where $\partial \mathcal{D}^k$ denotes the boundary of the element and $\boldsymbol{n^e}$ its unit outward normal. The problem we are facing now is that the solution at the interfaces between adjacent elements is not necessarily unique. We solve this by introducing a numerical flux $\boldsymbol{f^*}$ which combines information from both elements[1], so that a weak form of the advection equation becomes:

$$\int_{\mathcal{D}^e} \left[\partial_t u_h^e l_i^e - \boldsymbol{f_h^e} \cdot \boldsymbol{\nabla} l_i^e - g_h^e l_i^e\right] dS_{\boldsymbol{x}} = -\oint_{\partial \mathcal{D}^e} \boldsymbol{f_h^{*e}} \cdot \boldsymbol{n^e}\, l_i^e\, ds_{\boldsymbol{x}}. \tag{10}$$

Finally, by injecting (5) and (6) in (10), we arrive at the following DG-scheme:

$$\mathcal{M}_{ij}^e \frac{du_j^e}{dt} - \boldsymbol{\mathcal{S}_{ji}^e} \cdot \boldsymbol{f_i^e} - \mathcal{M}_{ij}^e g_j^e = \mathcal{F}_i^e \Leftrightarrow \frac{du_j^e}{dt} = \left(\mathcal{M}_{ij}^e\right)^{-1} \left(\boldsymbol{\mathcal{S}_{ji}^e} \cdot \boldsymbol{f_k^e} + \mathcal{M}_{ij}^e g_j^e + \mathcal{F}_i^e\right), \tag{11}$$

[1]Because the advection equation is conservative, the numerical flux leaving an element must enter the neighbor element.

while defining the following quantities:

$$\text{mass matrix } \mathcal{M}_{ij}^e = \int_{\mathcal{D}^e} l_i^e(\boldsymbol{x}) l_j^e(\boldsymbol{x})\, dS_{\boldsymbol{x}}, \tag{12}$$

$$\text{stiffness matrix } \boldsymbol{\mathcal{S}_{ij}^e} = \int_{\mathcal{D}^e} l_i^e(\boldsymbol{x}) \boldsymbol{\nabla} l_j^e(\boldsymbol{x})\, dS_{\boldsymbol{x}}, \tag{13}$$

$$\text{flux vector } \mathcal{F}_i^e = -\oint_{\partial\mathcal{D}^e} \boldsymbol{f_h^{*e}} \cdot \boldsymbol{n^e} l_i^e(\boldsymbol{x})\, ds_{\boldsymbol{x}}, \tag{14}$$

$$\text{nodal values } u_j^e = u_h^e\left(\boldsymbol{x_j^e}, t\right), \tag{15}$$

$$\text{flux function } \boldsymbol{f_j^e} = \boldsymbol{f_h^e}\left[u_h^e\left(\boldsymbol{x_j^e}, t\right)\right], \tag{16}$$

$$\text{source function } g_j^e = g_h^e\left(\boldsymbol{x_j^e}, t\right). \tag{17}$$

4 Numerical implementation of the DG-scheme

Instead of determining $\mathcal{M}_{ij}^e$, $\boldsymbol{\mathcal{S}_{ij}^e}$, and $\mathcal{F}_i^e$ in the real space $\mathcal{D}^e$ as defined above, we use a change of variable to compute it in the reference/parent space $\hat{\mathcal{D}}^e$. The advantage of this method is that, by contrast to the real space, the basis functions in the reference space do not depend on the coordinates of the element nodes. Consequently, the basis functions are stored only once per type of element at the beginning of the code, and are then called when needed. Moreover, the use of isoparametric elements renders possible the accurate description of curved geometries.

The change of variable is given by

$$d\mathcal{D}^e = J^e d\hat{\mathcal{D}}^e \quad \text{with} \quad J^e = \det\left(\mathcal{J}^e\right) \quad \text{and} \quad \mathcal{J}^e = \begin{pmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{pmatrix}, \tag{18}$$

the Jacobian of the transformation and the Jacobian matrix respectively.

4.1 Mass matrix

Using the change of variable between the real space and the reference space, the mass matrix associated to element e writes

$$\mathcal{M}_{ij}^e = \int_{\hat{\mathcal{D}}^e} l_i^e(\xi,\eta) l_j^e(\xi,\eta)\, J^e(\xi,\eta)\, d\xi\, d\eta, \tag{19}$$

with (ξ,η) the coordinates in the reference space.

An exact computation of the mass matrix is obtained through Gauss integration. Indeed, if Lagrange polynomials are chosen as basis functions, polynomials of order $2N-1$ can be integrated exactly if N optimized sampling points are used [4]. These sampling points are called Gauss/integration points and their coordinates are denoted by (ξ_g, η_g). Numerically, the mass matrix is thus computed as follows for each mesh element e:

$$\mathcal{M}^e_{ij} = \sum_{g=1}^{N_g} l^e_i\left(\xi_g,\eta_g\right) l^e_j\left(\xi_g,\eta_g\right) w^e\left(\xi_g,\eta_g\right) J^e\left(\xi_g,\eta_g\right), \tag{20}$$

where N_g is the number of integration points, $w^e\left(\xi_g,\eta_g\right)$ is the weight associated to an integration point, $J^e\left(\xi_g,\eta_g\right)$ is the Jacobian evaluated at an integration point, and $l^e_i\left(\xi_g,\eta_g\right)$ is a Lagrange polynomial associated to node i and evaluated at an integration point.

4.2 Stiffness matrix

The method to compute the stiffness matrix is similar to what has been done with the mass matrix. However, the transformation from general coordinates in the real space to parametric coordinates in the reference space involves the inverse Jacobian matrix when dealing with derivatives. We have

$$\frac{\partial l^e_j}{\partial x_n} = \frac{\partial l^e_j}{\partial \xi_m}\frac{\partial \xi_m}{\partial x_n} = (\mathcal{J}^e)^{-1}_{nm}\frac{\partial l^e_j}{\partial \xi_m} \quad \text{with} \quad (\mathcal{J}^e)^{-1} = \begin{pmatrix} \dfrac{\partial \xi}{\partial x} & \dfrac{\partial \eta}{\partial x} \\ \dfrac{\partial \xi}{\partial y} & \dfrac{\partial \eta}{\partial y} \end{pmatrix} \tag{21}$$

the inverse Jacobian matrix, leading to the generalized form of the stiffness matrix

$$\mathcal{S}^e_{ijn} = \int_{\hat{\mathcal{D}}^e} l^e_i(\xi,\eta)(\mathcal{J}^e(\xi,\eta))^{-1}_{nm}\frac{\partial l^e_j}{\partial \xi_m}(\xi,\eta) J^e(\xi,\eta)\, d\xi\, d\eta, \tag{22}$$

with $n = x, y$ the components in the reference space. Using Gauss integration, the stiffness matrix is thereby computed as follows for each mesh element e and component $n = x, y$:

$$\mathcal{S}^e_{ijn} = \sum_{g=1}^{N_g} l^e_i\left(\xi_g,\eta_g\right)(\mathcal{J}^e(\xi_g,\eta_g))^{-1}_{nm}\frac{\partial l^e_j}{\partial \xi_m}(\xi_g,\eta_g) w^e\left(\xi_g,\eta_g\right) J^e\left(\xi_g,\eta_g\right). \tag{23}$$

4.3 Vector of fluxes

Exploiting the change of variable between the real space and the reference space and Gauss integration, the flux vector becomes

$$\mathcal{F}^e_i = -\oint_{\partial\hat{\mathcal{D}}^e} \boldsymbol{f}^{*\boldsymbol{e}}_{\boldsymbol{h}} \cdot \boldsymbol{n}^{\boldsymbol{e}}\, l^e_i(\xi,\eta) J^e(\xi,\eta)\, ds_{\boldsymbol{\xi}} \tag{24}$$

$$= -\sum_{b=1}^{N_b}\sum_{g=1}^{N_g} \boldsymbol{f}^{*\boldsymbol{e},\boldsymbol{b}}_{\boldsymbol{g}} \cdot \boldsymbol{n}^{\boldsymbol{e},\boldsymbol{b}}_{\boldsymbol{g}}\, l^{e,b}_i(\xi_g,\eta_g) w^{e,b}(\xi_g,\eta_g) J^{e,b}(\xi_g,\eta_g), \tag{25}$$

with N_b the number of edges and N_g the number of Gauss points for each edge element. It should be noticed that the Gauss points, basis functions, weights and Jacobians are the ones associated to the edges of the elements. They are thus different from the ones used to compute the mass and stiffness matrices.

For simplicity, we assume that there exists only one normal to each element edge. However, this is only true if the elements are straight. Consequently, for curved elements, we will have to define a normal at each integration point. In the case of a straight 1D element making up an edge labeled b, going from node i to node j, the normalized normal is defined as follows:

$$\boldsymbol{n}^b = \frac{(\boldsymbol{r}_j - \boldsymbol{r}_i) \times \boldsymbol{e}_z}{\|(\boldsymbol{r}_j - \boldsymbol{r}_i) \times \boldsymbol{e}_z\|}, \tag{26}$$

where $\boldsymbol{r}_j$ and $\boldsymbol{r}_i$ are respectively the position vectors of nodes j and i, in cartesian coordinates, and $\boldsymbol{e}_z$ is the unit out-of-plane basis vector. Considering that we are working in two dimensions, the third components of the position vectors are zero and the normal becomes:

$$\boldsymbol{n}^b = \frac{[(r_{j,1} - r_{i,1})\boldsymbol{e}_x + (r_{j,2} - r_{i,2})\boldsymbol{e}_y)] \times \boldsymbol{e}_z}{\| [(r_{j,1} - r_{i,1})\boldsymbol{e}_x + (r_{j,2} - r_{i,2})\boldsymbol{e}_y)] \times \boldsymbol{e}_z\|} \tag{27}$$

$$\tag{28}$$

$$= \frac{-(r_{j,1} - r_{i,1})\boldsymbol{e}_y + (r_{j,2} - r_{i,2})\boldsymbol{e}_x}{\sqrt{(r_{j,1} - r_{i,1})^2 + (r_{j,2} - r_{i,2})^2}}. \tag{29}$$

Depending on the problem type, different possibilities exist for the choice of the numerical flux. We will use the Lax-Friedrichs flux since it is often the most efficient flux and maybe the simplest to implement. The local Lax-Friedrichs flux for a system of equations is given by

$$\boldsymbol{f}_{\boldsymbol{g}}^{*\boldsymbol{e},\boldsymbol{b}}(t) = \frac{1}{2}\left[\boldsymbol{f}_{\boldsymbol{g}}^{\boldsymbol{e},\boldsymbol{b}}(t) + \boldsymbol{f}_{\boldsymbol{g}}^{\tilde{\boldsymbol{e}},\boldsymbol{b}}(t)\right] + \frac{C}{2}\boldsymbol{n}_{\boldsymbol{g}}^{\boldsymbol{e},\boldsymbol{b}}\left[u_g^{e,b}(t) - u_g^{\tilde{e},b}(t)\right] \quad \text{with} \quad C = \max\left|\frac{\partial \boldsymbol{f}}{\partial u} \cdot \boldsymbol{n}\right|, \tag{30}$$

where $\boldsymbol{n}_{\boldsymbol{g}}^{\boldsymbol{e},\boldsymbol{b}}$ is the unit outward normal of edge b of element e evaluated at the Gauss point g and $\tilde{e}$ is the element adjacent to element e.

In the particular case where $\boldsymbol{f}(u) = \boldsymbol{a}u$ with $\boldsymbol{a} \in \mathrm{R}^2$, we have

$$\begin{aligned}\boldsymbol{f}_{\boldsymbol{g}}^{*\boldsymbol{e},\boldsymbol{b}}(t) &= \frac{\boldsymbol{a}}{2}\left[u_g^{e,b}(t) + u_g^{\tilde{e},b}(t)\right] + \frac{|\boldsymbol{a}\cdot\boldsymbol{n}^{\boldsymbol{e},\boldsymbol{b}}|\,\boldsymbol{n}^{\boldsymbol{e},\boldsymbol{b}}}{2}\left[u_g^{e,b}(t) - u_g^{\tilde{e},b}(t)\right] \\ &= \left(\frac{\boldsymbol{a} + |\boldsymbol{a}\cdot\boldsymbol{n}^{\boldsymbol{e},\boldsymbol{b}}|\,\boldsymbol{n}^{\boldsymbol{e},\boldsymbol{b}}}{2}\right) u_g^{e,b}(t) + \left(\frac{\boldsymbol{a} - |\boldsymbol{a}\cdot\boldsymbol{n}^{\boldsymbol{e},\boldsymbol{b}}|\,\boldsymbol{n}^{\boldsymbol{e},\boldsymbol{b}}}{2}\right) u_g^{\tilde{e},b}(t).\end{aligned} \tag{31}$$

Consequently,

$$\boldsymbol{f}_{\boldsymbol{g}}^{*\boldsymbol{e},\boldsymbol{b}}(t)\cdot\boldsymbol{n}_{\boldsymbol{g}}^{\boldsymbol{e},\boldsymbol{b}} = \left(\frac{\boldsymbol{a}\cdot\boldsymbol{n}^{\boldsymbol{e},\boldsymbol{b}} + |\boldsymbol{a}\cdot\boldsymbol{n}^{\boldsymbol{e},\boldsymbol{b}}|}{2}\right) u_g^{e,b}(t) + \left(\frac{\boldsymbol{a}\cdot\boldsymbol{n}^{\boldsymbol{e},\boldsymbol{b}} - |\boldsymbol{a}\cdot\boldsymbol{n}^{\boldsymbol{e},\boldsymbol{b}}|}{2}\right) u_g^{\tilde{e},b}(t). \tag{32}$$

Finally, the value of the unknown function $u_g^{e,b}$ at Gauss point g is obtained by interpolation between the nodal values:

$$u_g^{e,b}(t) = u_h^e(\boldsymbol{\xi}_{\boldsymbol{g}}, t) = \sum_{j=1}^{N_p} u_h^e\left(\boldsymbol{\xi}_{\boldsymbol{j}}^{\boldsymbol{e}}, t\right) l_j^e(\boldsymbol{\xi}_{\boldsymbol{g}}). \tag{33}$$

4.4 Time discretization

Time discretization can be performed by using the time-marching forward-Euler method:

$$\frac{du_j^e}{dt} \simeq \frac{u_{j,t+1}^e - u_{j,t}^e}{\Delta t} = \left(\mathcal{M}_{ij}^e\right)^{-1} \left(\boldsymbol{\mathcal{S}}_{\boldsymbol{ji}}^{\boldsymbol{e}} \cdot \boldsymbol{f}_{\boldsymbol{j}}^{\boldsymbol{e}} + \mathcal{M}_{ij}^e g_j^e + \mathcal{F}_i^e\right) \tag{34}$$

$$\iff u_{j,t+1}^e = u_{j,t}^e + \Delta t \left(\mathcal{M}_{ij}^e\right)^{-1} \left(\boldsymbol{\mathcal{S}}_{\boldsymbol{ji}}^{\boldsymbol{e}} \cdot \boldsymbol{f}_{\boldsymbol{j}}^{\boldsymbol{e}} + \mathcal{M}_{ij}^e g_j^e + \mathcal{F}_i^e\right) \tag{35}$$

However, this numerical scheme is only stable if the Courant–Friedrichs–Lewy (CFL) condition is satisfied:

$$c\frac{\Delta t}{\Delta x} < 1, \tag{36}$$

with c the largest propagation speed, Δt the largest time step and Δx the smallest mesh size. It should be noticed that the stability also depends on the order of the basis functions. If the order is increased, the mesh size and/or the time step should be refined accordingly.

4.5 Numerical implementation

This section explains shortly how we implemented the DG-scheme with the help of Gmsh [5] to solve different kinds of problems based on the advection and Maxwell equations. Our code is restricted to 2D-problems, but it should be possible to extend it to 3D-problems. To generate various geometries depending on the desired problem, we wrote a few lines at the beginning of the **main.cpp** file. Some geometry and simulation parameters, like the mesh size, the order of the basis functions, the time step, the number of time steps, and the mesh type (triangles/quadrangles) can be tuned in the **simParam.txt** file. This file is read at the beginning of the code by the function `readSimulationParam` implemented in **readParam.cpp**.

The implementation of the solver starts once the type of problem, the geometry, and the mesh type have been chosen. At first, global information, like element tags and node coordinates, is gathered about the 2D and 1D elements in the mesh. A function, called `sortNodes0D` and implemented in **build0D.cpp**, is used to reorder the nodes according to their coordinates, so that they can be accessed more easily in the following parts of the code. For the 2D elements, the basis functions, gradients of basis function, integration weights, Jacobian matrices, and Jacobians evaluated at the different Gausss points are obtained by calling the function `getInfo2D` implemented in the **build2D.cpp** file. The assembly of the mass matrix and its inverse is done in the **buildM.cpp** file. Noticing that the Jacobian is the only quantity in (20) depending on the element coordinates in the real space, the product of basis functions and weight can be stored once for each integration point, before computing the mass matrix for all mesh elements. To effectively build the mass matrix and its inverse, the function `buildM` must be called. The computation of the inverse Jacobian matrix and the assembly of the stiffness matrix is done by calling the function `buildS` which is stored in the **buildS.cpp** file.

To compute the flux vector, we have to exploit specific information about 1D elements. This information is gathered by calling the function `getInfo1D` implemented in the **build1D.cpp** file. It provides the node tags, basis functions, integration weights, and Jacobians evaluated at the Gauss points for all 1D elements in the mesh. However, before calling this function, 1D elements must be generated inside the mesh. Elements on the boundary must not be created, since they already exist. This is done by calling the function `create1DSegments`, also stored in the **build1D.cpp** file. At the same time, we store the neighbor element at the other side of the newly created 1D segment, because this information is needed to compute the Lax-Friedrichs fluxes. Before assembling the flux vector, the normal to each 1D element is computed and stored by calling the function `storeNormals` detailed in the **buildN.cpp** file. Moreover, one important key of the code is to call the function `getIndex` implemented in **buildF.cpp**. This function allows to set the correspondence between the node tags and the node of specific element edge. This step is crucial to perform the computations on the right unknown nodal values.

It is interesting to notice that all the steps described above are similar for the advection and Maxwell equations. This makes the code very versatile. Aside from the geometry, boundary/initial conditions, and other physical parameters, the difference between the advection and Maxwell equations lies in the computation of the numerical fluxes. This is the reason why, henceforth, we divide the code into two possible pathways: the advection code, which is run by calling `run_advection` detailed in **advection.cpp**, and the Maxwell code, which is chosen by calling `run_Maxwell` implemented in **maxwell.cpp**.

The advection code starts by reading the physical parameters of the problem and taking into account initial and boundary conditions. The parameters that can be tuned are detailed in the **advectionParam.txt** file. The three advection equations are the same, but they can be tuned with different initial and boundary conditions. To find an approximate solution of the advection equation, we implemented the numerical flux described in (32) and used the forward-Euler method (35) to compute the nodal values at the next time step. Finally, it is possible to compute nodal errors and the average nodal error of the first advection equation (u_1) incurred after a certain number of time steps.

The Maxwell code is divided into several problems, which will be described in detail in the following chapters. For each problem, there exists a specific file to tune some parameters. These parameters are used to impose initial or boundary conditions. The structure of the code to compute the nodal values at the next time step is the same as for the advection code. One difference is that the physical fluxes are given by (75). Moreover, some equations have been added to take into account reflective boundary conditions, perfectly matched layers, and time dependent solutions.

Finally, depending on the desired sampling rate, the nodal values are stored for specific time steps and then displayed in Gmsh. However, the number of time steps divided by the sampling rate should be an integer value.

5 Validation of the advection problem

To test our DG code, we initiate a Gaussian function at the center of a unit square domain, and look how it is transported through the domain depending on the imposed velocity. The initial condition of the Gaussian is

$$u(x,0) = A\exp\left(\frac{-(x-x_0)^2-(y-y_0)^2}{2C^2}\right), \tag{37}$$

with A its amplitude and C its half-height width. The exact solution to the advection equation

$$\frac{\partial u}{\partial t} + c_x\frac{\partial u}{\partial x} + c_y\frac{\partial u}{\partial y} = 0, \tag{38}$$

is then given by

$$u(x,t) = A\exp\left(\frac{-(x-x_0-c_xt)^2-(y-y_0-c_yt)^2}{2C^2}\right), \tag{39}$$

with c_x the propagation velocity along the x-axis and c_y the velocity along the y-axis.

The following simulation parameters can be used to do this test again:

$x_{\max}$: 1
$y_{\max}$: 1
mesh size: 0.025
time step: 5e-5
number of time steps: 100000
sampling: 1000
order of basis functions: 2
order for gauss integration: 4
mesh type: 3

Moreover, to compute the error of the first advection equation (u_1), the error option in the **advectionParam.txt** file should be set to 1. The advection problem is then launched by typing:

```
./bin/main 1
```

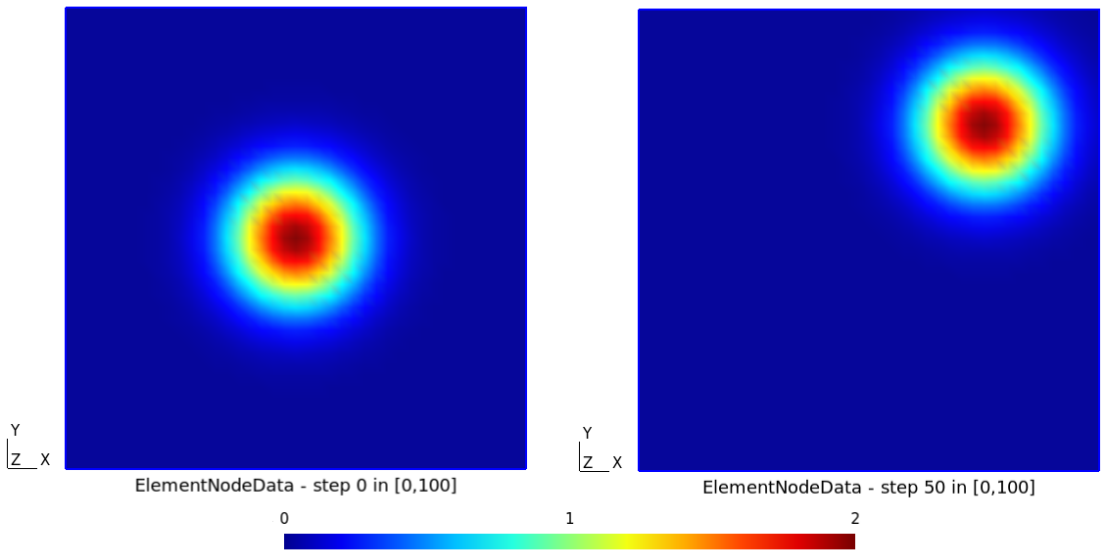


Figure 1: Advection of a Gaussian function ($A = 2$, $C = 0.1$) initialized at the center of the domain ($x_0 = 0.5$, $y_0 = 0.5$), in a diagonal velocity field ($c_x = 0.1$, $c_y = 0.1$).

5.1 Convergence

To verify whether the numerical solution is a good approximation of the analytical solution given in (39), we define the relative percent difference (RPD):

$$\varepsilon(t) = \frac{2}{N} \sum_{i=0}^{N} \frac{|u_{\text{exact,i}}(t) - u_{\text{num,i}}(t)|}{|u_{\text{exact,i}}(t)| + |u_{\text{num,i}}(t)|}, \tag{40}$$

where the sum is performed over the number of unknown nodal values. We decided to use this type of error, because the relative error is not defined when the analytical solution equals zero. It is also a useful tool to identify when the code diverges. If the code is unstable because the CFL condition is not satisfied or because the mesh is too coarse, computing this error will either output 2 or nan (not a number). An error smaller than 10^{-2} can already be assumed to be a good approximation of the analytical solution.

The convergence is influenced by the choice of numerical parameters, namely the mesh size, the order of the basis functions, and the time step. Figure 2 gives the evolution of the error at the end of the time loop as a function of the mesh size and the time step for different orders. The longer the simulation time, the larger the error at the end of the time loop, because errors accumulate at each time step. However, if the time step and mesh size are small enough, the effective simulation time before deviating strongly from the analytical solution can be a few seconds. Moreover, errors from one time step to another may compensate each other.

The DG-FEM scheme states that for the advection equation, the absolute difference between the approximated solution and the exact solution is proportional to a given constant multiplied by a given power of the mesh size. This power is linked to the order of the basis functions: the higher the order of the basis function, the larger the power, and the faster the numerical scheme converges towards a residual error. This is coherent with the results shown on the left-hand side in Figure 2: the higher the order, the larger the slope of the function that would fit the data points.

The second graph expresses the error as a function of the time step. We notice a sharp jump between a time step of order 10^{-3} and a time step of 10^{-2}. This indicates that the numerical scheme is unstable if the time step is too large compared to the mesh size, which, in this case, has been set to 0.01. Moreover, it is possible to fit a polynomial of the second order to the data points with time steps smaller than 10^{-3}. This is related to the error of the forward-Euler method that approximates the time derivative at first order. As a consequence, the error of the difference between the exact solution and the approximated should exhibits a squared dependence on the time step. Increasing the order of the basis functions above 2 for a mesh size of 0.01 and a time step smaller than 10^{-3} does not allow to decrease the error further, as the residual error for this fixed mesh size is already achieved with basis functions of the second order. Besides, above order 6, computations become unstable because we enforce the shape functions to be zero at too many nodes. This results in oscillations in the nodal error.

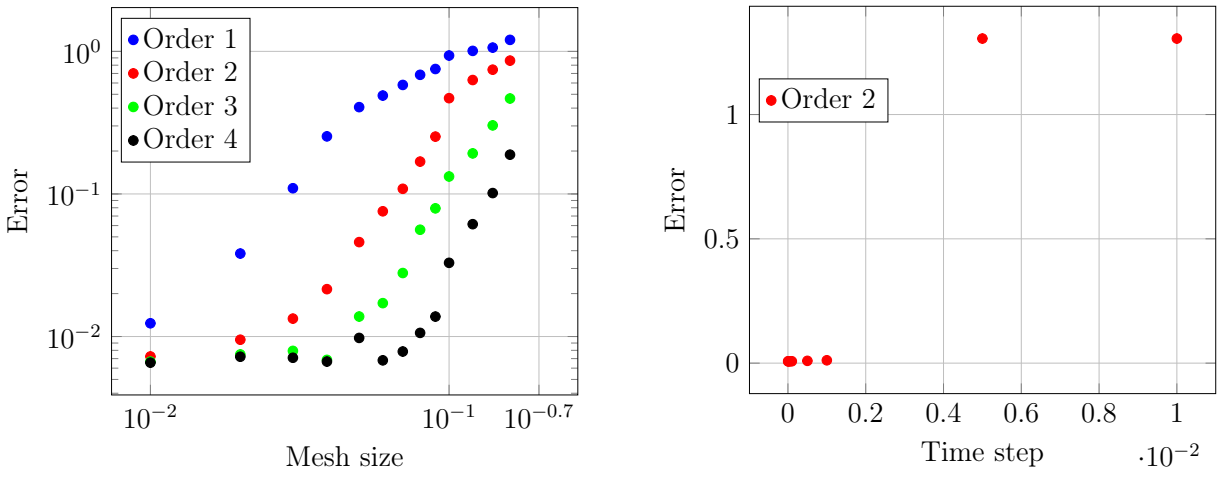


Figure 2: Nodal error for the advection problem after 1 second of simulation as a function of the mesh size for a fixed time step of 1×10^{-5} (left) and as a function of the time step for a fixed mesh size of 0.01 (right). The parameters of the gaussian used to perform these simulations were $A = 1$, $C = 0.1$, $x_0 = 0.5$, $y_0 = 0.5$, $c_x = 0.1$, $c_y = 0.1$.

5.2 Stability

In this section, the stability of the forward-Euler method for the advection equation is studied. Important parameters that influence the stability of the numerical scheme are the mesh size and the time step. According to (36), the relationship between the mesh size and the time step that gives the threshold of instability is linear. The data points in Figure 3 correspond to the smallest mesh size for which the error is higher than a threshold value at the end of the time loop for a given time step. Therefore, this function is supposed to give the threshold of instability: all the points above the curve are unstable.

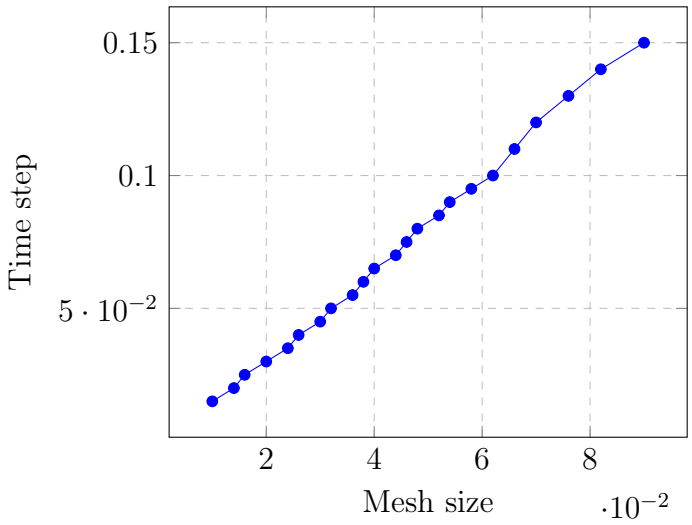


Figure 3: Time step as a function of the mesh size at points were the solution becomes unstable with $c_x = 0.15$ and $c_y = 0$ for a simulation time equal to $1s$ and basis function of the first order.

Nevertheless, an additional parameters than comes into play is the order of the basis functions. The higher the order of the basis functions, the faster the numerical scheme becomes unstable for a fixed mesh size and time step.

6 Parallelization with OpenMP

In order to decrease the execution time of our code, the loops to build the flux vector and to compute the nodal values at the next time step are parallelized. We only focus on this part of the code because the total execution time of the code is mainly affected by this part. Indeed, the mass matrix, stiffness matrix, and other information making up the solver are only computed once, while the flux vector and the forward Euler method must be evaluated at each time step.

In OpenMP, the most efficient way to proceed is to parallelize the outer loop. This loop over the elements is also the biggest one. It is not efficient to parallelize inner loops in OpenMP, because it takes significant time to create and destroy threads. To assess the efficiency of the parallelization, a strong and a weak scaling analysis is performed on Nic4 by earmarking a node with 16 threads.

6.1 Strong scaling

The strong scaling analysis is characterized by a constant execution workload. Thus, for each run on Nic4, the same workload is processed by fixing the simulation parameters at the values mentioned in the table of Figure 4. Only the number of threads is increased. The number of nodes corresponds to the number unknowns for which nodal values are computed.

To asses the efficiency of the parallel region, we define the speedup

$$s = \frac{t_1}{t_N} \tag{41}$$

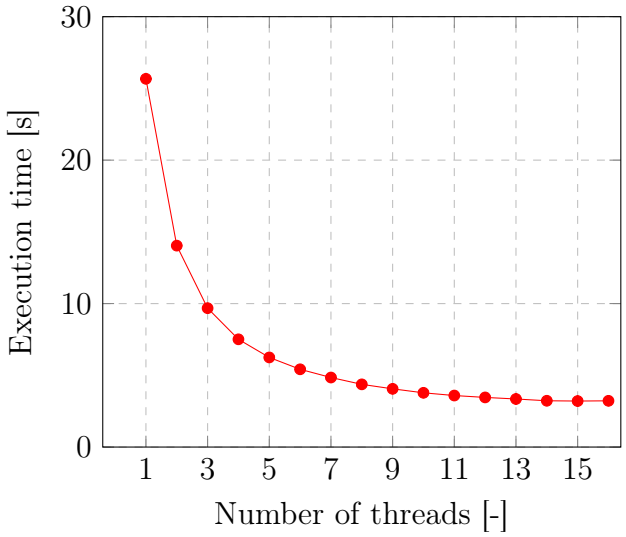
and the strong scaling efficiency

$$S = \frac{t_1}{N t_N} \times 100, \tag{42}$$

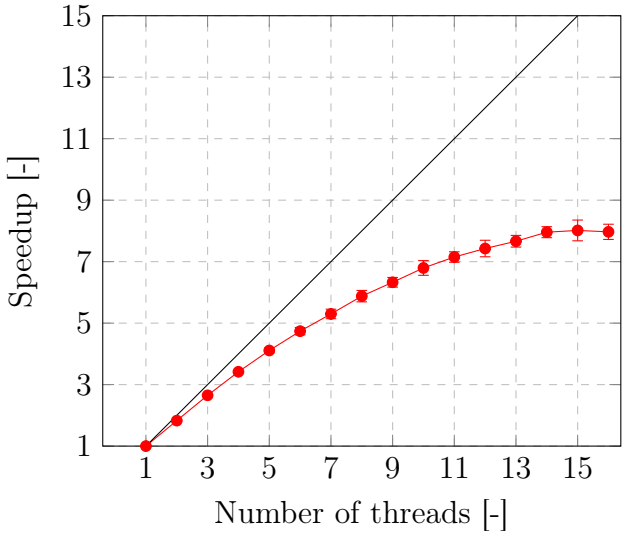
with t_1 the time to process the time loop with one thread and t_N the time required to complete the same time loop with N threads. Ideally, the speedup should be a straight line and the strong scaling efficiency should be 100%, as shown by the black lines in Figure 4.

According to our data points, a maximum speedup of 8 is obtained with 15 threads, but the corresponding scaling efficiency is only 50%. The graphs indicate that the parallel performance decreases when the number of threads increases, especially above 8 threads. This is because creating a parallel region has a cost. In each iteration of the time loop, a parallel region is created and destroyed two times. Hence, by increasing the number of threads, we better distribute the workload over the threads, but we also increase the cost in time to create and destruct them.

Finally, it important to notice that the speedup strongly depends on the amount of data that must be computed. The larger the workload distributed over the threads, the better the scaling.



Number of elements	25 088
Number of nodes	526 848
Mesh size	0.009
Order of basis functions	5
Order of Gauss integration	10
Time step [s]	0.0009
Simulation time [s]	0.09



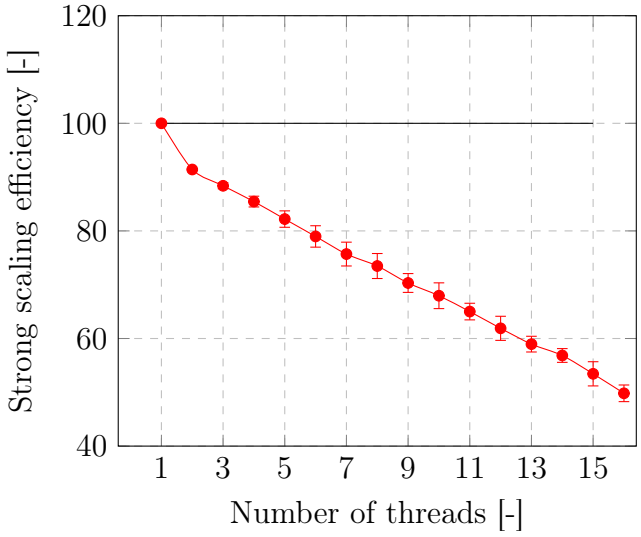


Figure 4: Execution time, speedup and strong scaling efficiency as a function of the number of threads after parallelizing the forward-Euler method using OpenMP. The parameters used for the simulation are summarized in the table.

6.2 Weak scaling

The weak scaling consists in studying the efficiency of the parallelization when each thread has the same workload. It means that the total workload given to the code needs to be a function of the number of threads that is used to execute it. Thus, the number of elements that is given to a thread needs to be a constant. For a mesh made of triangles, the number of elements N_e as a function of the mesh size h is given by:

$$N_e = \frac{2}{h^2}. \tag{43}$$

As the number of elements given to a thread is a constant, we have:

$$C = \frac{2}{h^2 N_{th}}, \tag{44}$$

where $C = N_e/N_{th}$ is the number of elements given to a thread, and N_{th} is the number of threads that is used. The mesh size that should be used for a given number of threads is therefore:

$$h = \sqrt{\frac{2}{C N_{th}}}, \tag{45}$$

where C is a constant. We chose to fix this constant to 1953.25, such that a mesh size of 0.008 is obtained for 16 threads.

As for the strong scaling, we can define the weak scaling efficiency as:

$$S = \frac{t_1}{t_N} \times 100, \tag{46}$$

which should be ideally equal to 100.

Figure 5 gives the execution time and the weak scaling efficiency as a function of the number of threads for the simulation parameters reported in Table 1. It can be seen that the evolution of the execution time is almost linear. The reason is the same as for the strong scaling: the larger the number of threads, the greater the cost in time to create/destroy threads. Another source of error is that imposed mesh size does not perfectly correspond to the value given by (45). By consequence, the number of elements per thread is not always exactly the same.

Number of elements per thread	1953.25
Order of basis functions	5
Order of Gauss integration	10
Time step [s]	0.0009
Simulation time [s]	0.09

Table 1: Simulation parameters for the weak scaling analysis with OpenMP.

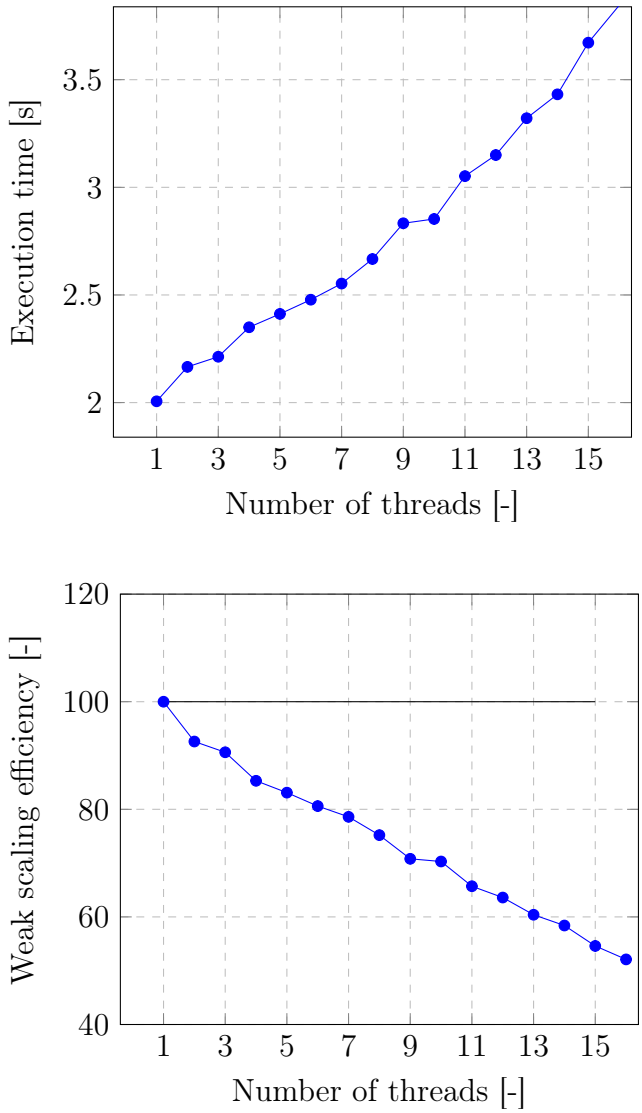


Figure 5: Execution time and weak scaling efficiency as a function of the number of threads after parallelizing the forward-Euler method using OpenMP. The parameters used for the simulation are summarized in Table 1.

7 Acceleration on GPU

In order to apply our code to more complex problems, like guiding a wave through an S-shaped waveguide or observing diffraction on both sides of a periodic series of waveguides, we have to expand the parallel computing performance of our code. Therefore, we will make our program run on accelerators (GPUs).

OpenACC is very similar to OpenMP: pragma commands are added to the code above loops that should run concurrently and be accelerated. However, to avoid unnecessary data transfer between the CPU and the GPU, data regions must be added where needed. Like for the parallelization in OpenMP, only the parts of the code to build the flux vector and to compute the nodal values at the next time step using the forward-Euler method have been parallelized and accelerated on GPU.

To avoid data transfer at each time step, a data region has been defined around the time loop. `Copyin` clauses are used when the values of the vector are needed at the first time step, while there is no need to copy them back to the GPU when all the time steps have been evaluated. The `create` clause is called if the vector is only used temporarily on the GPU, i.e. when there is no need to initialize them or to copy them back to the GPU at the end of the time loop. The `copyout` clause is used to export the nodal values at the time steps corresponding to the sampling rate.

Pointers have been defined to replace vectors, as they cannot be used in the unmanaged acceleration mode. The code works well with vectors in the managed mode, but to avoid any implicit data transfer, it is recommended to remove this option. This allowed us to better define the data region and achieve a better speedup. When parallelizing nested loops, we used the following hierarchy: gang, worker(4), and vector(32). Specifying the worker and vector loops avoids long length vectors, called vector(128), which perform operations on null vectors when the number of loop iterations is small. Finally to call functions from the GPU, a sequential acc routine must be defined in the header file containing the definition of this function.

To assess the performances of our GPU-accelerated solver, we compare the computation time for the advection, the cavity, and the oscillating charge problem when the code is run sequentially on CPU, parallellized on CPU (24 cores), and accelerated on GPU. The simulation parameters for all these codes were a mesh size of 0.01, a triangular mesh type, and a time step of 5×10^{-5}.

During the simulations, we noticed that the order of basis functions plays an important role in the acceleration performances. For basis functions of the first order - and a gauss integration of the second order - the speedup for the code run on GPU with respect to the code executed sequentially is lower than the speedup for the code run on multicore (CPU). The computation times and speedups for the different problems are reported in Table 2.

Problem	Sequential on CPU	Parallel on CPU		Accelerated on GPU	
	Time	Time	Speedup	Time	Speedup
Advection	30.72 min	1.32 min	23.24	2.31 min	13.30
Cavity	20.43 min	0.88 min	23.15	1.80 min	11.37
Charge	31.79 min	1.30 min	24.40	2.06 min	15.39

Table 2: Comparison of the computation time and speedup for different problems run sequentially on CPU, parallelized on CPU (24 cores), and accelerated on GPU. All the tests have been performed on Treebeard. The basis functions were of order 1 and the number of time steps was 100 000.

Problem	Sequential on CPU	Parallel on CPU		Accelerated on GPU	
	Time	Time	Speedup	Time	Speedup
Advection	12 h 48 min	8.36 min	91	2.76 min	278
Cavity	8 h 29 min	4.26 min	119	1.82 min	279
Charge	> 10 h	9.02 min	> 66	2.39 min	> 251

Table 3: Comparison of the computation time for different problems run sequentially on CPU, parallelized on CPU (24 cores), and accelerated on GPU. To asses the sequential computation time, we ran the code on a private node of Nic4, with only one thread. The basis functions were of order 5 and the number of time steps was 30 000.

By contrast, when the order of the basis functions is 5, the speedup for the code run on GPU with respect to the code executed sequentially is higher than the speedup for the code run on multicore (CPU). This is shown in Table 3. The extremely large speedup should be taken with caution, since the sequential code with basis functions of order 5 has been run on Nic4, while the others have been run on Treebeard. Nevertheless, this study indicates that the performance of the GPU increases when the workload is increased. Hence, for high order problems with a big geometry and a small mesh size, running the code on GPU instead of CPU can save a lot of time.

8 Application to Maxwell's equations

Firstly, let us remind the differential form of the four Maxwell's equations in a vacuum:

$$\nabla \cdot \boldsymbol{E} = \frac{\rho}{\varepsilon_0} \qquad \text{(Gauss's law)}, \tag{47}$$

$$\nabla \cdot \boldsymbol{B} = 0 \qquad \text{(Magnetic Gauss's law)}, \tag{48}$$

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t} \qquad \text{(Faraday's law)}, \tag{49}$$

$$\nabla \times \boldsymbol{B} = \mu_0 \left(\boldsymbol{J} + \varepsilon_0 \frac{\partial \boldsymbol{E}}{\partial t} \right) \qquad \text{(Ampère's law)}, \tag{50}$$

where $\boldsymbol{E}$ is the electric field in [V/m], $\boldsymbol{B}$ is the magnetic flux density in [T], ρ is the volume charge density in [C/m^3], $\boldsymbol{J}$ is the current charge density in [A/m^2], $\varepsilon_0 = 8.85 \times 10^{-12}$ [F/m] is the electric permittivity of free space, and $\mu_0 = 4\pi \times 10^{-7}$ [H/m] is the magnetic permeability of free space. The position and time dependence $(\boldsymbol{r}, t)$ of the vector and scalar fields are omitted for clarity.

These equations can be rewritten for a medium of permittivity ε and permeability μ by introducing the electric displacement field $\boldsymbol{D}$ and the magnetic field $\boldsymbol{H}$, which are respectively related to $\boldsymbol{E}$ and $\boldsymbol{B}$ by material-dependent constitutive equations. For linear, isotropic, homogeneous media, ε and μ are scalars and the constitutive equations write:

$$\boldsymbol{D} = \varepsilon \boldsymbol{E}, \tag{51}$$

$$\boldsymbol{B} = \mu \boldsymbol{H}. \tag{52}$$

8.1 Electromagnetic waves

The four Maxwell's equations for a source free ($\rho = 0$), non-conducting ($\boldsymbol{J} = 0$), linear, isotropic, and homogeneous region take the form:

$$\nabla \cdot \boldsymbol{E} = 0 \tag{53}$$

$$\nabla \cdot \boldsymbol{B} = 0 \tag{54}$$

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t} \tag{55}$$

$$\nabla \times \boldsymbol{B} = \mu\varepsilon \frac{\partial \boldsymbol{E}}{\partial t}. \tag{56}$$

By taking the curl of (55), using a well-known vector identity, and injecting (53) in the following relationship:

$$\boldsymbol{\nabla} \times \boldsymbol{\nabla} \times \boldsymbol{E} = \boldsymbol{\nabla}(\boldsymbol{\nabla} \cdot \boldsymbol{E}) - \boldsymbol{\nabla}^2 \boldsymbol{E} = -\boldsymbol{\nabla}^2 \boldsymbol{E} = -\boldsymbol{\nabla} \times \frac{\partial \boldsymbol{B}}{\partial t}, \tag{57}$$

we find that by taking the time derivative of (56), the electric field satisfies:

$$\boldsymbol{\nabla}^2\boldsymbol{E} - \mu\varepsilon\frac{\partial^2\boldsymbol{E}}{\partial t^2} = 0. \tag{58}$$

Similarly, it is easily shown that the magnetic field satisfies:

$$\boldsymbol{\nabla}^2\boldsymbol{B} - \mu\varepsilon\frac{\partial^2\boldsymbol{B}}{\partial t^2} = 0. \tag{59}$$

The last two equations constitute the electromagnetic wave equations. The speed of the wave is given by:

$$v = \frac{1}{\sqrt{\varepsilon\mu}}. \tag{60}$$

In a vacuum, the speed of the wave corresponds to the speed of light:

$$c_0 = \frac{1}{\sqrt{\varepsilon_0\mu_0}} = 3 \times 10^8\,\mathrm{m/s}. \tag{61}$$

The wave equations are solved by:

$$\boldsymbol{E}(\boldsymbol{r}, t) = \boldsymbol{E}_0 \exp\big(j\,(\omega t - \boldsymbol{k}\cdot\boldsymbol{r})\,\big), \tag{62}$$

$$\boldsymbol{B}(\boldsymbol{r}, t) = \boldsymbol{B}_0 \exp\big(j\,(\omega t - \boldsymbol{k}\cdot\boldsymbol{r})\,\big), \tag{63}$$

where $\boldsymbol{k}$ is the wave vector and ω is the frequency of the wave, provided that the dispersion relation

$$\frac{\omega^2}{\boldsymbol{k}^2} = \frac{1}{\varepsilon\mu} = v^2 \tag{64}$$

is satisfied. However, (62) and (63) must also satisfy Maxwell's equations; these constraints imply that $\boldsymbol{k}$, $\boldsymbol{E_0}$, and $\boldsymbol{B_0}$ are mutually perpendicular. Besides, the magnitudes of the electric and magnetic fields satisfy

$$\frac{E_0}{B_0} = \frac{1}{\sqrt{\varepsilon\mu}} = v. \tag{65}$$

While the electromagnetic modes in free space are plane waves in which the electric and magnetic fields are both perpendicular to the direction of propagation of the wave, the two principal modes in a bounded medium are transverse electric (TE) and transverse magnetic (TM) modes. In the TE mode, the electric field $\boldsymbol{E}$ is transverse to the direction of propagation $\boldsymbol{k}$, but the magnetic field $\boldsymbol{H}$ is not. In the TM mode, the magnetic field $\boldsymbol{H}$ is transverse to the direction of propagation $\boldsymbol{k}$, but the electric field $\boldsymbol{E}$ is not.

8.2 Numerical fluxes

In order to find an expression for the numerical fluxes, let us consider source free, non-conducting, linear, isotropic, and homogeneous region, so that Faraday's law and Ampère's law can be rewritten as:

$$\begin{cases} \boldsymbol{\nabla} \times \boldsymbol{E} = -\mu \dfrac{\partial \boldsymbol{H}}{\partial t}, \\ \varepsilon \dfrac{\partial \boldsymbol{E}}{\partial t} = \boldsymbol{\nabla} \times \boldsymbol{H}. \end{cases} \tag{66}$$

For simplicity, we restrict our study to the TM modes in two dimensions. Assuming that the electromagnetic wave propagates in the z direction, we keep only two components of the magnetic field (H^x, H^y) and the electric field E_z. The above equations then become:

$$\begin{cases} \mu \dfrac{\partial H^x}{\partial t} = -\dfrac{\partial E^z}{\partial y}, \\ \mu \dfrac{\partial H^y}{\partial t} = \dfrac{\partial E^z}{\partial x}, \\ \varepsilon \dfrac{\partial E^z}{\partial t} = \dfrac{\partial H^y}{\partial x} - \dfrac{\partial H^x}{\partial y}. \end{cases} \tag{67}$$

This system of equations can be normalized by introducing the phase velocity v, i.e. the speed of the wave, and by defining the dimensionless variables

$$\begin{aligned} &\bar{t} = \frac{vt}{L}, \quad \bar{x} = \frac{x}{L}, \quad \bar{y} = \frac{y}{L}, \quad \bar{z} = \frac{z}{L}, \\ &\bar{H}^x = \frac{H^x}{H_0}, \quad \bar{H}^y = \frac{H^y}{H_0}, \quad \bar{E}^z = \frac{E^z}{E_0} = \frac{E^z}{Z H_0}, \end{aligned} \tag{68}$$

where L [m] is some reference length (typically the wavelength of the phenomena of interest), H_0 [A/m] is a unit magnetic field strength, E_0 [V/m] is a unit electric field strength. The magnetic and electric field strength are related to each other by the wave impedance Z, expressed in [Ω]:

$$Z = \frac{E_0}{H_0} = \frac{\mu\, E_0}{B_0} = \sqrt{\frac{\mu}{\varepsilon}}. \tag{69}$$

In a vacuum, the impedance of free space is given by:

$$Z_0 = \sqrt{\frac{\mu_0}{\varepsilon_0}} \simeq 377\,\Omega. \tag{70}$$

The dimensionless equations become:

$$\begin{cases} \dfrac{\partial \bar{H}^x}{\partial \bar{t}} = -\dfrac{\partial \bar{E}^z}{\partial \bar{y}}, \\ \dfrac{\partial \bar{H}^y}{\partial \bar{t}} = \dfrac{\partial \bar{E}^z}{\partial \bar{x}}, \\ \dfrac{\partial \bar{E}^z}{\partial \bar{t}} = \dfrac{\partial \bar{H}^y}{\partial \bar{x}} - \dfrac{\partial \bar{H}^x}{\partial \bar{y}}. \end{cases} \tag{71}$$

In what follows, we will drop the bar notation for the dimensionless variables to alleviate the notations. Noticing that

$$\begin{cases} \dfrac{\partial E^z}{\partial y} = \boldsymbol{\nabla} \cdot \boldsymbol{f_{H^x}} \quad \text{with} \quad \boldsymbol{f_{H^x}} = \begin{pmatrix} 0 & E^z \end{pmatrix}^{\mathrm{T}} \\ -\dfrac{\partial E^z}{\partial x} = \boldsymbol{\nabla} \cdot \boldsymbol{f_{H^y}} \quad \text{with} \quad \boldsymbol{f_{H^y}} = \begin{pmatrix} -E^z & 0 \end{pmatrix}^{\mathrm{T}} \\ \dfrac{\partial H^x}{\partial y} - \dfrac{\partial H^y}{\partial x} = \boldsymbol{\nabla} \cdot \boldsymbol{f_{E^z}} \quad \text{with} \quad \boldsymbol{f_{E^z}} = \begin{pmatrix} -H^y & H^x \end{pmatrix}^{\mathrm{T}}, \end{cases} \tag{72}$$

and by comparing equations (71) and (3), we can identify the flux functions $\boldsymbol{f_{H^x}}$, $\boldsymbol{f_{H^y}}$, $\boldsymbol{f_{E^z}}$. Using the Rankine-Hugoniot conditions, it can be shown [2] that the numerical fluxes write

$$\begin{cases} \boldsymbol{n} \cdot (\boldsymbol{f_{H^x}} - \boldsymbol{f^*_{H^x}}) = \dfrac{n_y}{2} \left[E^z\right] + \dfrac{\alpha}{2} \left(n_x [[\boldsymbol{H}]] - [H^x]\right), \\ \boldsymbol{n} \cdot (\boldsymbol{f_{H^y}} - \boldsymbol{f^*_{H^y}}) = -\dfrac{n_x}{2} \left[E^z\right] + \dfrac{\alpha}{2} \left(n_y [[\boldsymbol{H}]] - [H^y]\right), \\ \boldsymbol{n} \cdot (\boldsymbol{f_{E^z}} - \boldsymbol{f^*_{E^z}}) = \dfrac{n_y}{2} [H^x] - \dfrac{n_x}{2} [H^y] - \dfrac{\alpha}{2} [E^z], \end{cases} \tag{73}$$

where the following notations are defined to shorten the expressions

$$\begin{aligned} [E^z] &\equiv E^{z-} - E^{z+} \\ [H^x] &\equiv H^{x-} - H^{x+} \\ [H^y] &\equiv H^{y-} - H^{y+} \\ [[\boldsymbol{H}]] &\equiv \mathbf{n}^- \cdot \boldsymbol{H}^- + \mathbf{n}^+ \cdot \boldsymbol{H}^+ \\ &= n_x^- \left(H^{x-} - H^{x+}\right) + n_y^- \left(H^{y-} - H^{y+}\right), \end{aligned} \tag{74}$$

and where $-$ denotes the interior element while $+$ denotes the neighboring exterior element. Isolating the numerical fluxes in the left-hand side, injecting (74) and the physical fluxes, provided in (72), in (73), and considering that $E^z = E^{z-}$, $H^x = H^{x-}$, $H^y = H^{y-}$, leads to

$$\begin{cases} \boldsymbol{n} \cdot \boldsymbol{f^*_{H^x}} = \dfrac{n_y}{2} (E^{z-} + E^{z+}) - \dfrac{\alpha}{2} \left(n_x [[\boldsymbol{H}]] - [H^x]\right), \\ \boldsymbol{n} \cdot \boldsymbol{f^*_{H^y}} = -\dfrac{n_x}{2} (E^{z-} + E^{z+}) - \dfrac{\alpha}{2} \left(n_y [[\boldsymbol{H}]] - [H^x]\right), \\ \boldsymbol{n} \cdot \boldsymbol{f^*_{E^z}} = \dfrac{n_y}{2} (H^{x-} + H^{x+}) - \dfrac{n_x}{2} (H^{y-} + H^{y+}) + \dfrac{\alpha}{2} [E^z]. \end{cases} \tag{75}$$

The parameter α in the numerical flux can be used to control dissipation. Taking $\alpha = 0$ yields a non-dissipative central flux and $\alpha = 1$ results in the classic upwind Lax-Friedrichs flux. However, we could also choose α to be any value in between 0 and 1.

8.3 Boundary conditions

By contrast to free space, the electric and magnetic fields must satisfy boundary conditions in a bounded medium. As we will model metallic cavities, waveguides, and photonic crystals in the next sections, the boundary conditions at the surface of a conductor deserve particular attention. The amplitude of an electromagnetic wave propagating through a conductor decreases exponentially over a characteristic length given by the skin depth

$$\delta = \sqrt{\frac{2}{\omega\mu\sigma}}, \tag{76}$$

where σ is the conductivity of the material. Hence, for a perfect conductor with infinite conductivity, the electromagnetic wave cannot penetrate inside the material. Therefore, when an electromagnetic wave impinges on the surface of a perfect conductor, we can assume a complete reflection of the wave.

Because electromagnetic waves are reflected by good conductors, they can be used to store energy in the form of standing waves or to guide energy from one place to another. Numerically, the perfectly conducting surfaces at the boundaries of the cavity, the waveguide, and the photonic crystal are imposed by setting:

$$\begin{aligned} H^{x*} &= H^{x-}, \\ H^{y*} &= H^{y-}, \\ E^{z*} &= -E^{z-}, \end{aligned} \tag{77}$$

where H^{x*}, H^{y*}, and E^{z*} denote the values at the boundary, and H^{x-}, H^{y-}, and E^{z-} denote the values of the boundary elements.

8.4 Perfectly matched layers

The issue when the boundaries of a finite domain are modeled by perfectly conducting surfaces is that the reflected waves all interfere. This may be a problem if we want to represent the electromagnetic field emitted by an oscillating charge or to visualize a plane wave striking the entry of a waveguide, without being perturbed by the reflected waves due to the finite domain extension.

The solution to model waves extending to infinity is to use open boundaries. At an interface between two boundaries, a wave is always (partially) reflected, unless the effective refractive indices of the two materials are exactly the same. To avoid any reflection, the goal is thus to have the same effective refractive index on both sides of the boundary of a perfectly matched layer (PML), with the particularity that the medium beyond the PML is absorptive. This can be simulated by defining those boundaries inside the initial domain.

A particularly simple PML is formed by adding extra terms in (78), (79), (80) and equations (81), (82), (83), (84) to Maxwell's equations that do not involve spatial derivatives. The modified equations for the TM Maxwell equations are [2]:

$$\frac{\partial H^x}{\partial t} = -\frac{\partial E^z}{\partial y} - \sigma^y \left(2H^x + P^y\right) \tag{78}$$

$$\frac{\partial H^y}{\partial t} = \frac{\partial E^z}{\partial x} - \sigma^x \left(2H^y + P^x\right) \tag{79}$$

$$\frac{\partial E^z}{\partial t} = -\frac{\partial H^x}{\partial y} + \frac{\partial H^y}{\partial x} - \frac{d\sigma^x}{dx}Q^x + \frac{d\sigma^y}{dy}Q^y \tag{80}$$

$$\frac{\partial P^x}{\partial t} = \sigma^x H^y \tag{81}$$

$$\frac{\partial P^y}{\partial t} = \sigma^y H^x \tag{82}$$

$$\frac{\partial Q^x}{\partial t} = -\sigma^x Q^x - H^y \tag{83}$$

$$\frac{\partial Q^y}{\partial t} = -\sigma^y Q^y - H^x \tag{84}$$

where $\sigma^x = \sigma^x(x)$ and $\sigma^y = \sigma^y(y)$ define the material property of rectangular absorbing regions. For instance, if we consider the origin of a rectangle domain to be at $(x, y) = (0, 0)$ with length x_{max} and y_{max},

$$\sigma^x := \begin{cases} 0 & |x| < x_0 \\ \sigma_0^x (x - x_0)^p & x \geq x_0 \\ \sigma_0^x (x_{max} - x_0 - x)^p & x \leq x_{max} - x_0 \end{cases}$$

$$\sigma^y := \begin{cases} 0 & |y| < y_0 \\ \sigma_0^y (y - y_0)^p & y \geq y_0 \\ \sigma_0^y (y_{max} - y_0 - y)^p & y \leq y_{max} - y_0 \end{cases}$$

defines a PML that will damp the solution in the region exterior to $\{x_{max} - x_0, x_0\}$ and $\{y_{max} - y_0, y_0\}$. The solution will decay exponentially as it propagates through the layer, with the rate determined by the reference parameters σ_0^x and σ_0^y. The parameter $p = 2$ is a measure of the smoothness of the absorption profile.

To test the implementation of the PML, we developed a simple problem in which an oscillating electric field $E_z = \sin(\omega t)$ is imposed at the center of a unit square domain. The real boundary conditions, drawn in blue, are assumed to be perfect electrical conductors. Therefore, any wave impinging on this boundary will be reflected. Figure 6 shows the electric field distribution without PML on the left-hand side, and with a PML on the right-hand side. The PML is defined with $x_0 = 0.8$, $y_0 = 0.8$, and $\sigma_0 = 2500$, such that any wave crossing these frontiers becomes exponentially attenuated.

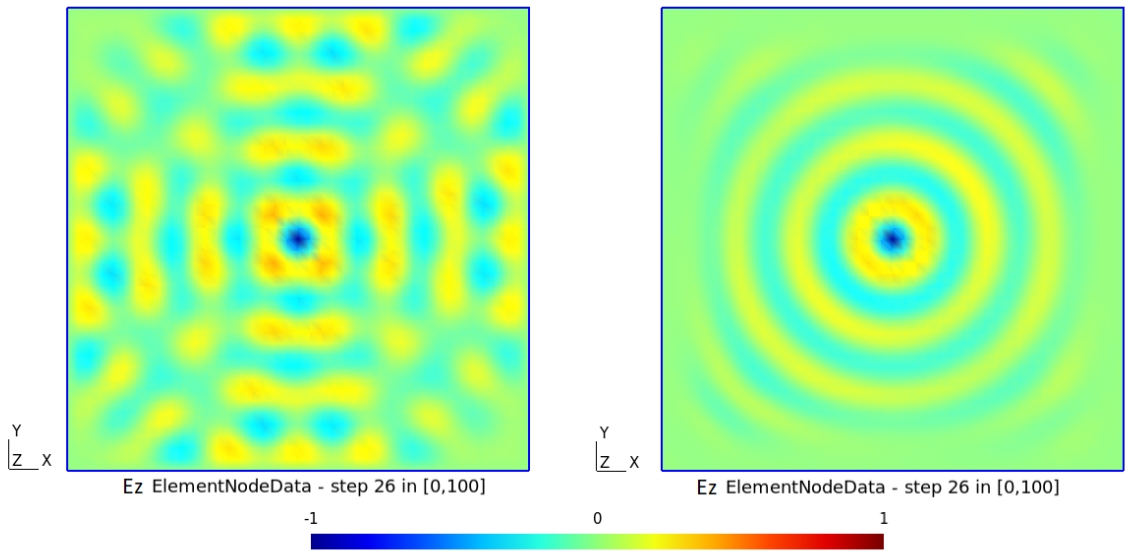


Figure 6: Charge oscillating at the center of a unit square, where PML's are defined with $x_0 = 0.8$, $y_0 = 0.8$, and $\sigma_0 = 2500$.

The result is conclusive, because thanks to the PML the reflected waves do not interfere with the waves emitted by the oscillating charge. As an example, the code has been run by imposing the following parameters in the **simParam.txt** file:

$x_{\max}$: 1

$y_{\max}$: 1

mesh size: 0.025

time step: 1e-5

number of time steps: 100000

sampling: 1000

order of basis functions: 1

order for gauss integration: 2

mesh type: 3

The geometry of the rectangle, the flux parameter $0 < \alpha < 1$, and the PML parameters $0 < x_0 < 1$, $0 < y_0 < 1$, σ_0 can be modified in the **chargeParam.txt** file. Finally, the code is launched with the following command

```
./bin/main 5
```

to select the oscillating charge problem.

8.5 Rectangular cavity

8.5.1 Theory

Let us consider a rectangular cavity with perfectly conducting walls. The dimensions of the cavity are a_x and a_y in the x and y directions respectively. By contrast to Figure 7, we will assume that the dimension of the cavity in the z direction is infinite. By consequence, the only relevant independent variables in this problem are t, x, and y. Moreover, we restrict our analysis to the transverse magnetic (TM) mode. Because z is the direction of propagation of the electromagnetic wave, as this direction is not confined by the walls of the cavity, the only non-zero components of the electromagnetic wave are H_x, H_y, and E_z. Finally, we assume that the cavity is filled with air.

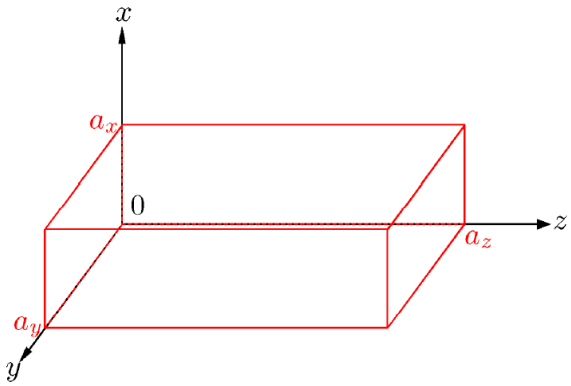


Figure 7: Schematic of a rectangular cavity. Because our code is restricted to 2D problems, we consider that the size of the cavity along the z direction (a_z) is infinity. The electromagnetic wave is thus not confined in this direction.

Inside the cavity, the electric field must satisfy (58), where in good approximation $\varepsilon = \varepsilon_0$ and $\mu = \mu_0$ for air. To satisfy the boundary conditions, the electric field E_z tangential to the wall and the component of the magnetic flux density $\boldsymbol{B} = (B_x, B_y)$ normal to the wall must be zero at the walls of the cavity. Regarding Figure 7, E_z must therefore vanish at

$$x = 0, \quad x = a_x, \quad y = 0, \quad y = a_y. \tag{85}$$

These requirements can be satisfied if

$$E_z(x, y, t) = \sin(k_x x) \sin(k_y y) \cos(\omega t), \tag{86}$$

with

$$k_x = \frac{\pi n_x}{a_x} \quad \text{and} \quad k_y = \frac{\pi n_y}{a_y}, \tag{87}$$

for any integers n_x and n_y. These numbers are called the modes of the electromagnetic wave.

By injecting (86) in the wave equation (58), we notice that the dispersion relation between the wave vector $\boldsymbol{k}$ and the frequency ω is given by:

$$\boldsymbol{k}^2 = k_x^2 + k_y^2 = \frac{\omega^2}{c^2} \quad \Longleftrightarrow \quad \omega = \pi c \sqrt{\frac{n_x^2}{a_x^2} + \frac{n_y^2}{a_y^2}}. \tag{88}$$

The latter relationship indicates that the frequency of the wave is only allowed to take discrete values, since the modes n_x and n_y must be integers.

The expressions for the magnetic field components are obtained by injecting (86) in (55):

$$H^x(x, y, t) = -\frac{k_y}{\omega} \sin(k_x x) \cos(k_y y) \sin(\omega t), \tag{89}$$

$$H^y(x, y, t) = \frac{k_x}{\omega} \cos(k_x x) \sin(k_y y) \sin(\omega t). \tag{90}$$

Nevertheless, because our code has been implemented for dimensionless values and a unit square domain, the frequency that will be used in the rest of this section is

$$\omega = \pi \sqrt{n_x^2 + n_y^2}. \tag{91}$$

This is consistent with the fact that Maxwell's equations have been normalized by the speed of the wave, which is equivalent to the speed of light in vacuum.

8.5.2 Numerical implementation

The cavity problem has been solved numerically by extending the advection code to the three coupled scalar equations given in (71). The two critical parts are implementing the numerical fluxes described in Section 8.2 and taking into account the perfectly conducting boundary conditions mentioned in (77). In order to keep a multifunctional code, we used the following notation for the magnetic and electric fields:

$$u_1 = H^x, \quad u_2 = H^y, \quad u_3 = E^z. \tag{92}$$

The parameters that can be tuned by the user for the cavity problem are gathered in the **cavityParam.txt** file. These parameters are the oscillation modes

$$n_x = 1, 2, 3, \ldots \quad \text{and} \quad n_y = 1, 2, 3 \ldots, \tag{93}$$

the flux parameter α, described at the end of Section 8.2, and an option to compute the (average) nodal error after a certain simulation time. During our simulations we always fixed $\alpha = 1$, because this is the most stable numerical scheme. The error computation option can be set to 1 if the user wants to know how far the numerical solution is from the analytical solution.

To run the code, all parameters in the **simParam.txt** and **cavityParam.txt** should firstly be specified. The dimensions of the rectangle can be modified in the simParam.txt file by changing the values of $x_{\max}$ and $y_{\max}$. Then the code is executed by entering the following command:

```
./bin/main 2
```

The argument is required to select the cavity problem. This defines the rectangular or squared geometry, runs the Maxwell code, uses the perfectly conducting boundary conditions, and sets the initial conditions

$$\begin{aligned} H^x &= 0 \\ H^y &= 0 \\ E^z &= \sin(n_x \pi x) \sin(n_y \pi y), \end{aligned} \tag{94}$$

where x, and y are the coordinates of the nodal values.

The following simulation parameters make it for instance possible to show the oscillation of mode $n_x = n_y = 2$ in a unit squared cavity, while achieving an average nodal error smaller than 4×10^{-4} after a simulation time of 1:

$x_{\max}$: 1

$y_{\max}$: 1

mesh size: 0.01

time step: 1e-5

number of time steps: 100000

sampling: 1000

order of basis functions: 3

order for gauss integration: 6

mesh type: 3

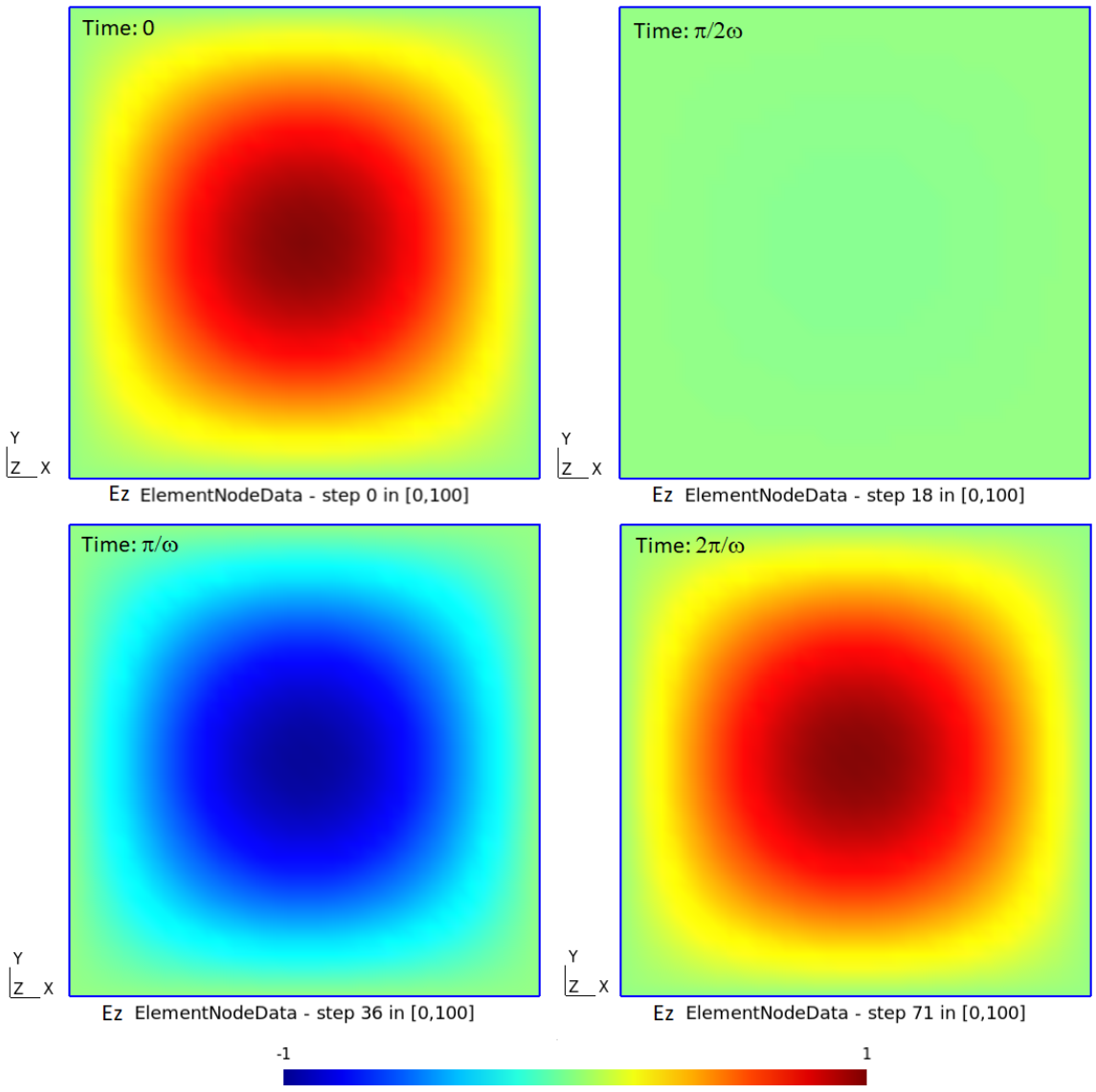


Figure 8: Electric field of the mode $n_x = n_y = 1$ at different times.

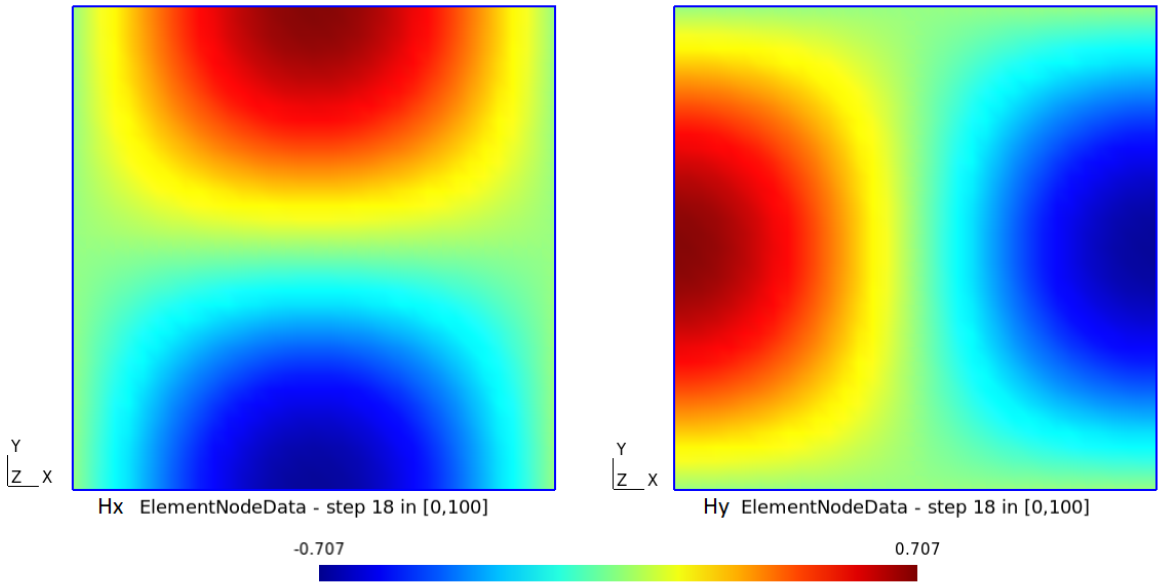


Figure 9: Magnetic field of the mode $n_x = n_y = 1$ at different times.

To confirm that the code displays the correct values, we can compare the simulation results and the analytical solution at a specific time t and particular (x, y) values. Simulations for the first mode are shown in Figure 8 for the electric field and in Figure 9 for the magnetic field. The simulations have been performed with a mesh size of 0.03 (2312 elements), a time step of 1×10^{-5}, and basis functions of order 3. 200 000 time steps have been computed to reach a simulation time of 2. However, data have been saved with a sampling rate of 2 000 steps. The correspondences between simulation steps and real time values is thus the following:

$$t = 2000 \times 1 \times 10^{-5} \times \text{step}. \tag{95}$$

For example, if we consider the first mode $n_x = 1$ and $n_y = 1$ such that $\omega = \sqrt{2}\pi$, one period of the electromagnetic wave lasts

$$T = \frac{2\pi}{\omega} = \frac{2}{\sqrt{2}} = 1.41. \tag{96}$$

The corresponding time step according to (95) is 71. Figure 8 confirms that 71 steps are needed for the electric field to oscillate over one period. However, the figures at initial time and after one period are not exactly the same because the numerical solution is subjected to errors due to a discretized space and time, and a solution approximated by polynomials.

Knowing that the origin of the coordinate system is in the southwest corner and that the domain is a square of dimension $a_x = a_y = 1$, it is readily verified that we have the correct values for the first mode $n_x = n_y = 1$ at the time steps and particular points of the domain reported in Table 4.

Field	t	x	y	Solution	Field	t	x	y	Solution
E^z	0	0.5	0.5	1	H^x	$\pi/2\omega$	0.5	0	-0.707
E^z	$\pi/2\omega$	0.5	0.5	0	H^x	$\pi/2\omega$	0.5	1	0.707
E^z	π/ω	0.5	0.5	-1	H^y	$\pi/2\omega$	0	0.5	0.707
E^z	$2\pi/\omega$	0.5	0.5	1	H^y	$\pi/2\omega$	1	0.5	-0.707

Table 4: Analytical solution of the cavity problem for particular points in space and time. The mode is $n_x = n_y = 1$.

8.5.3 Convergence test

To verify whether the numerical solution is a good approximation of the analytical solution given in (89), (90), and (86), we use the relative percent difference that has already been defined in (40) to analyze the convergence of the advection problem. The analytical solution implemented for the cavity problem is obtained by evaluating (89), (90), and (86) at the nodal coordinates, $t = 2$, $n_x = n_y = 1$, and by using the frequency defined in equation (91).

Figure 10 displays the cumulative error for different time steps and mesh sizes, computed after a simulation time of 2 with basis functions of the first order. The red dots on the top left confirm that the numerical scheme is unstable if the CFL condition is not satisfied: the time step is too big compared to the mesh size. The columns of red dots on the right show that it is not possible to converge towards the analytical solution if the mesh is too coarse. Finally, the error is decreased by several orders of magnitude if the mesh and the time step are both refined.

The error as a function of the mesh size for different orders of the basis functions is given in Figure 11. We notice that increasing the order reduces the error for a fixed mesh size. Moreover, the convergence towards a fixed error is faster for higher orders. The order of convergence can be modeled by a power law

$$\varepsilon \sim h^p, \tag{97}$$

where h is the mesh size, and p is the order of convergence. The fitting parameter p is reported in Table 5 and Table 6 for different orders, for the magnetic field and the electric field respectively. The coefficient of determination R^2 assesses the fitting quality: $R^2 = 1$ indicates that the power law perfectly fits the data.

The residual error ($\varepsilon = 2 \times 10^{-4}$), observed in Figure 11, is quickly obtained by using higher order basis functions (≥ 3). This residual error is due to the fact that a sinusoidal function cannot be perfectly approximated with polynomials. Decreasing the mesh size and time step once this minimal error has been reached is thus a waste of computation time.

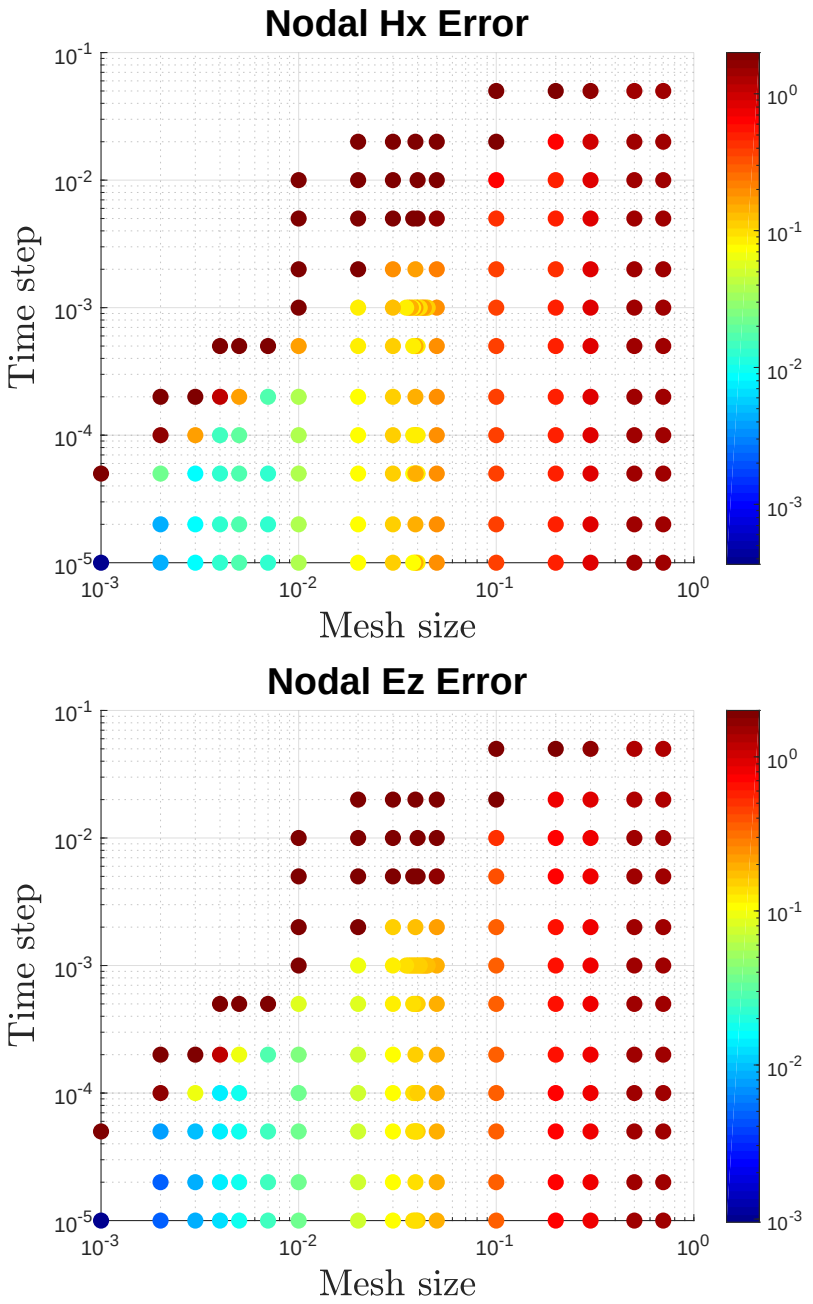


Figure 10: Cumulative error between the analytical and numerical solution of the cavity problem for different time steps and mesh sizes, computed after a simulation time of 2 with basis functions of the first order. The error for the magnetic fields along x and y being the same, only the errors for H^x and E^z are displayed. Dark red dots indicate an unstable numerical scheme.

Fitting a power law for higher orders becomes more complicated because the minimal error can be reached with coarser meshes. For example, by using basis functions of the fourth order and a time step of 1×10^{-5}, the residual error is already achieved for a mesh size of 0.09. To conclude, using higher order basis functions allows to reach a target error with a coarser mesh. However, as the order of the basis functions is increased for a fixed mesh size, the time step must be reduced accordingly.

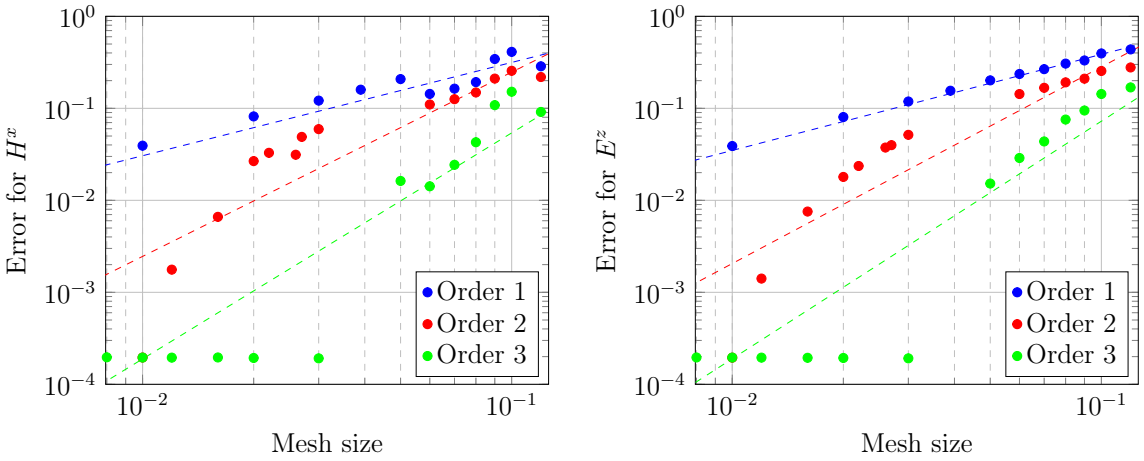


Figure 11: Cumulative error of the magnetic field (left) and electric field (right) after a simulation time of 2 as a function of the mesh size for a fixed time step of 1×10^{-5}.

Order	p	R^2	Error
1	1.014	0.967	0.121
2	2.003	0.8019	0.059
3	2.457	0.8976	1.9×10^{-4}

Table 5: Fitting parameter p, fitting quality R^2, and error of the magnetic field for different orders of basis functions. The error in the last column has been obtained by running the code with a time step of 1×10^{-5} and a total time of 2, and a mesh size of 0.03 (2312 elements).

Order	p	R^2	Error
1	1.045	0.996	0.119
2	2.137	0.841	0.051
3	2.582	0.904	1.9×10^{-4}

Table 6: Fitting parameter p, fitting quality R^2, and error of the electric field for different orders of basis functions. The error in the last column has been obtained by running the code with a time step of 1×10^{-5} for a total time of 2 and a mesh size of 0.03 (2312 elements).

8.5.4 Oscillation modes

For the mode $n_x = n_y = 1$, we saw in Figure 8 that E^z was zero at any time step for

$$\begin{aligned} x &= 0 \quad \forall\, y, \\ x &= 1 \quad \forall\, y, \\ y &= 0 \quad \forall\, x, \\ y &= 1 \quad \forall\, x. \end{aligned} \tag{98}$$

Similarly, Figure 9 shows that H^x is zero at any time step for

$$\begin{aligned} x &= 0 \quad \forall\, y, \\ x &= 1 \quad \forall\, y, \\ y &= 0.5 \quad \forall\, x. \end{aligned} \tag{99}$$

As H^y is obtained by rotating H^x by 90°, the zeros of H^y at any time step are obtained by switching the variables x and y in (99). By increasing the modes n_x, n_y, the oscillation frequency of the magnetic and electric fields is increased, hence H^x, H^y, and E^z exhibit more zeros. For instance, the mode $n_x = n_y = 2$ exhibits extra zeros at $x = 0.5$ $\forall y$ and $y = 0.5$ $\forall x$ for E^z and extra zeros at $x = 0.5$ $\forall y$, $y = 0.25$ $\forall x$, and $y = 0.75$ $\forall x$ for H^x. Simulations for the modes $n_x = n_y = 2$ and $n_x = n_y = 3$ are displayed in Figure 12. The simulation parameters are summarized in Table 7.

Mode	Mesh size	Elements	Time step	Sampling rate	Simulation time
1	0.03	2 312	10^{-5}	2 000	2
2	0.01	20 000	10^{-5}	2 000	2
3	0.008	31 250	10^{-6}	5 000	0.5

Table 7: Simulation parameters used to compute the magnetic and electric fields for different modes $n_x = n_y$.

It is important to notice that the error of the numerical solution with respect to the analytical solution increases with increasing modes. This is shown in Figure 13, where the error of the magnetic field has been computed after one period for different modes, a fixed mesh size of 0.03, a fixed time step of 10^{-5}, and basis functions of the third order. Therefore, care should be taken if the discretized space and time cannot follow the rapid oscillations of the electromagnetic wave.

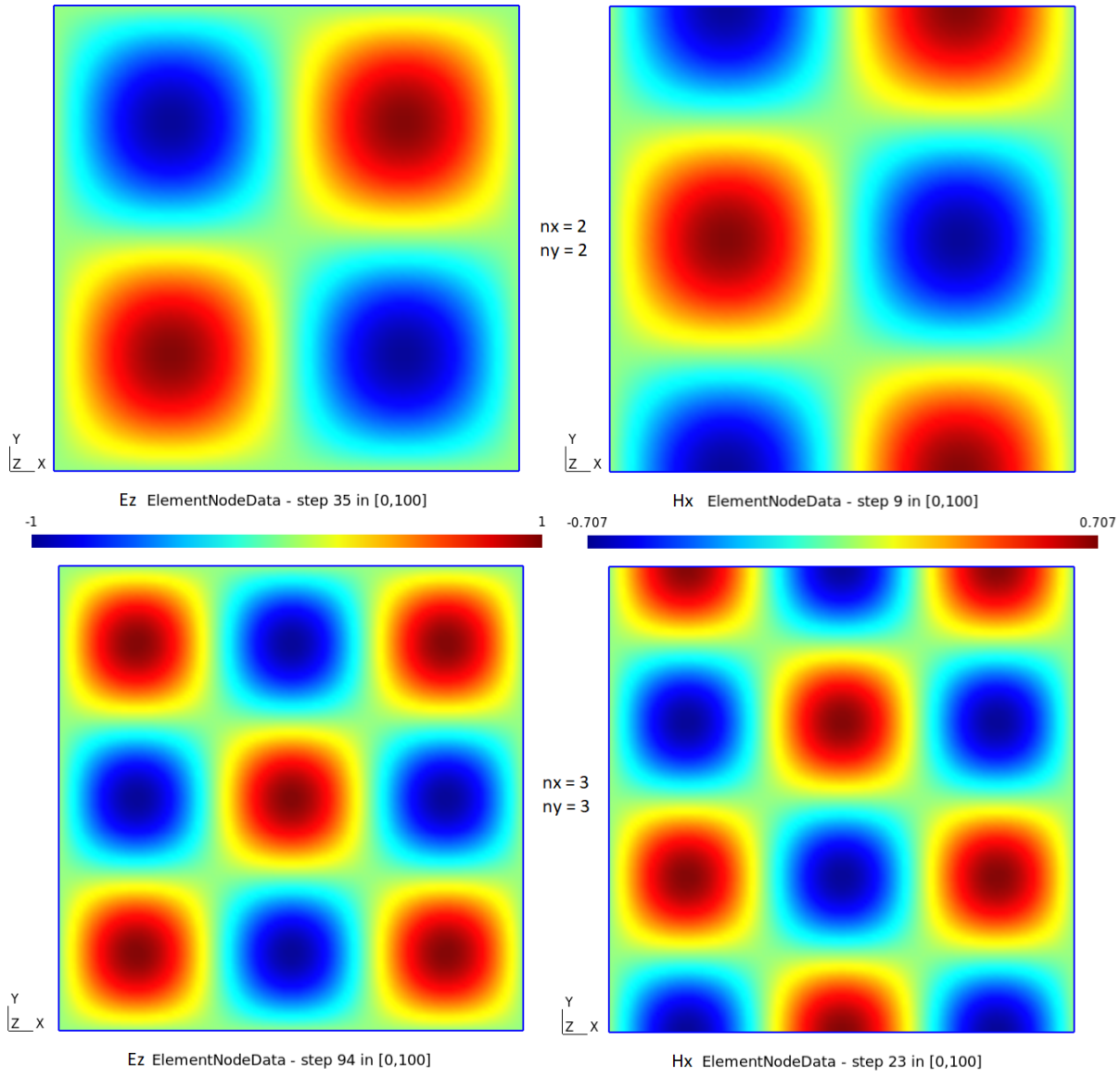


Figure 12: Electric field (left) and magnetic field (right) for the modes $n_x = n_y = 2$ and $n_x = n_y = 3$ after one period for the electric field ($T = 2\pi/\omega$) and after a quarter of a period for the magnetic field.

Mode	Period	Steps	H^x error
1	1.414	141,422	0.0157
2	0.707	70,711	0.225
3	0.471	47,140	0.513
4	0.353	35,355	0.932
5	0.283	28,284	1.245
6	0.236	23,570	1.474
7	0.202	20,203	1.836

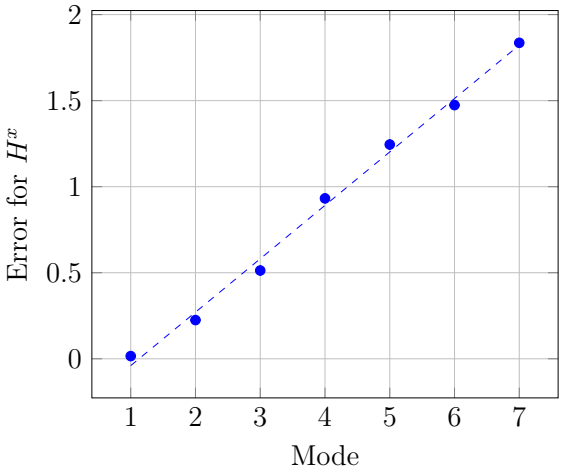


Figure 13: Average nodal error of the magnetic field as a function of the oscillation mode $n_x = n_y$ after one period.

8.5.5 Computation time

The computation time of a numerical code depends on the imposed workload. This workload can be increased by refining the mesh, decreasing the time step, or increasing the order of the basis functions, while keeping the same physical problem. To study how the computation time depends on the mesh size, i.e. on the number of elements or more precisely on the number of nodal unknowns, the element type (triangles or quadrangles), the time step, and the order of the basis functions, we fixed the physical simulation time to one. The imposed problem is the oscillation of the electromagnetic wave in a unit square cavity with perfectly conducting walls and an oscillation mode $n_x = n_y = 2$. The graphs reported in this section have been obtained by fixing all simulation parameters but one. For the fixed parameters, we always kept a mesh size of 0.01, a time step of 10^{-5}, 10^5 steps, and basis functions of the first order.

Let us first analyze the computation time as function of the order of the basis functions. Figure 14 displays the number of unknown nodal values (on the left) and the computation time (on the right) as a function of the order of the basis functions. The computation time reported in this graph has been obtained by running the code on Nic4 with 15 threads. The data points have been fitted with a second order polynomial. On GPU, the computation time is shorter (10 times faster on GPU than on Nic4 for order 5) and the evolution of the computation time as a function of the order can also be fitted with a polynomial of the second order. However, the computation time is more likely to be influenced by programs running on the background, as we do not have the ability to reserve a whole node like on Nic4. The nature of the fit is explained by the dependence of the number of unknown nodal values as a function of the order of the basis functions: these data points have also been fitted with a second order polynomial.

Finally, the increase in computation time is sharper than the increase in number of unknowns, because increasing the order of the basis functions also involves an increase in the order for the Gauss integration, ensuring an exact integration of Lagrange polynomials when computing the mass matrix, the stiffness matrix, and the vector of fluxes. As quadrangles are obtained by recombining two triangles, the computation time for a mesh made of quadrangles is shorter than for a mesh consisting of triangles.

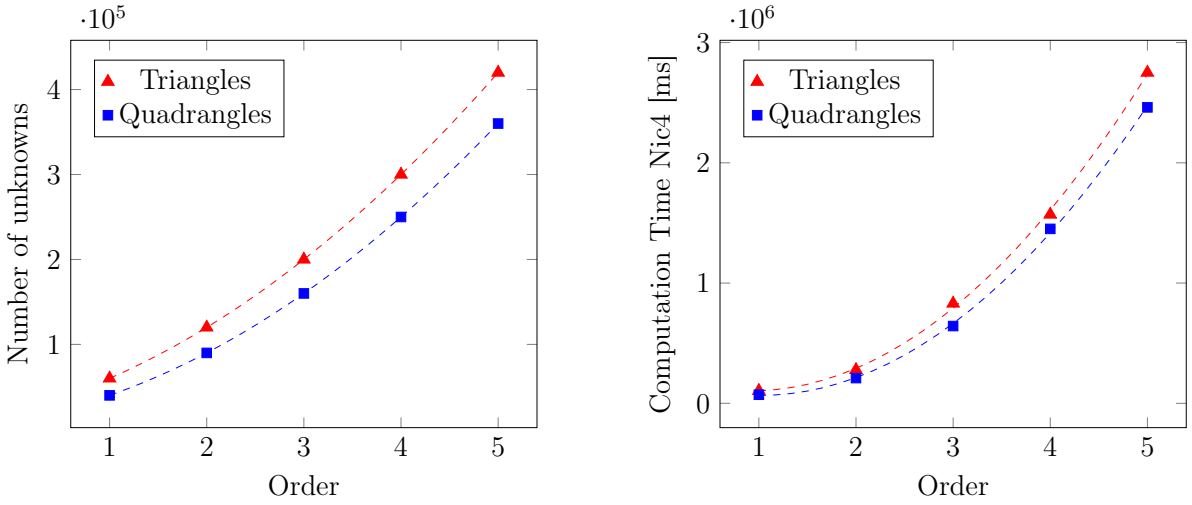


Figure 14: Number of unknowns (left) and computation time (right) as a function of the order of the basis functions.

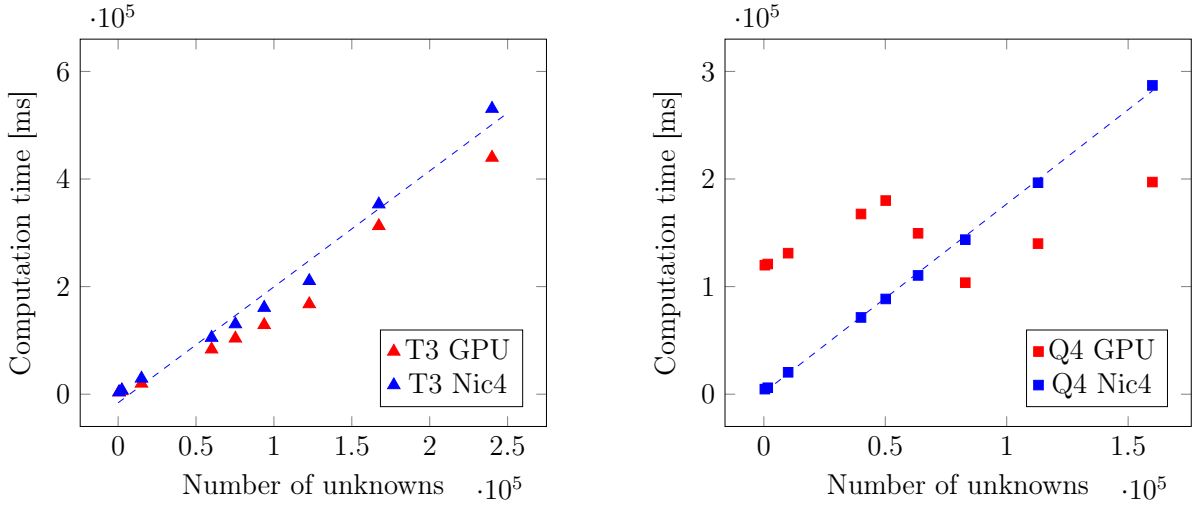


Figure 15: Computation time as a function of the number of nodal unknowns for a mesh made of triangles (left) and quadrangles (right).

Figure 15 shows that the computation time increases linearly with the number of unknown nodal values, both for triangles as for quadrangles. The graph on the right-hand side shows that the computation time on the GPU can be greatly perturbed by the occupancy rate of the device. Nevertheless, the graph on the left-hand side indicates that the efficiency of the GPU increases as the workload becomes more important. The difference in computation time between Nic4 and the GPU grows as the number of unknown nodal values increases.

Finally, the same cavity problem run on Nic4 with 15 threads shows a perfect linear relationship between the computation time and the time step. A similar relationship is recovered on GPU, but the data points are subjected to more variations. Moreover, it may be unexpected that the computation time on GPU is larger than on CPU. In addition to possible programs running on the background which slow down the process, the workload at each time step is maybe too small. Indeed, for a mesh size with triangles of characteristic length 0.01, the number of nodal unknowns is 0.6×10^5, resulting in a small difference in computation time between GPU and Nic4 as shown in Figure 15 for a time step of 10^{-5}.

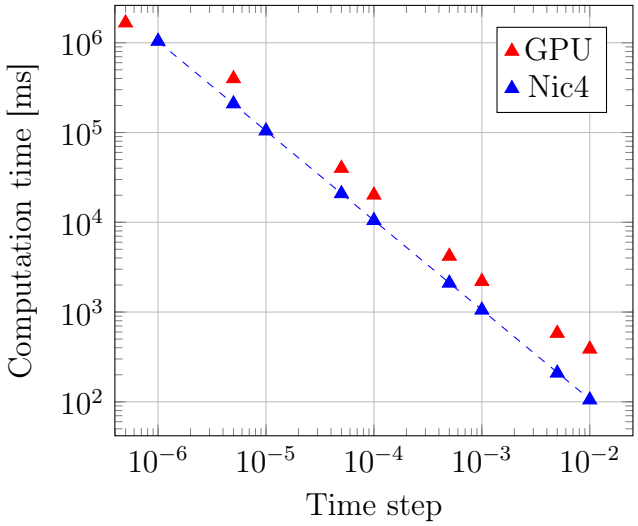


Figure 16: Computation time as a function of time step for the cavity problem with a T3 mesh, run on GPU and on Nic4 (15 threads).

8.6 Waveguide

8.6.1 Theory

A waveguide is a structure that restricts the propagation of waves into a specific direction. Hence, by contrast to spherical waves which decay radially with increasing distance, waveguides allow to focus the energy of the wave into the desired direction. This is particularly useful to transfer information to a specific location. Waveguides concern both electromagnetic waves, such as light, and mechanical waves, like acoustic waves. In most cases, internal reflection inside the structure is the driving principle to propel the wave into the unconfined direction. In this project, we rely on the reflection from perfectly conducting metallic plates. In what follows, we will call a material with infinite conductivity a perfect electrical conductor (PEC). As already mentioned, an electromagnetic wave impinging on this type of surface is entirely reflected.

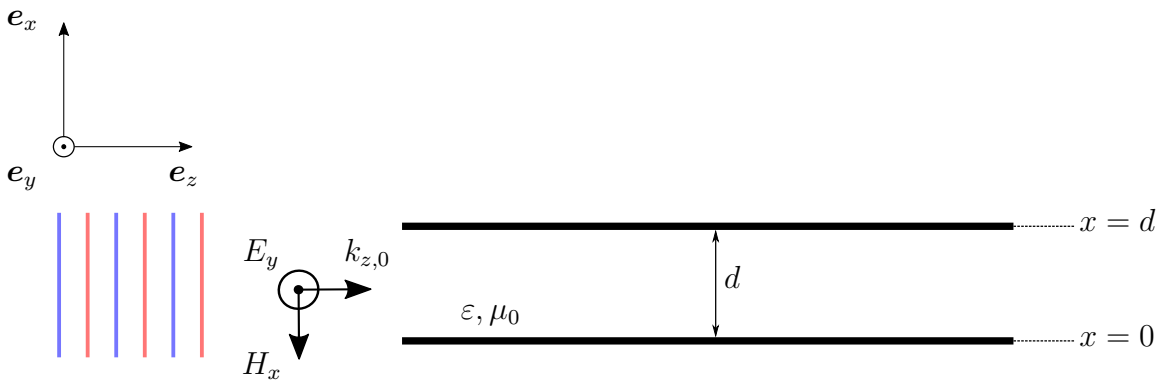


Figure 17: Incoming TE plane wave on a PEC parallel plates waveguide.

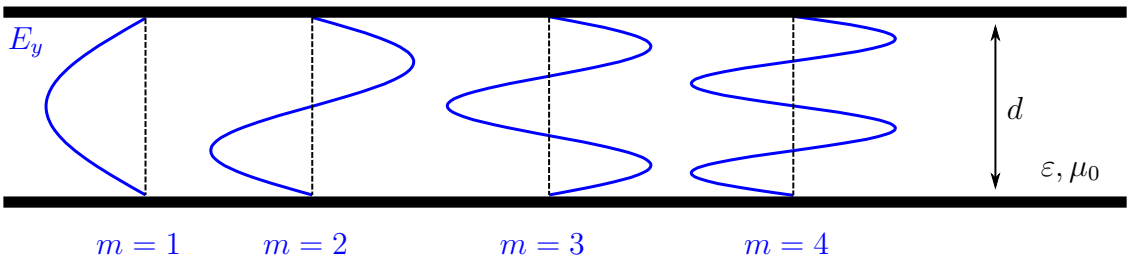


Figure 18: Spatial distribution along x of the $E_{y,m}$ component, for the four first modes of the TE polarization.

Let us now consider a medium with electrical permittivity ε and magnetic permeability μ_0, sandwiched between two infinite PEC plates separated by a distance d, upon which a plane wave impinges. Let us also consider a TE mode, for which z is the direction of propagation, E_y is the only non-zero electric field component, and H_x and H_z are the only non-zero components of the magnetic field. This is illustrated in Figure 17.

Between the plates, the electric field must satisfy the wave equation (58). Furthermore, to satisfy the perfectly conducting boundary conditions, the electric field tangential to the PEC plates must be zero at the plates, i.e.

$$E_y(x = 0, y, z) = E_y(x = d, y, z) = 0. \tag{100}$$

Given the infinite nature of the plate along y and z, their presence can only influence the electric field distribution along x. The electric field, propagating along z, therefore takes the form:

$$\boldsymbol{E} = E_0 F(x) \exp\big(j(\omega t - k_z z)\big)\boldsymbol{e}_y, \tag{101}$$

where $F(x)$ is a function to be determined which satisfies the boundary conditions $F(0) = F(d) = 0$, and k_z is the wave vector along z. Plugging this field into (58), while satisfying the boundary conditions yields:

$$F(x) = \sin(k_{x,m}x), \tag{102}$$

where k_x is the wave vector along x which can only take discrete values according to the so-called guidance condition:

$$k_{x,m} = \frac{m\pi}{d}, \quad m \in \mathbb{Z}_0. \tag{103}$$

Here, the integer m is the index of the mode propagating through the waveguide. It describes the distribution of the fields along x, which are labeled with an index m and are referred to by the TE_m denomination. The corresponding magnetic fields along x and y can be obtained from Equation (49). Finally, the fields for each mode m are given by:

$$E_{y,m} = E_0 \sin(k_{x,m}x) \exp\big(j(\omega t - k_{z,m}z)\big), \tag{104}$$

$$H_{x,m} = -\frac{k_{z,m}}{\omega\mu_0} E_0 \sin(k_{x,m}x) \exp\big(j(\omega t - k_{z,m}z)\big), \tag{105}$$

$$H_{z,m} = \frac{jk_{x,m}}{\omega\mu_0} E_0 \cos(k_{x,m}x) \exp\big(j(\omega t - k_{z,m}z)\big). \tag{106}$$

For the TE mode considered here, $m = 0$ yields a trivial solution with $E_y = 0$ everywhere, and is therefore not acceptable. The four first modes for the electric field are plotted in Figure 18. Each odd-labeled mode is symmetrical around the center of the waveguide ($x = d/2$), while the even-labeled modes are anti-symmetrical.

Plugging (104) in (58) yields the dispersion relation for the TE polarization PEC parallel plates waveguide:

$$k_{z,m} = \sqrt{\omega^2\varepsilon\mu_0 - k_{x,m}^2} = \sqrt{\omega^2\varepsilon\mu_0 - \left(\frac{m\pi}{d}\right)^2}, \quad m \in \mathbb{Z}_0. \tag{107}$$

When propagating through a medium characterized by an electric permittivity ε and a magnetic permeability μ_0, the electromagnetic wave is slowed down and its wavelength λ is compressed by the refractive index $\tilde{n}$ of the medium:

$$\lambda = \frac{\lambda_0}{\tilde{n}}, \tag{108}$$

where λ_0 is the free-space wavelength. Instead of a wave vector $k_{z,m}$, we can associate to each mode m along z an effective refractive index $\tilde{n}_{eff,m}$ by rewriting (107) as follows:

$$\tilde{n}_{eff,m} = \sqrt{\tilde{n}^2 - \left(\frac{m\lambda_0}{2d}\right)^2}, \tag{109}$$

where $\tilde{n}$ is the refractive index of the medium between the parallel plates, and d is the distance between the two plates. From this relation, we can see that $k_{z,m}$ is either purely real or purely imaginary, corresponding respectively to an evanescent or a propagative mode. When a given wave number $k_{z,m}$ is imaginary, the corresponding mode with index m is said to be *cutoff*. For each mode m, we define a cutoff frequency below which the mode does not propagate but is instead evanescent:

$$\omega_{c,m} = \frac{1}{\sqrt{\varepsilon\mu_0}}\frac{m\pi}{d}. \tag{110}$$

Since m cannot be zero for TE modes, light cannot propagate inside the waveguide until its frequency is higher than the cutoff frequency for the TE_1 mode. These results can be summarized and visualized easily with a band structure, plotted in Figure 19. For a fixed free-space wavelength, the effective refractive indices of the five first modes are plotted as a function of the separation distance between the plates d. Below the cutoff distance $d_{c,m}$, a mode m has a purely imaginary refractive index and the associated wave is evanescent. However, as we approach the cutoff distance, the wave has a greater penetration depth. Above the cutoff distance, the refractive index becomes purely real; the wave can propagate. Finally, as the distance between the two plates increases even more, the effective refractive index converges towards $\tilde{n}$, the refractive index of the medium between the plates. This band structure allows to easily visualize which modes can propagate for a fixed parameter of interest.

In summary, light in a waveguide cannot propagate as easily as a plane wave in an infinite medium. Instead, it must adapt to the presence of the parallel plates by modifying its field distribution along x, and it can only propagate according to some particular modes for which there exists a cutoff frequency.

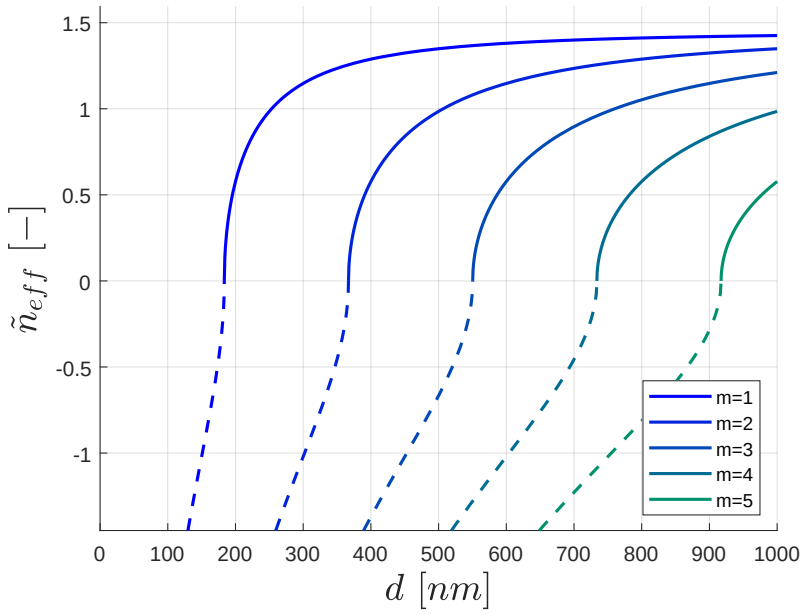


Figure 19: Band structure for the five first modes of the TE polarization in a PEC parallel plates waveguide, for a free-space wavelength $\lambda_0 = 532\,\text{nm}$ and an effective refractive index $\tilde{n} = 1.45$. The solid lines represent the real part of the refractive index, while the dashed lines represent the opposite of the imaginary part.

8.6.2 Numerical implementation

Our 1D waveguide is modeled by two parallel plates separated by a vertical dimensionless distance $y = 1$ (Figure 20). The dimensionless horizontal distance is $x = 6$. To avoid any confusion, it should be noticed that coordinate system used for the simulations is not the same as the one used to describe the theory. As the direction of propagation for the simulations is along the x direction, the only non-zero components of the TE electromagnetic wave are the magnetic fields H_x and H_y, and the electric field E_z. Hence, like for the cavity problem, the following correspondence is used in the numerical code:

$$u_1 = H_x, \quad u_2 = H_y, \quad u_3 = E_z. \tag{111}$$

At the left opening of the waveguide, a wave is excited by imposing an oscillating electric field E_z at a frequency ω, as shown in Figure 20. A perfectly matched layer (PML) is placed slightly before the right opening to simulate an open boundary. The parameters for the waveguide problem are summarized in the **waveguideParam.txt** file. The first one is the frequency of the electric field imposed at the entry of the waveguide:

$$\omega = n\pi, \tag{112}$$

where $n \in \mathrm{R}_0^+$.

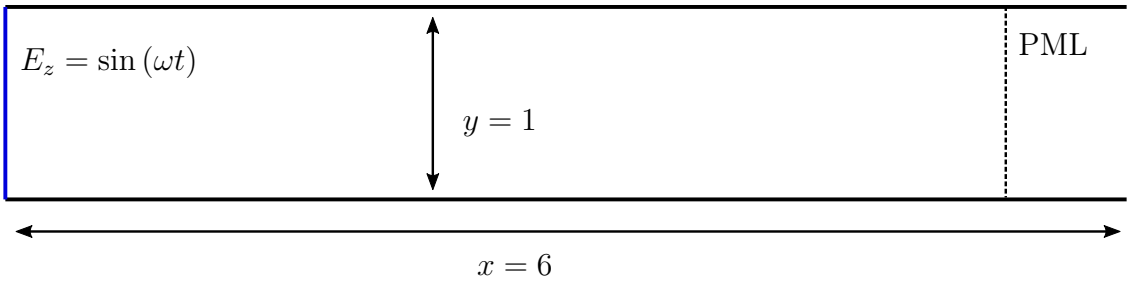


Figure 20: Sketch of the simulated waveguide.

Nevertheless, the simulated solution will be erroneous if the imposed frequency is too large compared to the characteristic mesh size and time step. Therefore, like for the cavity problem in Section 8.5.4, a study of the frequency-dependence on the error of the numerical solution with respect to the analytical solution should be performed.

The other parameters that can be tuned are the flux parameter $0 \leq \alpha \leq 1$ and the PML parameters $x_0 \leq 1$ and σ_0. We recommend to fix these parameters to $\alpha = 1$, $x_0 = 0.9$, and $\sigma_0 = 2500$. The smaller x_0, the more the PML extends into the waveguide. Finally, the geometry of the waveguide can be adapted by tuning $x_{\max}$ and $y_{\max}$ in the **simParam.txt** file. The x dimension should be much larger than the y dimension, as we assumed the perfectly conducting plates to be infinite. Moreover, it should be noticed that the y dimension plays a role in the cutoff frequency, below which waves cannot propagate through the waveguide. The following simulation parameters have been used for the simulations presented below:

$x_{\max}$: 6

$y_{\max}$: 1

mesh size: 0.02

time step: 5e-5

number of time steps: 160000

sampling: 1000

order of basis functions: 1

order for gauss integration: 2

mesh type: 3

To run the code once the **simParam.txt** and **waveguideParam.txt** have been filled in, please type:

```
./bin/main 3
```

to select the waveguide problem.

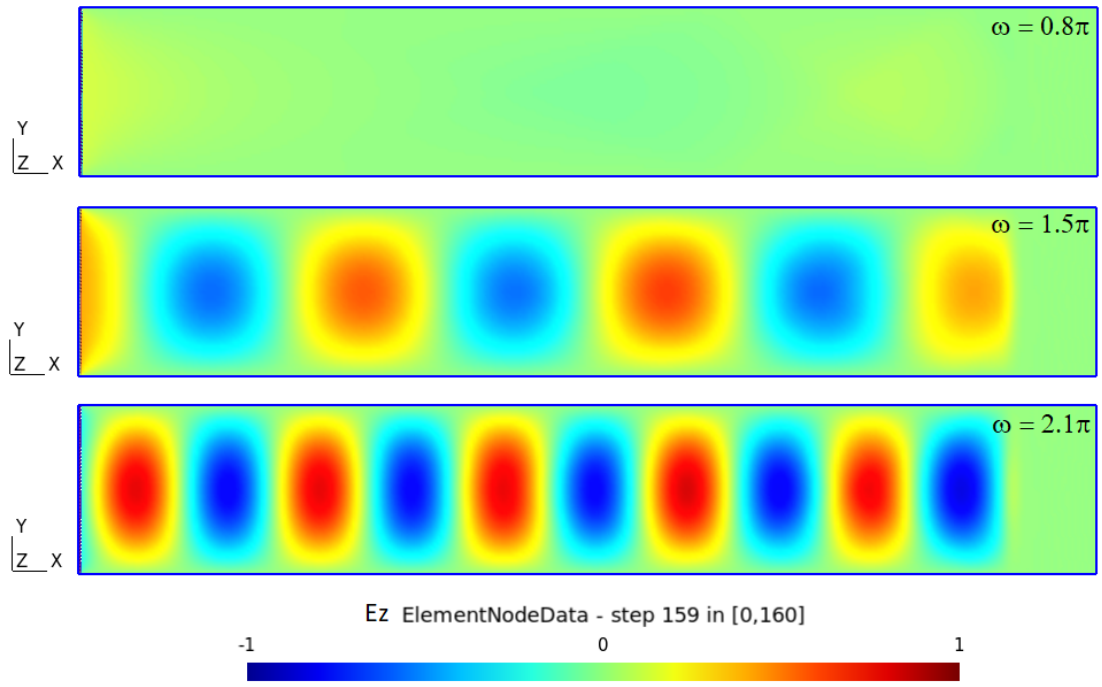


Figure 21: Visualization of the wave propagating in the waveguide, for frequencies ranging from 0.8π to 2.1π.

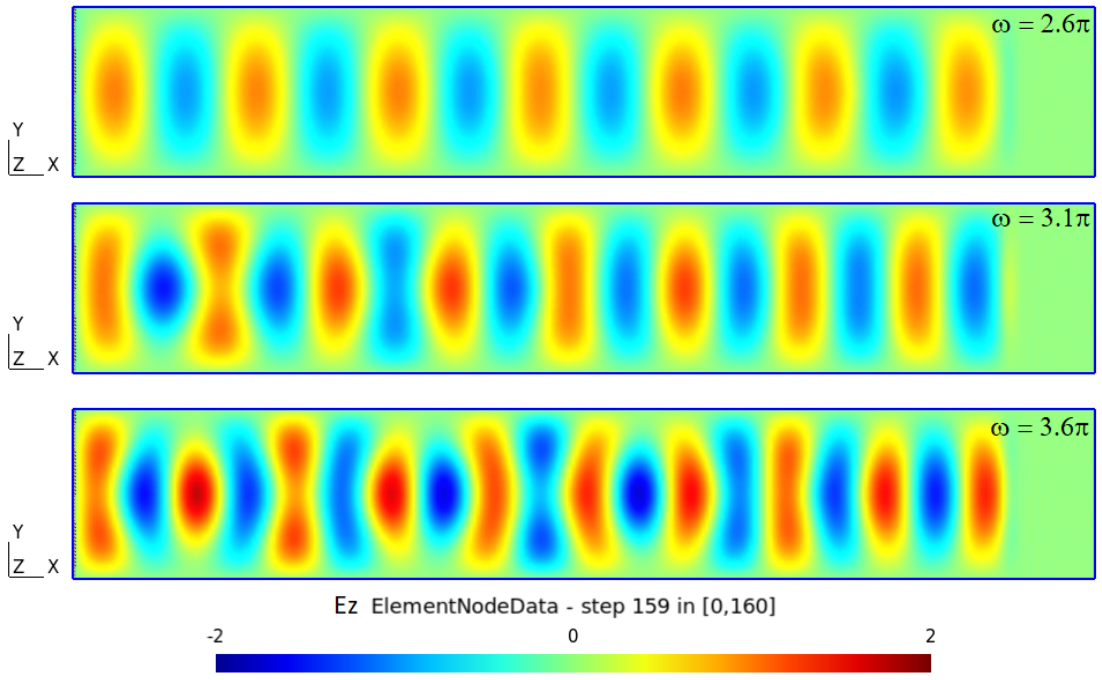


Figure 22: Visualization of the wave propagating in the waveguide, for frequencies ranging from 2.6π to 3.6π.

8.6.3 Propagation modes

Following the results of Section 8.6.1, the wave can only propagate in accordance with the modes to which a cutoff frequency is associated. Below the cutoff frequency, the wave is purely evanescent. Considering that the waveguide is filled with air, for which $\varepsilon \simeq \varepsilon_0$, the expression for the cutoff frequencies (109) in terms of dimensionless units becomes:

$$\omega_{c,m} = \frac{m\pi}{d}, \quad \mathrm{m} \in \mathbb{N}_0. \tag{113}$$

Therefore, for $d = 1$ like this is the case here, and for a frequency below $\omega = \pi$, the wave should be attenuated in the waveguide. Then, for each integer multiple of π, a new mode will become available for the wave to propagate.

A few simulations are run with different frequencies and the results can be visualized in Figures 21 and 22. For $\omega = 0.8\pi$, the wave cannot propagate and is attenuated inside the waveguide. When the frequency becomes greater than π (e.g. 1.5π), the wave can couple into the $m = 1$ mode and propagates through the waveguide with a $\sin(\pi y)$ spatial dependence, according to Equation (104) and Figure 18.

As the frequency is increased above 2π, the $m = 2$ mode is not excited because this mode is asymmetrical, while the incoming wave is a plane wave at normal incidence. The electric field matching at the interface in this situation is very poor, so that light does not couple into the asymmetrical mode, but instead stays in the symmetrical $m = 1$ mode.

When reaching $\omega = 3\pi$, a new mode is activated: a $\sin(3\pi y)$ profile is superimposed on top of the spatial distribution of the first mode. Nevertheless, due to the incoming plane wave at normal incidence, the higher order modes are more difficult to identify than in the fundamental one ($m = 1$ here). Finally, according to (104), the amplitude of the electric field increases with increasing frequency. We also see that the wave is strongly attenuated when the PML on the right is reached. This prevents the wave from being reflected, as it would be the case in a cavity with perfectly conducting walls.

8.6.4 Bent waveguide

As stated in the previous section, the purpose of a waveguide is to guide the electromagnetic energy along a chosen direction. It was shown how a straight waveguide can transport the energy along one dimension, according to the allowed modes of propagation. The goal is now to see how it is possible to deviate the power into another direction by bending the waveguide. A geometry for the associated simulation is sketched in Figure 23. Similarly to the straight waveguide, the parameters for the simulation of the bent waveguide can be modified in the **simParam.txt** and **SwaveguideParam.txt**.

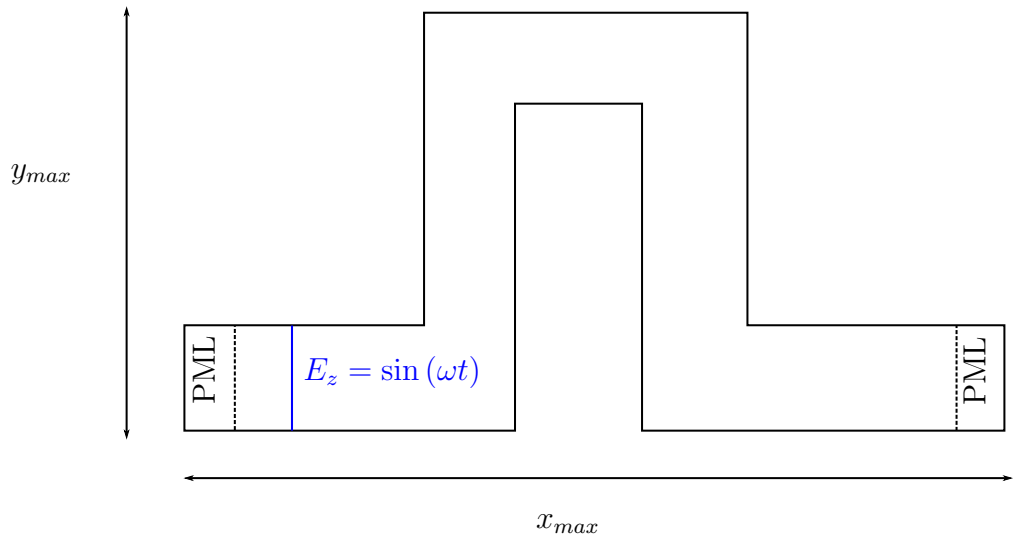


Figure 23: Sketch of the bent waveguide.

The following simulation parameters have been used for the simulations presented below:

$x_{\max}$: 2
$y_{\max}$: 1
mesh size: 0.02
time step: 5e-5
number of time steps: 100000
sampling: 1000
order of basis functions: 1
order for gauss integration: 2
mesh type: 3

To run the the bent waveguide problem, please type:

```
./bin/main 6
```

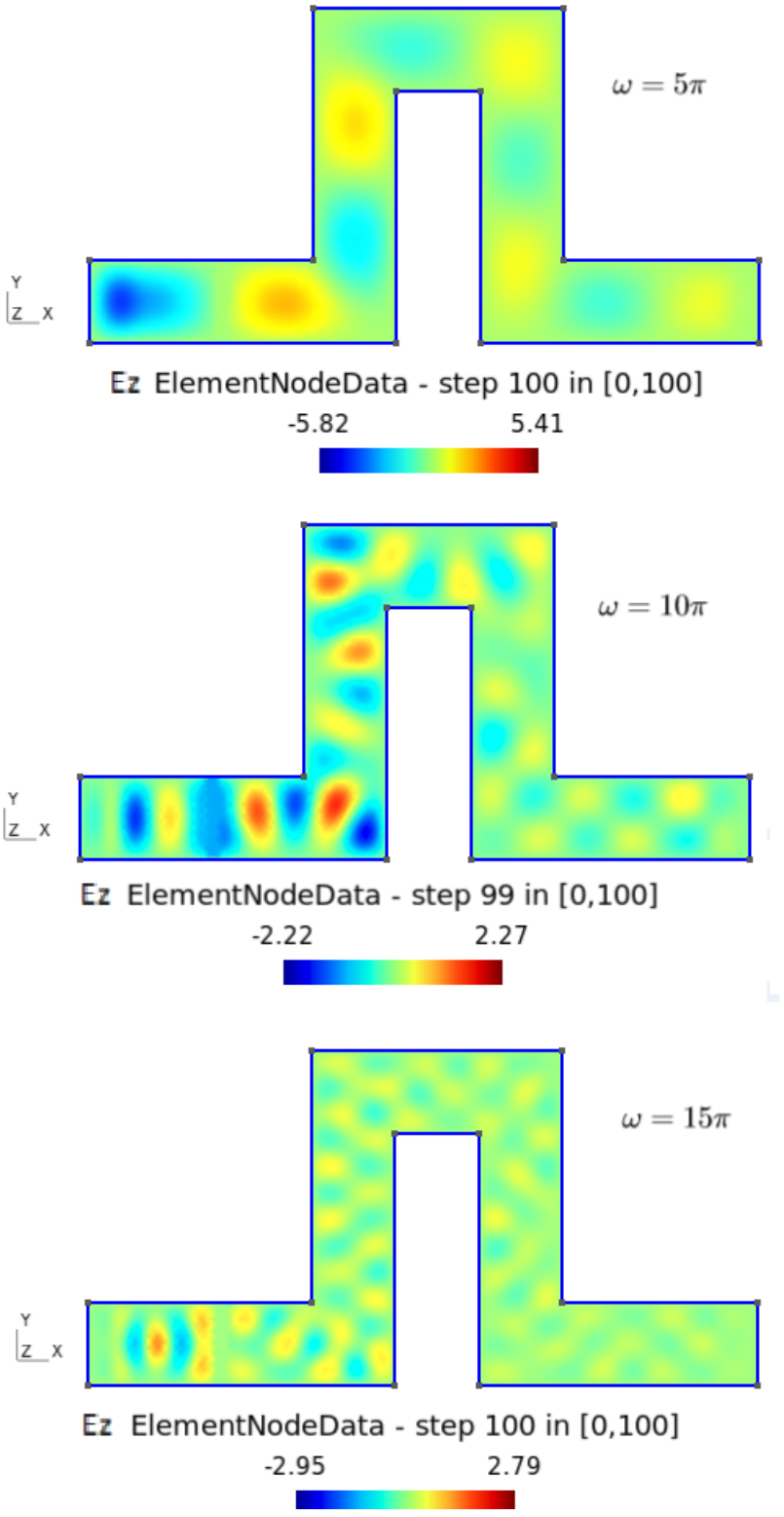


Figure 24: Visualization of the wave propagating in the bent waveguide, for frequencies ranging from 5π to 15π.

The results are plotted in Figure 24 for three different frequencies: $\omega = 5\pi, \omega = 10\pi$ and $\omega = 15\pi$. As can be seen, the wave is able to follow the waveguide's orientation and propagate inside it. As it was observed before, increasing the frequency allows the wave to couple into more modes. For $\omega = 5\pi$, only the first mode is available. When doubling the frequency, we notice that, this time, the asymmetric $m = 2$ mode can be excited in the right side of the waveguide, since the wave no longer comes at a normal incidence. Further increasing the frequency activates the $m = 3$ mode.

8.6.5 Waveguide in a photonic crystal

A photonic crystal is a periodic optical nanostructure that affects the motion of photons in much the same way that ionic lattices affect electrons in solids. For example, it is possible to create a "frequency band gap" by placing periodic pillars with a higher dielectric constant than the surrounding medium, preventing frequencies lying inside the band gap to propagate through the structure.

In this work, however, we are only able to simulate perfect electrical conductors. A guiding effect similar to that of a waveguide can be obtained by removing rows of pillars from a 2D photonic crystal, as illustrated in Figure 25. As the wave cannot propagate between the closely placed pillars, the wave is guided like inside a normal waveguide. The same remarks as for the normal waveguide also seem to hold in this case: different modes can be observed depending on the frequency. The simulation results are shown in Figure 26 and have been obtained by using the same parameters as for the bent waveguide, only the geometry has changed. To run the code once the **simParam.txt** and **SwaveguideParam.txt** have been filled in, please type:

```
./bin/main 7
```

to select the photonic crystal waveguide problem.

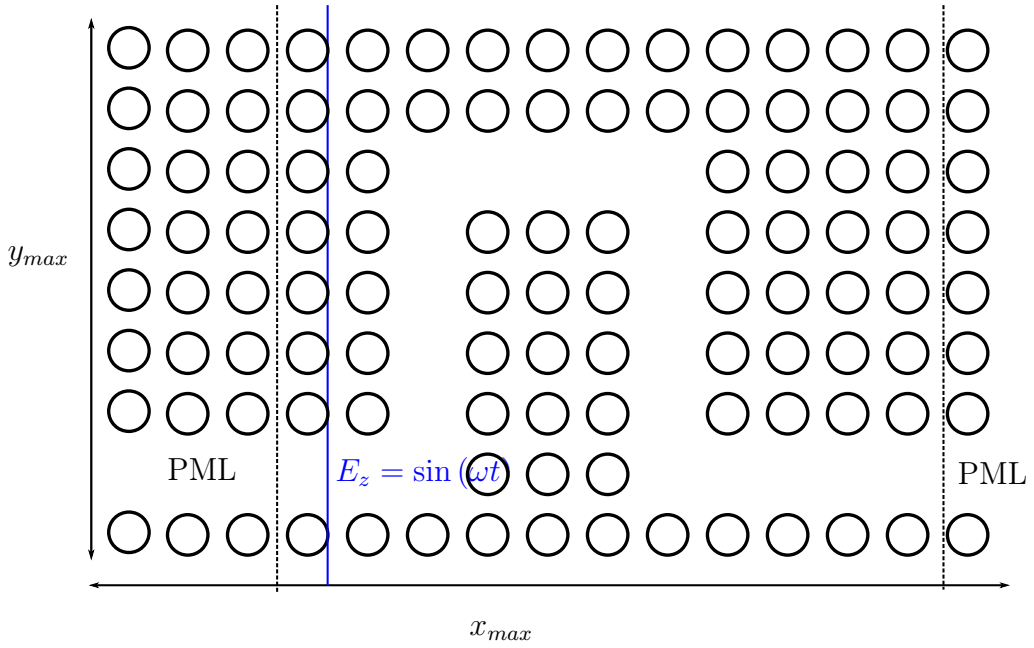


Figure 25: Sketch of the waveguide carved in a crystal.

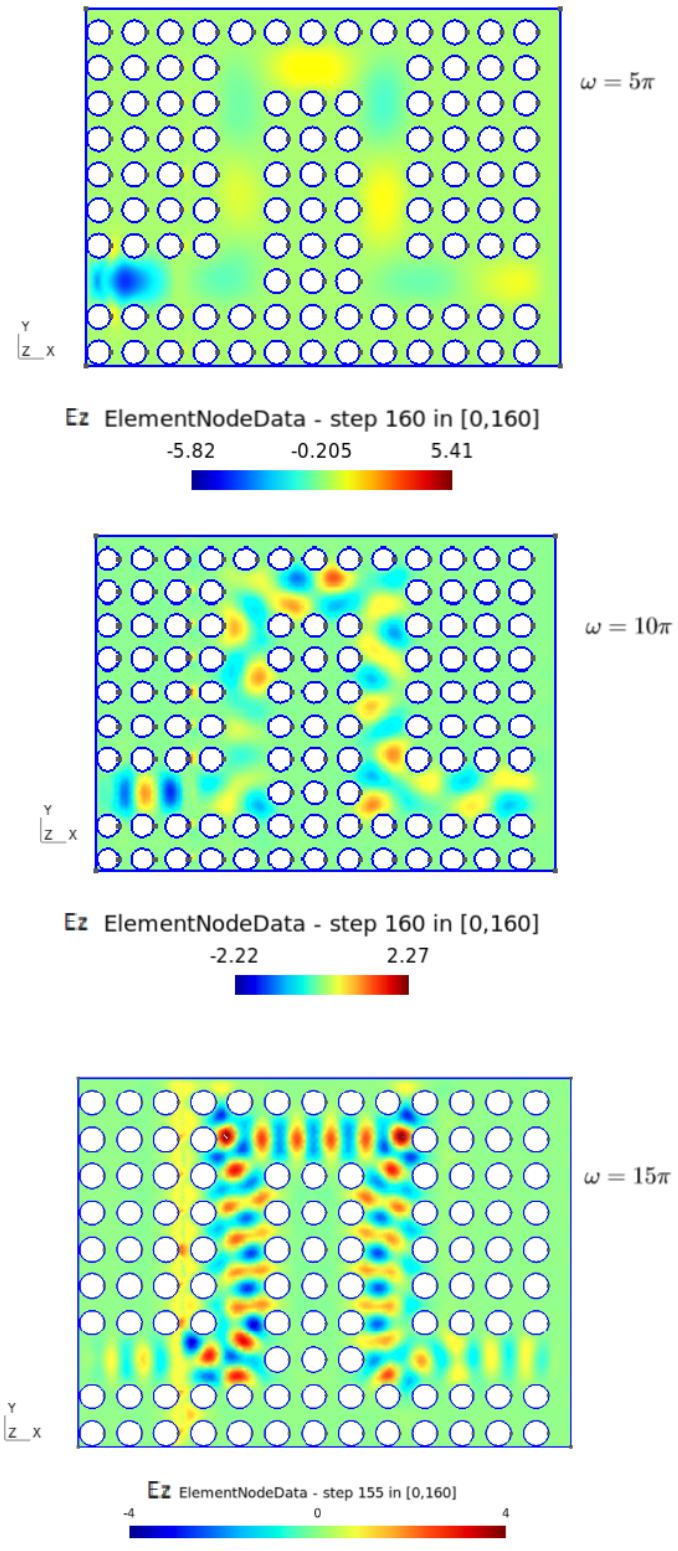


Figure 26: Visualization of the wave propagating in the bent waveguide, for frequencies ranging from 5π to 15π.

8.7 Diffraction grating

8.7.1 Theory

When light is incident on a surface with irregular profile at a length scale comparable to the wavelength, it is reflected and refracted in many different directions. If the irregularities are periodic, constructive interferences will produce replicas of the incident and transmitted beams in different directions (Figure 28).

More precisely, when incident light impinges upon a grating (Figure 27), the boundary conditions require the tangential component of the incoming wave vector $k_{x,i}$ to be continuous. Upon entering a periodic grating, the phase matching condition is

$$k_{x,m} = k_{x,i} + mK, \quad m \in \mathbb{Z}, \tag{114}$$

where the subscript i stands for incident, m is an integer, and K is the wave number of the grating. Using Snell's identity, we obtain the so-called grating equation:

$$n_{r,t} \sin(\theta_m) = n_i \sin(\theta_i) + m\frac{\lambda_0}{\Lambda}, \quad m \in \mathbb{Z}, \tag{115}$$

where n is the refractive index of the medium, the subscripts r, t, i stand respectively for reflection, transmission and incident, λ_0 is the free space wavelength, and $\Lambda = 2\pi/K$ is the periodicity of the grating. Constructive interferences occur at an angle θ_m when the grating equation is satisfied, leading to replicas of the incident beam in those directions defined by θ_m. It is possible to derive a dispersion relation similar to those for the parallel plates waveguide from the phase matching condition (114):

$$k_{z,m} = \sqrt{k_0^2 n_{r,t}^2 - (k_0 n_i \sin(\theta_i) + mK)^2}, \quad m \in \mathbb{Z}, \tag{116}$$

or, when normalized to the incident wave number $k_0 = 2\pi/\lambda_0$:

$$\frac{k_{z,m}}{k_0} = \sqrt{n_{r,t}^2 - \left(n_i \sin(\theta_i) + \frac{m\lambda_0}{\Lambda}\right)^2}, \quad m \in \mathbb{Z}. \tag{117}$$

Similarly to the waveguide, for each diffraction mode m, there exists a cutoff pitch $\Lambda_{c,m}$ below which the wave number is imaginary and for which the corresponding diffracted mode is evanescent:

$$\Lambda_{c,m} = \begin{cases} \dfrac{\lambda_0 m}{(n_{r,t} - n_i \sin(\theta_i))} & \text{if } m > 0, \\ \dfrac{\lambda_0 |m|}{(n_{r,t} + n_i \sin(\theta_i))} & \text{if } m < 0. \end{cases} \tag{118}$$

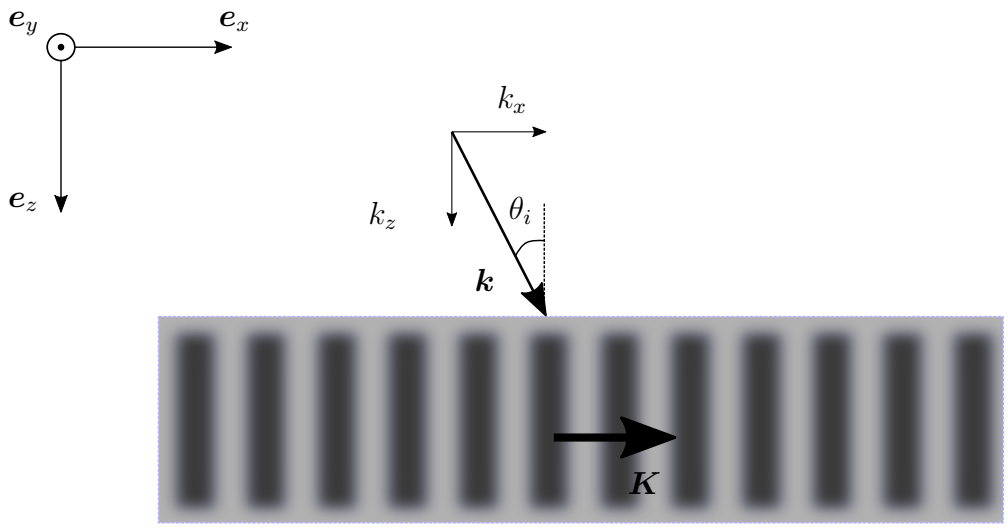


Figure 27: Phase matching condition on the tangential component of the incoming wave vector incident on a periodic grating with wave vector $\boldsymbol{K}$.

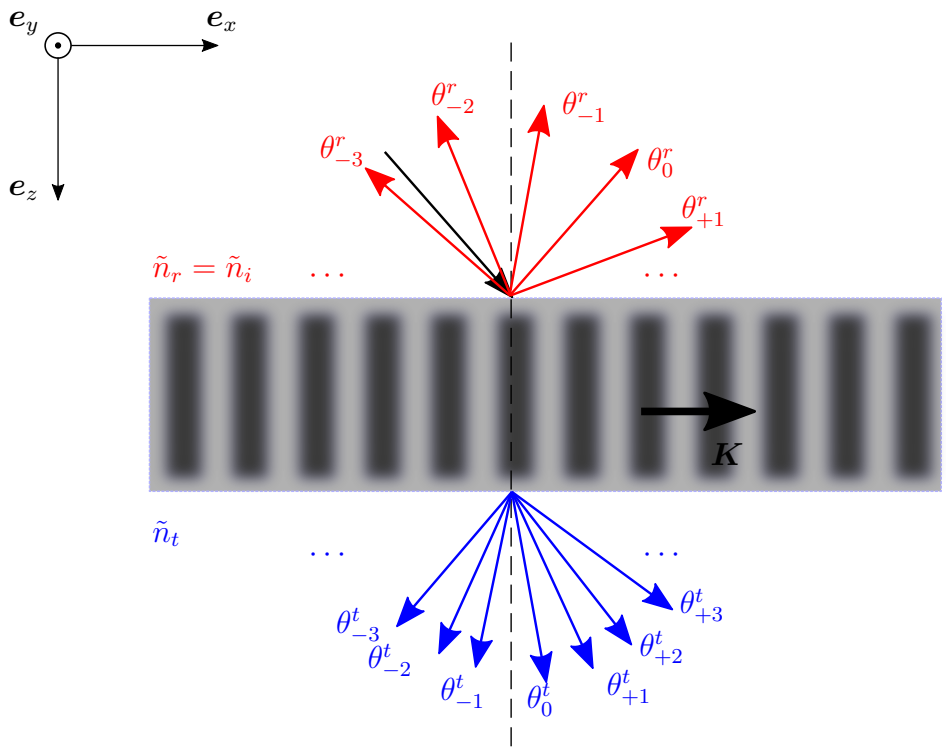


Figure 28: Example of diffraction orders for an incoming plane wave with incidence angle θ_i. In this case, $n_t > n_i$ and more modes are available in transmission.

According to this relationship, propagation modes become available more quickly in gratings with a higher refractive index. Besides, by contrast to the waveguide problem, the $m = 0$ mode always exists. In this particular case, there is no diffraction: the incident beam behaves according to Snell's laws of reflection and refraction. It is also possible to plot a band structure for the diffraction modes. For a given set of parameters n_i, n_t, θ_i and λ_0, it is then easy to visualize which modes are available for propagation as a function of the pitch Λ. Given a mode m, it is then possible to infer the angle θ_m at which the corresponding mode propagates.

8.7.2 Numerical implementation

The goal of the grating problem is to simulate the behavior of a plane wave impinging on a periodic perfectly conducting metal grid. The periodic metal lines, shown in Figure 29 can be used both as waveguides and diffraction objects. Depending on the frequency of the incident wave, they should be able to guide waves and/or diffract light in different directions.

The medium before, in between, and after the grating is assumed to be air. Therefore, the refractive index is 1 and the conditions for diffraction in reflection or transmission are the same. As depicted in Figure 29, the period of the grating Λ is 0.1, while the width dy of the metal lines is 0.04. The separation distance d in each waveguide is therefore $\Lambda - dy$. An oscillating electric field $E_z = \sin(\omega t)$ is imposed before the grating, and the resulting plane wave impinges at a normal incidence on the structure. PML's are defined at the borders of the domain to simulate open boundaries.[2]

The parameters for the grating problem are reported in the **gratingParam.txt** file. The first parameter is $0 \leq a \leq 0.4$, allowing to tune the distance $d = 2a$ from the left boundary at which the electric field is imposed. The second one is the frequency of the electric field $\omega = n\pi$, where $n \in \mathrm{R}_0^+$. The others are the flux parameter $0 \leq \alpha \leq 1$ and the PML parameters $x_0 \leq 1$, $y_0 \leq 1$, and σ_0. We recommend to fix these parameters to $a = 0.2$, $\alpha = 1$, $x_0 = 0.9$, $y_0 = 0.9$, and $\sigma_0 = 750$. The smaller x_0 and y_0, the more the PML's extend into the domain. Lowering σ_0 compared to the value chosen for the waveguide problem allows to reduce oscillations in the left corners.

Finally, the geometry of the diffraction grating is fixed, because according to (118) the grating should have specific dimensions in order to be able to observe diffraction. There is no need to change $x_{\max}$ and $y_{\max}$ in the **simParam.txt** file, as they will automatically be replaced by $x_{\max} = 2$ and $y_{\max} = 1$.

[2]The ideal case would be to impose periodic boundary conditions to simulate an ideal, infinite grating but this is out of the scope of this project.

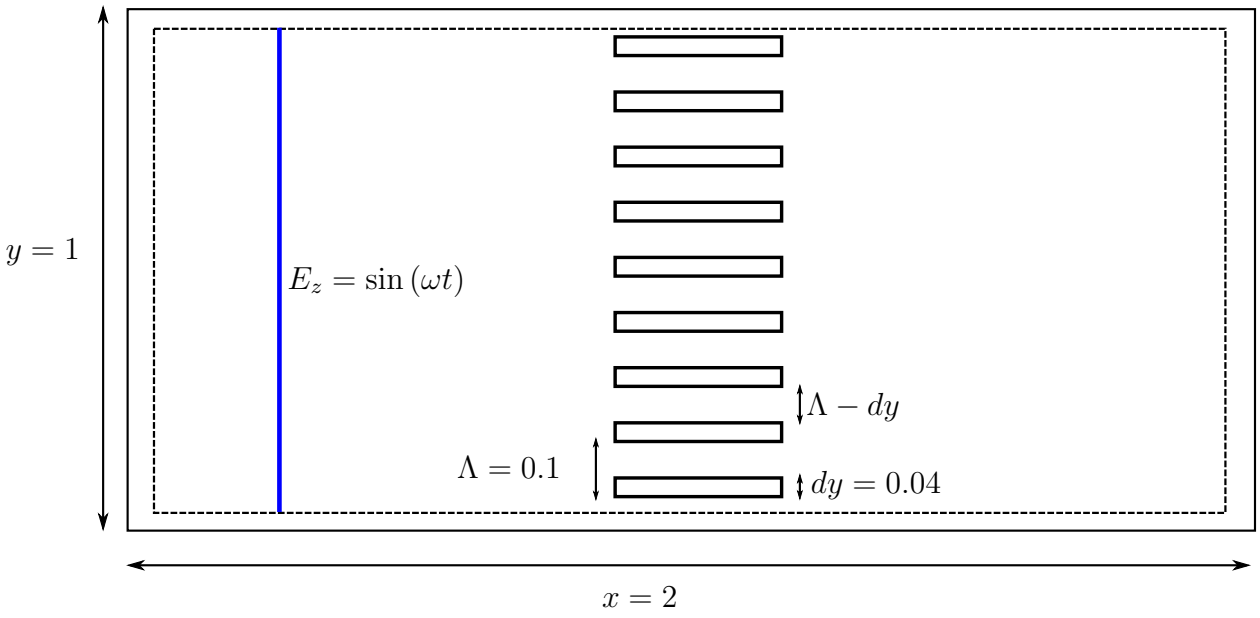


Figure 29: Schematics for the simulation of periodic PEC waveguides. The area exterior to the dashed lines is a Perfectly Matched Layer to simulate open boundaries.

The following simulation parameters have been used for the simulations presented below:

mesh size: 0.02

time step: 5e-5

number of time steps: 149000

sampling: 1000

order of basis functions: 1

order for gauss integration: 2

mesh type: 3

To run the code once the **simParam.txt** and **gratingParam.txt** have been fixed, please type:

```
./bin/main 4
```

to select the grating problem.

8.7.3 Frequency regimes

The cutoff frequencies for the waveguides and the diffraction modes (reflection/transmission) can be deduced from (113) and (117). Remembering that

$$\frac{c}{n} = \frac{\omega}{k}, \tag{119}$$

considering that all media are air ($n = 1$) and that the plane wave has a normal incidence ($\theta_i = 0$), and in accordance to the dimensionless units used so far, we find that the cutoff frequencies for the waveguides and the diffraction modes respectively write:

$$\omega_{c,\mathrm{w},m} = \frac{m\pi}{\Lambda - dy}, \qquad m \in \mathbb{N}_0, \tag{120}$$

$$\omega_{c,\mathrm{d},p} = \frac{2p\pi}{\Lambda}, \qquad p \in \mathbb{Z}. \tag{121}$$

From those results, different regimes can be identified as a function of the frequency:

- A regime for which the frequency is lower than the cutoff frequency of both the first waveguide mode and the first diffraction mode. In this situation there should only be a reflected plane wave, because the wave entering the waveguide is attenuated.
- A regime for which the frequency is greater than the cutoff frequency of the first waveguide mode, but smaller than that of the first diffraction mode. In this case there should be a reflected and transmitted plane wave.
- A regime for which the frequency is greater than the cutoff frequency of both the first waveguide mode and the first diffraction mode. In this regime, a diffraction pattern should be observed for both the reflected and transmitted wave.

By setting the frequencies of the incident plane wave to 15π, 18π, and 21π, these three regimes can be observed. Figure 30 shows that the wave cannot penetrate into the waveguide when $\omega = 15\pi$, because its frequency is lower than the cutoff frequency of the waveguide $\omega_{c,\mathrm{w},1}$. Indeed, regarding our geometry, $\omega_{c,\mathrm{w},1} = 16.66\pi$. Moreover, the reflected wave is a plane wave of the zeroth order as the frequency is also too low for diffraction to occur ($\omega_{c,\mathrm{d},1} = 20\pi$).

For $\omega = 18\pi$, the wave is transmitted through the waveguide as its frequency exceeds $\omega_{c,w,1}$. Nevertheless, this frequency is still too low to observe a clear diffraction pattern on both sides of the waveguide. The diffracted mode is evanescent. There is a non-negligible diffraction close to the grating, but not in the far field. Far from the grating, the waves in reflection and transmission correspond to the zeroth order of diffraction.

Finally, when the frequency exceeds $\omega_{c,\mathrm{d},1} = 20\pi$, the incident plane wave is both transmitted through the waveguide and diffracted by the grating. This is visible from the interference patterns before and after the grating, where the plane waves of the zeroth order, traveling along x at normal incidence, interfere with the plane waves of order 1 and -1, which travel at oblique angles.

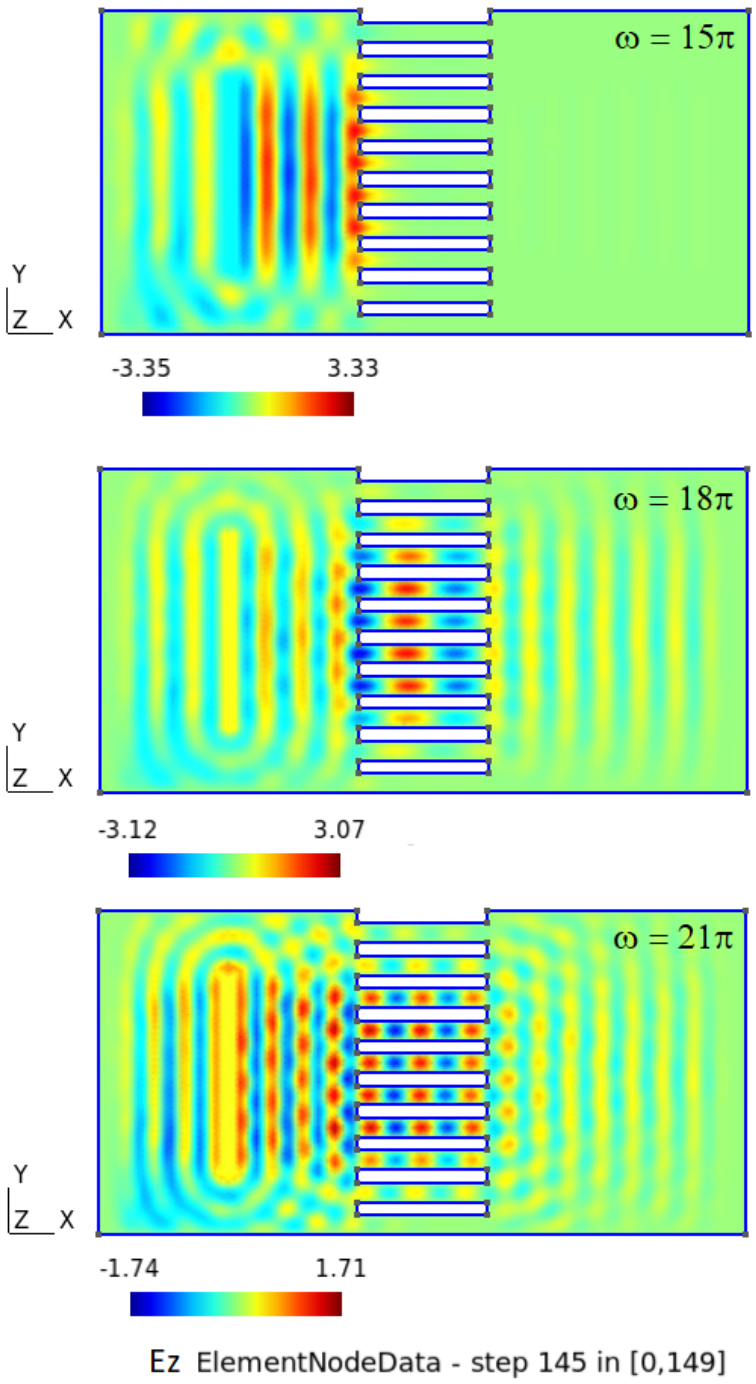


Figure 30: Electric field for the grating problem. Different dimensionless frequencies ω have been set for the incident plane wave (15π, 18π, 21π) to show the three different regime.

9 Conclusion and perspectives

The implementation of a numerical code based on the discontinous Galerkin method allowed us to model complex physical phenomena. We started by representing the advection of a gaussian function to get familiar with this powerful simulation method. The convergence and stability of the scheme were tested and approved the correct implementation of the code. Then, the code was parallelized to increase the performances. This was done on a shared memory CPU on the Nic4 cluster and on a GPU accelerator. The time spent adapting the code to be accelerated on a GPU proved to be worthwhile, as we reached very high speedups.

As soon as we found the right expression for the physical fluxes, the reflective boundary conditions, and the perfectly matched layers, it was impressive how fast we could adapt the DG scheme to Maxwell's equation, as the core of the method was the same as for the advection problem. The parallelization on GPU allowed for testing problems with more complex geometries and longer simulation times. The convergence of the numerical solution with respect to the analytical solution for the waveguides has not been verified, but we recovered the expected physical phenomena.

The DG scheme proved to be a very powerful finite element method, but our code still has a some limitations. Our code is limited to 2D problems, but Gmsh provides the features needed to expand the code to 3D geometries. Moreover, the power of the DG method is to deal with curved geometries and elements. Nevertheless, the normals computed in our code are only good for straight edges. For curved edges, a different normal should be computed at each integration point, whereas we only calculated a normal for a whole edge.

The purpose of the advection code was to serve as an introduction to the DG method. Therefore, the boundary conditions are limited to constants. It may be interesting to model open boundaries, so that discontinuities at the boundaries are avoided when the gaussian is transported through the domain.

Regarding the propagation of electromagnetic waves, the only media that we modeled are either vacuum/air or perfect electrical conductors. This is because the boundary conditions to model a PEC are straightforward. Extending the code to media with a finite conductivity, different permittivities, and different permeabilities would allow more physical phenomena to be simulated. We think, for example, of designing a photonic band gap or light refraction at an interface. It may also be interesting to convert the electric and magnetic field back to their SI units, as we only dealt with dimensionless quantities.

Although the stability of the scheme as a function of α for Maxwell's equation has not been studied, we noticed that using $\alpha = 1$ was a lot more stable than $\alpha = 0$. A more comprehensive study on this topic could be conducted. Finally, since the forward Euler method is an explicit scheme, the time step is strongly constrained by the mesh size due to the CFL condition. Developing an implicit scheme decreases the constraints on the simulation parameters, but it is more challenging to implement and it significantly increases the computation time, as a system of equations needs to be solved at each time step.

References

[1] Maarten Arnst. *Lecture notes of "Modeling with partial differential equations"*. Université de Liège. Sept. 2019.

[2] Tim Warburton and Jan S. Hesthaven. *Nodal Discontinuous Galerkin Methods*. Springer, 2008.

[3] Benjamin Loret. *Lecture notes about "Classification of partial differential equations into elliptic, parabolic and hyperbolic types"*. Consulted on 22 Feb. 2020 and available at `http://people.3sr-grenoble.fr/users/bloret/enseee/maths/enseee-maths-IBVPs-3.pdf`.

[4] Jean-Philippe Ponthot. *Lecture notes of "Introduction to the finite element method"*. Université de Liège. 2019.

[5] C. Geuzaine and J.-F. Remacle. *Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities*. `http://gmsh.info`. International Journal for Numerical Methods in Engineering 79(11), pp. 1309–1331, 2009.