Unconditionally stable and parallel Discontinuous Galerkin solver

Pierre Gerhard, Philippe Helluy, Victor Michel-Dansac

UNCONDITIONALLY STABLE AND PARALLEL DISCONTINUOUS GALERKIN SOLVER

PIERRE GERHARD [*,†], PHILIPPE HELLUY [*,†], AND VICTOR MICHEL-DANSAC [†,*]

Abstract. We describe a parallel and quasi-explicit Discontinuous Galerkin (DG) kinetic scheme for solving systems of balance laws. The solver is unconditionally stable (i.e., the CFL number can be arbitrary) and has the complexity of an explicit scheme. It can be applied to any hyperbolic system of balance laws, see [2, 21]. In this work, we assess the performance of the scheme in the particular cases of the three-dimensional wave equation and of Maxwell's equations. We measure the benefit of the unconditional stability by performing experiments with very large CFL numbers. In addition, the parallel possibilities of the method are investigated.

1. Introduction

The Discontinuous Galerkin (DG) method is an increasingly popular method for solving hyperbolic systems of balance laws. Initially designed for solving neutron transport equations [42, 38], it has been progressively applied to a wide range of linear or non-linear wave models. Now, many books have been written on the subject. We refer, for instance, to [33, 41, 24, 18]. This method is especially useful for numerical simulations in complex geometries, because it allows completely unstructured grids. It is a natural generalization of the finite volume method for high-order approximations. It is well adapted to linear wave models, such as Maxwell's equations [10, 29] or seismic models [36]. It also works for non-linear hyperbolic equations, but requires limiting tools to handle shock waves in a robust and accurate way [16]. The method is also well adapted to parallel software optimizations and hybrid computing (see for instance [36, 37, 11]).

Of course, the method also has some drawbacks. Namely, it contains more unknowns and is in general more computationally expensive than its finite differences or finite volumes counterparts. However, its main issue lies in the correct determination of the time integration, and more specifically the dependence of the time step on the spatial mesh.

In this paper, we tackle this delicate point of time integration. Usually, the DG space approximation is coupled with an explicit Runge-Kutta time integration. This kind of time integration imposes quite constraining CFL conditions [13, 44]. Indeed, the time step size is then constrained by the smallest cell size in the mesh, as long as the numerical solution changes slowly in time. This often results in inflexible restrictions, since the smallest cell size may be imposed by geometric constraints rather than by the required numerical precision. In addition, the possibility to coarsen the mesh to compute low-frequency solutions can be limited by an economic aspect: meshing a complex geometry is time-consuming and it is generally unfeasible, in real-world applications, to generate several meshes with various refinement levels. In principle, it would be possible to adopt an implicit time integration scheme, but it requires the inversion of a linear system at each time step and is thus very computationally expensive.

In this work, we propose a DG solver that is free from both restrictive CFL stability conditions and expensive linear system inversions. The method is based on an abstract kinetic representation of the system of balance laws introduced by Bouchut [8] and Aregba-Natalini [2]. This is a general method described for example in [25, 21]. After writing this abstract kinetic representation, we use a splitting method to separate the transport and relaxation parts, which are treated distinctly.

The objective of this paper is to apply this general principle with a minimal set of kinetic velocities in order to reduce the memory footprint of the method. This methodology yields a generic scheme, which can be applied to any balance law, such as Maxwell's equations or the linear wave equations, which are used for numerical experiments.

The paper is organized as follows. First, in section 2, we introduce the minimal vectorial kinetic representation of balance laws that is used throughout this paper. Then, in section 3, we present in detail the algorithm, based on the Discontinuous Galerkin framework that we develop to solve the transport part of the kinetic

*: IRMA, Université de Strasbourg, CNRS UMR 7501, 7 rue René Descartes, 67084 Strasbourg, France
†: Université de Strasbourg, CNRS, Inria, IRMA, F-67000 Strasbourg, France
Key words and phrases. discontinuous Galerkin, kinetic approximation, CFL-less, unconditional stability, parallelization.

equations. There, we also describe software optimizations that have been performed to parallelize the method. Afterwards, the relaxation part is treated in section 4, where we also give the full second-order accurate in time discretization. A short comment on the unconditional stability, and hence the CFL-less aspects of the method, is proposed in section 5. In section 6, the method is applied to several academic and physically relevant test cases, to provide a comparison, in terms of accuracy and efficiency, with the usual explicit DG method. Performance and parallelism tests are carried out in section 7. Finally, section 8 contains a short conclusion as well as perspectives for future work.

2. Minimal vectorial kinetic representation of balance laws

In this section, we recall the theoretical framework of the vectorial kinetic representation that our numerical scheme is based on. After introducing notations for hyperbolic systems in section 2.1, we recall the vectorial kinetic representation itself in section 2.2. Section 2.3 explains the over-relaxation needed to use this representation in practice, and section 2.4 analyzes the effect of this over-relaxation on the initial hyperbolic system. Finally, in section 2.5, we discuss how the kinetic representation behaves when adding a source term to the hyperbolic system under consideration.

2.1. Hyperbolic systems of conservation laws.

In this work, we are interested in the numerical resolution of a hyperbolic system. The vector of unknowns is denoted $\mathbf{u}(\mathbf{x},t) \in \mathbb{R}^m$. It depends on time t and the space variable $\mathbf{x} = (x^1, \dots, x^d) \in \mathbb{R}^d$, where d is the space dimension (in practice, $d = 1, 2$ or 3). We denote by ∂_i the derivative with respect to x^i. The general form of the system is

$$\partial_t \mathbf{u} + \sum_{i=1}^{d} \partial_i \mathbf{q}^i(\mathbf{u}) = 0. \tag{2.1}$$

The flux functions $\mathbf{q}^i : \mathbb{R}^m \to \mathbb{R}^m$ are supposed to be smooth. For $\mathbf{n} = (n_1, \dots, n_d) \in \mathbb{R}^d$, we define the physical flux in the direction $\mathbf{n}$ by

$$\mathbf{q}(\mathbf{u}, \mathbf{n}) = \sum_{i=1}^{d} \mathbf{q}^i(\mathbf{u}) n_i.$$

We assume that the system satisfies the hyperbolicity property, i.e. that the Jacobian matrix $\boldsymbol{\nabla}_{\mathbf{u}} \mathbf{q}(\mathbf{u}, \mathbf{n})$ is diagonalizable with real eigenvalues for all $\mathbf{n} \in \mathbb{R}^d$ and $\mathbf{u}$ in the hyperbolicity domain $\mathcal{C}$, which is supposed to be a convex set.

2.2. Vectorial kinetic representation.

We now recall how to construct a minimal kinetic representation of (2.1).

For one-dimensional systems, with $d = 1$, this construction reduces to the Jin-Xin relaxation [34].

The general case for $d \geq 1$ is studied in [8, 2, 21]. We only recall here the simplest case, where the number of kinetic velocities is $d + 1$. We consider a set of constant vectors

$$\mathcal{V} = \left\{ \mathbf{v}_k = (v_k^1, \dots, v_k^d)^T \in \mathbb{R}^d, \ k \in \{0, \dots, d\} \right\}, \tag{2.2}$$

called the kinetic velocities. We suppose that the rank of $\mathcal{V}$ is maximal (it must be equal to d). To each kinetic velocity $\mathbf{v}_k$, we associate a kinetic unknown $\mathbf{f}_k(\mathbf{x}, t) \in \mathbb{R}^m$. The kinetic unknowns and the macroscopic data $\mathbf{u}$ are related by

$$\sum_{k=0}^{d} \mathbf{f}_k = \mathbf{u}. \tag{2.3}$$

We then consider the following system of kinetic equations

$$\partial_t \mathbf{f}_k + \mathbf{v}_k \cdot \boldsymbol{\nabla} \mathbf{f}_k = \frac{1}{\tau} (\mathbf{m}_k(\mathbf{u}) - \mathbf{f}_k), \tag{2.4}$$

where $\mathbf{m}_k(\mathbf{u})$ are the equilibrium kinetic functions, to be determined, and where τ is a small positive fixed parameter and represents a relaxation time.

This system is similar to the BGK approximation in the kinetic theory of gases. In (2.4), the equilibrium kinetic functions $\mathbf{m}_k(\mathbf{u})$ play the role of the Maxwellian state of the kinetic theory of gases. The main differences are that, in this context, the number of kinetic velocities is finite and the kinetic unknown is a vector instead of a scalar function.

Let us now compute the equilibrium kinetic functions $\mathbf{m}_k(\mathbf{u})$. Summing the kinetic equations (2.4) on k, we obtain

$$\partial_t \mathbf{u} + \sum_{i=1}^{d} \partial_i \left(\sum_{k=0}^{d} v_k^i \cdot \mathbf{f}_k \right) = \frac{1}{\tau} \left(\sum_{k=0}^{d} \mathbf{m}_k(\mathbf{u}) - \mathbf{u} \right). \tag{2.5}$$

Formally, when $\tau \to 0$, in (2.4), we have $\mathbf{f}_k \simeq \mathbf{m}_k(\mathbf{u})$. If we assume that

$$\sum_{k=0}^{d} \mathbf{m}_k(\mathbf{u}) = \mathbf{u},$$

then, when $\tau \to 0$, (2.5) becomes

$$\partial_t \mathbf{u} + \sum_{i=1}^{d} \partial_i \left(\sum_{k=0}^{d} v_k^i \cdot \mathbf{m}_k(\mathbf{u}) \right) = 0. \tag{2.6}$$

In conclusion, when $\tau \to 0$, the kinetic system (2.4) is equivalent to the initial hyperbolic system (2.1) provided that

$$\sum_{k=0}^{d} \mathbf{m}_k(\mathbf{u}) = \mathbf{u} \quad \text{and} \quad \forall i \in \{1, \dots, d\}, \ \sum_{k=0}^{d} v_k^i \cdot \mathbf{m}_k(\mathbf{u}) = \mathbf{q}^i(\mathbf{u}). \tag{2.7}$$

Note that (2.7) is nothing but a set of $m(d+1)$ linear equations, whose $m(d+1)$ unknowns are the components of the $d+1$ Maxwellian vectors $\mathbf{m}_k(\mathbf{u})$. This linear system can also be written in matrix form

$$\big(\mathbf{m}_0(\mathbf{u}) \quad \cdots \quad \mathbf{m}_d(\mathbf{u})\big)\mathbf{V} = \big(\mathbf{u} \quad \mathbf{q}^1(\mathbf{u}) \quad \cdots \quad \mathbf{q}^d(\mathbf{u})\big),$$

with

$$\mathbf{V} = \begin{pmatrix} 1 & v_0^1 & \cdots & v_0^d \\ 1 & v_1^1 & \cdots & v_1^d \\ \vdots & \vdots & & \vdots \\ 1 & v_d^1 & \cdots & v_d^d \end{pmatrix} = \begin{pmatrix} 1 & \mathbf{v}_0^T \\ 1 & \mathbf{v}_1^T \\ \vdots & \vdots \\ 1 & \mathbf{v}_d^T \end{pmatrix}.$$

We assume that $\mathbf{V}$ is invertible and we obtain the Maxwellian state from the macroscopic data $\mathbf{u}$ and the physical flux $\mathbf{q}$ as follows:

$$\big(\mathbf{m}_0(\mathbf{u}) \quad \cdots \quad \mathbf{m}_d(\mathbf{u})\big) = \big(\mathbf{u} \quad \mathbf{q}^1(\mathbf{u}) \quad \cdots \quad \mathbf{q}^d(\mathbf{u})\big)\mathbf{V}^{-1}. \tag{2.8}$$

Let us remark that once the Maxwellian state is obtained, the physical flux is given by

$$\mathbf{q}(\mathbf{u}, \mathbf{n}) = \sum_{k=0}^{d} (\mathbf{v}_k \cdot \mathbf{n})\, \mathbf{m}_k(\mathbf{u}).$$

Now that $\mathbf{m}_k(\mathbf{u})$ is known, we explain how to solve the equation (2.4) in practice.

2.3. **Splitting and over-relaxation.** In (2.4), the BGK source term

$$\frac{1}{\tau}(\mathbf{m}_k(\mathbf{u}) - \mathbf{f}_k)$$

is of theoretical interest, but it is not very useful in practice because it couples the $d+1$ kinetic equations in a non-linear way.

In practice, the kinetic system (2.4) is replaced by

$$\partial_t \mathbf{f}_k + \mathbf{v}_k \cdot \boldsymbol{\nabla} \mathbf{f}_k = \boldsymbol{\mu}_k, \tag{2.9}$$

where the source term $\boldsymbol{\mu}_k$ is designed in such a way to obtain $\mathbf{f}_k \simeq \mathbf{m}_k$ but is not a BGK source term. We introduce a time step $\Delta t > 0$ and the Dirac comb:

$$\varphi(t) = \sum_{j \in \mathbb{Z}} \delta(t - j\Delta t).$$

The source term $\boldsymbol{\mu}_k$ is then defined by

$$\boldsymbol{\mu}_k(x,t) = \boldsymbol{\Omega}(\mathbf{u})\, \varphi(t) \left(\mathbf{m}_k(\mathbf{u}(x,t^-)) - \mathbf{f}_k(x,t^-) \right), \tag{2.10}$$

where the so-called relaxation matrix $\boldsymbol{\Omega}$ is such that $\boldsymbol{\Omega} = \theta \mathbf{I}$, where $1 \le \theta \le 2$ and with $\mathbf{I}$ the identity matrix. From the distribution theory, we see that at time $t = i\Delta t$, $\mathbf{f}_k$ is discontinuous: $\mathbf{f}_k(x,t^+) \neq \mathbf{f}_k(x,t^-)$, and

$$\mathbf{f}_k(x,t^+) = \boldsymbol{\Omega}\, \mathbf{m}_k(\mathbf{u}(x,t)) + (\mathbf{I} - \boldsymbol{\Omega})\, \mathbf{f}_k(x,t^-). \tag{2.11}$$

If $\theta = 1$ and the relaxation matrix is $\mathbf{\Omega} = \mathbf{I}$, we recover the classical first-order Jin-Xin splitting algorithm, where $\mathbf{m}_k(\mathbf{u}) = \mathbf{f}_k$ at the end of each time step. The *over-relaxation* corresponds to $\mathbf{\Omega} = 2\mathbf{I}$. It can be proven that the resulting scheme is a second-order in time approximation of (2.1), up to $\mathcal{O}(\Delta t^2)$. For more details, the reader is referred for instance to [21, 25] and included references.

We see that most of the time, the kinetic variables $\mathbf{f}_k$ satisfy free transport equations at velocity $\mathbf{v}_k$, with relaxation to equilibrium at each time step.

2.4. **Equivalent equation.** In this section, we analyze the effect of the kinetic over-relaxation on the initial variables $\mathbf{u}$ of the hyperbolic system (2.1). In order to get a more compact writing, we set

$$\mathbf{q}^0(\mathbf{u}) = \mathbf{u}, \quad \mathbf{Q}(\mathbf{u}) = \begin{pmatrix} \mathbf{q}^0(\mathbf{u}) & \mathbf{q}^1(\mathbf{u}) & \cdots & \mathbf{q}^d(\mathbf{u}) \end{pmatrix},$$

and

$$\mathbf{M}(\mathbf{u}) = \begin{pmatrix} \mathbf{m}_0(\mathbf{u}) & \cdots & \mathbf{m}_d(\mathbf{u}) \end{pmatrix}.$$

With these definitions, (2.8) becomes

$$\mathbf{Q}(\mathbf{u}) = \mathbf{M}(\mathbf{u})\mathbf{V}. \tag{2.12}$$

We also define the vector of kinetic variables

$$\mathbf{F} = \begin{pmatrix} \mathbf{f}_0 & \cdots & \mathbf{f}_d \end{pmatrix}. \tag{2.13}$$

Let us introduce the approximate fluxes $\mathbf{Z}$, defined by

$$\mathbf{Z} = \mathbf{F}\mathbf{V}, \quad \text{i.e. } \begin{pmatrix} \mathbf{z}^0 & \mathbf{z}^1 & \cdots & \mathbf{z}^d \end{pmatrix} = \begin{pmatrix} \mathbf{f}_0 & \cdots & \mathbf{f}_d \end{pmatrix}\mathbf{V}. \tag{2.14}$$

Let us emphasize that $\mathbf{z}^0 = \mathbf{u}$. We also introduce the flux errors $\mathbf{Y} = \begin{pmatrix} \mathbf{y}^0 & \cdots & \mathbf{y}^d \end{pmatrix}$, defined by

$$\mathbf{Y} = \mathbf{Z} - \mathbf{Q}(\mathbf{u}), \quad \text{i.e. } \forall i \in \{0, \dots, d\},\ \mathbf{y}^i = \mathbf{z}^i - \mathbf{q}^i(\mathbf{u}). \tag{2.15}$$

Plugging (2.12) and (2.14) into (2.15), we find that

$$\mathbf{Y} = (\mathbf{F} - \mathbf{M}(\mathbf{u}))\mathbf{V}.$$

Let us insist that, with our definition, while $\mathbf{Y}$ has $d+1$ columns, it lives in a $(m \times d)-$dimensional space since

$$\mathbf{y}^0 = 0.$$

With these notations, we can rewrite the transport-relaxation algorithm as follows. We start with a macroscopic field $\mathbf{u}(\mathbf{x}, 0)$ and flux error field $\mathbf{Y}(\mathbf{x}, 0)$ at time $t = 0$. From this flux error, we compute the approximate fluxes

$$\mathbf{Z} = \mathbf{Y} + \mathbf{Q}(\mathbf{u})$$

and the kinetic variables

$$\mathbf{F} = \mathbf{Z}\mathbf{V}^{-1}.$$

Then, the kinetic variables $\mathbf{F}$ are transported with velocities $\mathcal{V}$ during a time step Δt, as follows:

$$\forall k \in \{0, \dots, d\},\ \mathbf{f}_k(\mathbf{x}, \Delta t) = \mathbf{f}_k(\mathbf{x} - \Delta t \mathbf{v}_k, 0). \tag{2.16}$$

This defines the transport operator $\mathcal{T}(\Delta t)$ acting on the initial field $\mathbf{x} \mapsto \mathbf{F}(\mathbf{x}, 0)$. More precisely, $\mathcal{T}$ is unambiguously defined by (2.13), (2.16), and

$$(\mathcal{T}(\Delta t)\mathbf{F}(\cdot, 0))(\mathbf{x}) = \mathbf{F}(\mathbf{x}, \Delta t).$$

Finally, we return to the $(\mathbf{u}, \mathbf{Y})$ variables by computing

$$\mathbf{u} = \mathbf{F} \cdot \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \quad \text{and} \quad \mathbf{Y} = \mathbf{F}\mathbf{V} - \mathbf{Q}(\mathbf{u}).$$

The transport step can thus be expressed in the following operator form

$$\mathbf{u}(\cdot, \Delta t) = \left\{\mathcal{T}(\Delta t)\left[\left(\mathbf{Y}(\cdot, 0) + \mathbf{Q}(\mathbf{u}(\cdot, 0))\right)\mathbf{V}^{-1}\right]\right\} \cdot \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

$$\mathbf{Y}(\cdot, \Delta t^-) = \left\{\mathcal{T}(\Delta t)\left[\left(\mathbf{Y}(\cdot, 0) + \mathbf{Q}(\mathbf{u}(\cdot, 0))\right)\mathbf{V}^{-1}\right]\right\}\mathbf{V} - \mathbf{Q}(\mathbf{u}(\cdot, \Delta t)).$$

In this $(\mathbf{u}, \mathbf{Y})$ set of variables, the over-relaxation step simply reads

$$\mathbf{Y}(\cdot, \Delta t^+) = -\mathbf{Y}(\cdot, \Delta t^-).$$

The equivalent equation allows to better understand the effect of the relaxation matrix $\mathbf{\Omega}$. Through Taylor expansions in time, it is possible to obtain the partial differential equations satisfied by the pair $(\mathbf{u}, \mathbf{Y})$. Details can be found in [25]. Similar works using similar techniques can also be found in [26, 20].

For a simpler exposition, we only give the equivalent equation for a particular set of kinetic velocities and in a two-dimensional context, i.e. we take $d = 2$. In this context, we have $m = d + 1 = 3$ kinetic velocities. We choose the following kinetic velocities:

$$\mathbf{v}_0 = \lambda \begin{pmatrix} \cos(0) \\ \sin(0) \end{pmatrix}, \quad \mathbf{v}_1 = \lambda \begin{pmatrix} \cos(2\pi/3) \\ \sin(2\pi/3) \end{pmatrix}, \quad \mathbf{v}_2 = \lambda \begin{pmatrix} \cos(4\pi/3) \\ \sin(4\pi/3) \end{pmatrix},$$

where λ is a positive scaling factor. Then it can be proven that $\mathbf{u}$ and $\mathbf{Y}$ satisfy the following PDE up to second-order terms in Δt:

$$\begin{aligned} \partial_t \begin{pmatrix} \mathbf{u} \\ \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} + \begin{pmatrix} \mathbf{A}^1(\mathbf{u}) & 0 & 0 \\ 0 & \frac{\lambda}{2}\mathbf{I} - \mathbf{A}^1(\mathbf{u}) & 0 \\ 0 & -\mathbf{A}^2(\mathbf{u}) & -\frac{\lambda}{2} \end{pmatrix} \partial_{x_1} \begin{pmatrix} \mathbf{u} \\ \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} \\ + \begin{pmatrix} \mathbf{A}^2(\mathbf{u}) & 0 & 0 \\ 0 & 0 & -\frac{\lambda}{2}\mathbf{I} - \mathbf{A}^1(\mathbf{u}) \\ & -\frac{\lambda}{2} & -\mathbf{A}^2(\mathbf{u}) \end{pmatrix} \partial_{x_2} \begin{pmatrix} \mathbf{u} \\ \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} = \mathcal{O}(\Delta t^2), \end{aligned} \tag{2.17}$$

where we have set

$$\mathbf{A}^i(\mathbf{u}) = \nabla_{\mathbf{u}} \mathbf{q}^i(\mathbf{u}).$$

We note that this analysis recovers the expected equation on $\mathbf{u}$ in the first line of (2.17). In addition, it also provides a system of PDEs satisfied by $\mathbf{Y}$ in the next two lines. In [25], it is shown in a similar case that the equivalent equation (2.17) is a hyperbolic system under the condition that the scaling factor λ is large enough. In the literature, this kind of condition is often called a sub-characteristic condition. It was first introduced by Whitham in [45]. It is then encountered in numerous works using the relaxation approach. We refer for instance to [34, 19, 8, 5] and included references.

2.5. **Source term treatment.** Finally, we discuss how to treat a source term added to the hyperbolic system (2.1), as follows:

$$\partial_t \mathbf{u} + \sum_{i=1}^{d} \partial_i \mathbf{q}^i(\mathbf{u}) = \mathbf{s}(\mathbf{u}), \tag{2.18}$$

where the source term $s : \mathbb{R}^d \to \mathbb{R}^d$ is assumed to be smooth.

The kinetic system (2.4) then becomes:

$$\partial_t \mathbf{f}_k + \mathbf{v}_k \cdot \boldsymbol{\nabla} \mathbf{f}_k = \mathbf{g}_k(\mathbf{u}) + \frac{1}{\tau}(\mathbf{m}_k(\mathbf{u}) - \mathbf{f}_k), \tag{2.19}$$

where $\mathbf{g}_k(\mathbf{u})$ is a kinetic representation of the source term $\mathbf{s}(\mathbf{u})$. In the framework of section 2.3, we write

$$\partial_t \mathbf{f}_k + \mathbf{v}_k \cdot \boldsymbol{\nabla} \mathbf{f}_k = \mathbf{g}_k(\mathbf{u}) + \boldsymbol{\mu}_k,$$

instead of (2.9), where $\boldsymbol{\mu}_k$ is still given by (2.10).

Recall the definition (2.3) of the kinetic variables $\mathbf{f}_k$ and the conditions (2.7) on the equilibrium kinetic functions $\mathbf{m}_k(\mathbf{u})$. Equipped with these relations, summing (2.19) for $k \in \{0, \dots, d\}$ and formally taking $\tau \to 0$ leads to

$$\partial_t \mathbf{u} + \sum_{i=1}^{d} \partial_i \mathbf{q}^i(\mathbf{u}) = \sum_{k=0}^{d} \mathbf{g}_k(\mathbf{u}) \tag{2.20}$$

instead of (2.6). Therefore, the kinetic representation $\mathbf{g}_k(\mathbf{u})$ of the source term has to satisfy

$$\sum_{k=0}^{d} \mathbf{g}_k(\mathbf{u}) = \mathbf{s}(\mathbf{u}) \tag{2.21}$$

in order for (2.20) to coincide with (2.18).

As explained in [21], a Taylor expansion of the kinetic system performed in [2] shows that the following expression of $\mathbf{g}_k(\mathbf{u})$ ensures some consistency:

$$\mathbf{g}_k(\mathbf{u}) = \nabla_{\mathbf{u}} \mathbf{m}_k(\mathbf{u})\, \mathbf{s}(\mathbf{u}).$$

Indeed, since $\sum_k \nabla_{\mathbf{u}} \mathbf{m}_k(\mathbf{u})$ is nothing but the identity matrix, this expression of $\mathbf{g}_k(\mathbf{u})$ satisfies (2.21).

3. Parallel solvers for the transport step

In section 2, we have provided a vectorial kinetic representation (2.19) of the hyperbolic balance law (2.18). In practice, we split this equation into its transport part

$$\partial_t \mathbf{f}_k + \mathbf{v}_k \cdot \boldsymbol{\nabla} \mathbf{f}_k = 0, \tag{3.1}$$

and its relaxation-source part

$$\partial_t \mathbf{f}_k = \mathbf{g}_k(\mathbf{u}) + \frac{1}{\tau}(\mathbf{m}_k(\mathbf{u}) - \mathbf{f}_k). \tag{3.2}$$

The goal of this section is to propose an efficient algorithm to solve the transport part (3.1). The relaxation-source part (3.2) is treated in section 4.1.

To that end, we rely on the Discontinuous Galerkin (DG) method, see for instance [33]. It is applied to the kinetic variables $\mathbf{F}$, and we end up solving $d+1$ transport equations with constant velocity. Since this solver is the elementary brick of the whole algorithm, it is crucial to provide an efficient parallel implementation. Such an implementation was introduced in several previous works [4, 21].

We describe in this section how to apply the DG solver to the following generic transport equation with a constant velocity $\mathbf{v}$, of unknown $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}$:

$$\partial_t f + \mathbf{v} \cdot \boldsymbol{\nabla} f = 0. \tag{3.3}$$

This equation represents the transport part (3.1) of the equation (2.4), with simplified notation.

In order to avoid restrictive CFL conditions stemming from a few small tetrahedra in the unstructured mesh, we only consider implicit DG solvers. First, we recall in section 3.1 the implicit DG formulation, and we give more details on the computation of the time update on a single tetrahedral element. We emphasize that this implicit formulation ensures the unconditional L^2 stability of the resulting scheme. However, even though the scheme as a whole is implicit, it is possible to sweep the mesh in the direction of the velocity vector to enable a quasi-explicit resolution of (3.3). This procedure is explained in section 3.2, where we also discuss the possibility to parallelize this algorithm to increase its computational efficiency. We also give a faster but more memory-intensive way to solve the DG system in section 3.3. Finally, the three algorithms are summarized in section 3.4 for the sake of convenience. For more details on our implementation of the Discontinuous Galerkin method, we refer to [4]. We also refer to [12] for technical algorithmic aspects of the task-graph algorithm.

3.1. DG formulation on a three-dimensional unstructured tetrahedral mesh.

For more details on the DG method, we refer (for instance) to [33]. The objective is to solve the transport equation with constant velocity (3.3). We assume that $d = 3$, i.e. we consider three space dimensions.

We consider a space domain $\mathcal{D} \subset \mathbb{R}^3$, meshed with a three-dimensional unstructured mesh $\mathcal{M}$ made of n_t tetrahedral cells with ten nodes (the four nodes of the tetrahedra and the six nodes at the edge midpoints), also known as "T10" elements in the finite element literature. In each cell L, we consider ten polynomial basis functions φ_i^L of degree $p = 2$. For efficiency reasons, the basis functions are Lagrange polynomials based on the ten nodes of the tetrahedra. For the sake of completeness, these basis functions are given on the reference tetrahedron in Appendix A. Here, we use this third-order nodal discretization for the sake of simplicity in the presentation. Using higher-order nodal discretizations would result in a more accurate – but more complex – scheme, yet the forthcoming derivation and connectivity computations would be mostly unchanged. Moving to non-nodal discretizations would mean rewriting the DG scheme, but the main properties of the method would remain unchanged.

In each cell in the mesh $\mathcal{M}$, the unknown function f is approximated with the basis functions, as follows:

$$\forall L \in \mathcal{M},\ \forall \mathbf{x} \in L,\ \forall t \geq 0,\ f(\mathbf{x}, t) \simeq f_L(\mathbf{x}, t) = \sum_{i=0}^{9} f_{L,i}(t) \varphi_i^L(\mathbf{x}). \tag{3.4}$$

Since the basis functions are polynomials of degree $p = 2$ in $\mathbf{x}$, this approximation is also a polynomial of degree $p = 2$, and the order of accuracy in space is expected to be $p+1 = 3$. In the formalism described in this section, the scheme is only first-order accurate in time. In practice, we actually use a second-order implicit Crank-Nicolson time stepping, which is very similar to the forthcoming description, but which we omit for the sake of clarity. Now, the goal is to provide a relevant approximation of the coefficients $f_{L,i}(t)$ of f_L in the polynomial basis. The procedure is fairly standard; however, we recall its main lines in Appendix B for the

sake of completeness. The DG solver for (3.3) then reads, with mesh notations from figure 3.1:

$$\begin{aligned}\sum_{i=0}^{9} \frac{f_{L,i}^{n+1} - f_{L,i}^{n}}{\Delta t} \left(\int_L \varphi_i^L(\mathbf{x}) \varphi_j^L(\mathbf{x})\, d\mathbf{x} \right) - \sum_{i=0}^{9} f_{L,i}^{n+1} \left(\int_L \varphi_i^L(\mathbf{x})\, \mathbf{v} \cdot \boldsymbol{\nabla} \varphi_j^L(\mathbf{x})\, d\mathbf{x} \right) \\ + \sum_{i=0}^{9} f_{L,i}^{n+1} \left[\sum_{\alpha=0}^{3} \left(\int_{\partial L_\alpha} \varphi_i^L(\boldsymbol{\eta}) \varphi_j^L(\boldsymbol{\eta})\, d\boldsymbol{\eta} \right) (\mathbf{v} \cdot \mathbf{n}_\alpha)_+ \right] \\ + \sum_{i=0}^{9} f_{R_\alpha,i}^{n+1} \left[\sum_{\alpha=0}^{3} \left(\int_{\partial L_\alpha} \varphi_i^{R_\alpha}(\boldsymbol{\eta}) \varphi_j^L(\boldsymbol{\eta})\, d\boldsymbol{\eta} \right) (\mathbf{v} \cdot \mathbf{n}_\alpha)_- \right] = 0.\end{aligned} \tag{3.5}$$

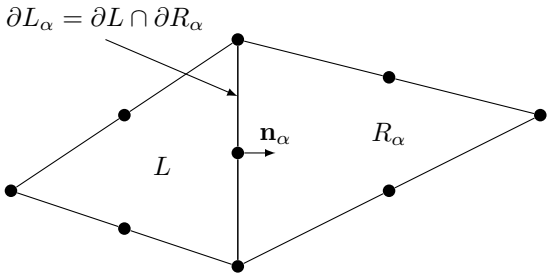


Figure 3.1. Notations for cells, interfaces and normal vectors. Given a cell L, its faces are numbered $\alpha \in \{0, \dots, 3\}$ and denoted by ∂L_α. The neighbor of L through the face ∂L_α is denoted by R_α, and the outer normal vector pointing from L to R_α is denoted by $\mathbf{n}_\alpha$. This figure is in two dimensions for the sake of simplicity, but the actual meshes we are interested in are unstructured three-dimensional tetrahedral meshes.

It turns out that (3.5) can be rewritten under a more compact form. We define two 10×10 matrices, the mass matrix M_L and the volume derivative matrix D_L, as follows:

$$(M_L)_{i,j} = \int_L \varphi_i^L(\mathbf{x}) \varphi_j^L(\mathbf{x})\, d\mathbf{x} \quad \text{and} \quad (D_L)_{i,j} = \int_L \varphi_i^L(\mathbf{x})\ \mathbf{v} \cdot \boldsymbol{\nabla} \varphi_j^L(\mathbf{x})\, d\mathbf{x}. \tag{3.6}$$

With these notations, the scheme (3.5) reads:

$$\begin{aligned}(M_L - \Delta t D_L) \begin{bmatrix} f_{L,0}^{n+1} \\ \vdots \\ f_{L,9}^{n+1} \end{bmatrix} = M_L \begin{bmatrix} f_{L,0}^{n} \\ \vdots \\ f_{L,9}^{n} \end{bmatrix} - \sum_{i=0}^{9} f_{L,i}^{n+1} \left[\sum_{\alpha=0}^{3} \left(\int_{\partial L_\alpha} \varphi_i^L(\boldsymbol{\eta}) \varphi_j^L(\boldsymbol{\eta})\, d\boldsymbol{\eta} \right) (\mathbf{v} \cdot \mathbf{n}_\alpha)_+ \right] \\ - \sum_{i=0}^{9} f_{R_\alpha,i}^{n+1} \left[\sum_{\alpha=0}^{3} \left(\int_{\partial L_\alpha} \varphi_i^{R_\alpha}(\boldsymbol{\eta}) \varphi_j^L(\boldsymbol{\eta})\, d\boldsymbol{\eta} \right) (\mathbf{v} \cdot \mathbf{n}_\alpha)_- \right].\end{aligned} \tag{3.7}$$

In (3.7), we are left with the determination of the two integrals in the right-hand side. To compute them, we need to exploit the mesh connectivity, and develop a procedure allowing us to systematically determine the neighbors of a given tetrahedron. This procedure is detailed in section 3.1.1. Then, once the neighboring tetrahedra and nodes are computed, an algorithm to evaluate the two remaining integrals is given in section 3.1.2.

3.1.1. *Geometry and connectivity.* Here, we detail how to use the mesh connectivity to find the neighbors of a given ten-node tetrahedron L. From now on, we identify the tetrahedron $L \in \mathcal{M}$ with its number $L \in \{0, \dots, n_t - 1\}$, with n_t the number of cells in the mesh $\mathcal{M}$. The DG nodes are numbered locally in the cell and also globally in the mesh. The local numbering is given on figure 3.2.

Because we use a discontinuous approximation, several global DG nodes are found at the same location (for instance at the corners of the tetrahedra or at the middle of the edges). Note that each tetrahedron contains 10 local nodes, and therefore the mesh with n_t tetrahedra contains $10n_t$ global nodes. Therefore, if a given node N has a local number i in a cell L, where $0 \leq i < 10$ and $0 \leq L < n_t$, then its global number i', with $0 \leq i' < 10n_t$, is obtained through the formula

$$i' = 10L + i.$$

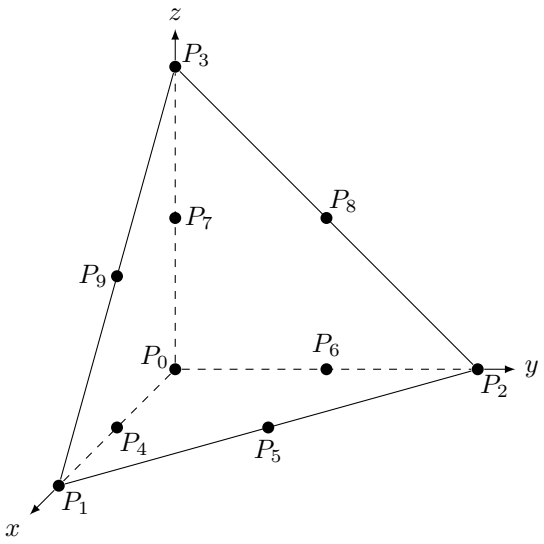


Figure 3.2. Local numbering of the nodes in a cell.

The reverse connectivity is then given by

$$L = \left\lfloor \frac{i'}{10} \right\rfloor \quad \text{and} \quad i = i' \bmod 10.$$

Contrary to the continuous Galerkin method, the connectivity between the local DG nodes and the global DG nodes is a bijection. Let us now denote by $N_{L,i}$ the local nodes of cell L and $N_{i'}$ the global nodes: we get

$$N_{i'} = N_{L,i} \iff i' = 10L + i.$$

We also need a numbering of the nodes inside the cell faces. Consider the local face α in cell L, with $0 \leq \alpha < 4$. We denote by $R_\alpha = \nu(L, \alpha)$ the neighbor cell to L along the face α. We also require a numbering of the nodes inside the local faces. Consider a local face α in cell L, and define k the local node number in the face α, with $0 \leq k < 6$. Then the local node number i in the cell L, with $0 \leq i < 10$, is given by the constant array

$$i = \mathsf{f2c}[\alpha, k], \quad \text{with} \quad \mathsf{f2c} = \begin{pmatrix} 1 & 2 & 3 & 5 & 8 & 9 \\ 0 & 3 & 2 & 7 & 8 & 6 \\ 0 & 1 & 3 & 4 & 9 & 7 \\ 0 & 2 & 1 & 6 & 5 & 4 \end{pmatrix}.$$

This array is obtained by considering the first-order nodes of each face, numbered 0 through 3, and ordering them in a counter-clockwise fashion. Then, we also perform a counter-clockwise ordering of the second-order nodes, numbered 4 through 9.

We leverage the connectivity one last time to recover the global node index in the neighbor along a given face. Let L and R_α be two neighboring cells, connected through local face α in L:

$$R_\alpha = \nu(L, \alpha).$$

We can also find the unique local face β in cell R_α such that

$$L = \nu(R_\alpha, \beta).$$

Now, let k be a local node number in the local face α of cell L. The local node number in cell L is given by

$$i = \mathsf{f2c}(\alpha, k).$$

It corresponds to a global node

$$i' = 10L + i.$$

In the same way, there is a node in cell R_α that is geometrically at the same position than node i'. Suppose that this node is the local node l of face β of cell R_α. The local index j of this node in cell R_α is given by

$$j = \mathsf{f2c}(\beta, l).$$

It corresponds to a global node

$$j' = 10R_\alpha + j.$$

We then set

$$j' = \mathsf{ext\_node}(L, \alpha, k).$$

In other words, the ext_node connectivity array allows us to recover the global nodes that match on a given face. From the above definitions, we also have

$$i' = \mathsf{ext\_node}(R_\alpha, \beta, l).$$

The last ingredient we need to compute the scheme (3.7) is the basis functions φ_i^L. Since they are Lagrange polynomials based on the ten nodes of the tetrahedra, they satisfy the interpolation property

$$\varphi_i^L(N_{L,j}) = \delta_{i,j}\ ,$$

where $\delta_{i,j}$ denotes the usual Kronecker delta. This implies that the components of f in the expansion (3.4) are simply the nodal values at the DG nodes

$$f_L(N_{L,i}, t) = f_{L,i}(t).$$

3.1.2. *Application to the DG scheme.* Having introduced all these notations, we can now rewrite the DG scheme (3.7) in cell L under the form of a local 10×10 linear system, where the unknown is the vector of values $f_L^n := (f_{L,i}^n)_{i\in\{0,\dots,9\}}$ at the DG nodes of cell L. This linear system reads

$$\left(M_L - \Delta t D_L + \Delta t F_L^+\right) \begin{bmatrix} f_{L,0}^{n+1} \\ \vdots \\ f_{L,9}^{n+1} \end{bmatrix} = M_L \begin{bmatrix} f_{L,0}^{n} \\ \vdots \\ f_{L,9}^{n} \end{bmatrix} - \Delta t \sum_{\alpha=0}^{3} F_{L,\alpha}^- \begin{bmatrix} f_{\mathsf{ext\_node}(L,\alpha,0)}^{n+1} \\ \vdots \\ f_{\mathsf{ext\_node}(L,\alpha,5)}^{n+1} \end{bmatrix}, \tag{3.8}$$

with M_L and D_L given by (3.6), and where F_L^+ and $F_{L,\alpha}^-$ are defined as follows:

- F_L^+ is a 10×10 matrix defined by the following assembly algorithm:
  $F_L^+ \leftarrow 0$
  for $\alpha \leftarrow 0$ to 3 **do**
    for $k \leftarrow 0$ to 5 **do**
      $i \leftarrow \mathsf{f2c}(\alpha, k)$
      for $l \leftarrow 0$ to 5 **do**
        $j \leftarrow \mathsf{f2c}(\alpha, l)$
        $\left(F_L^+\right)_{i,j} \leftarrow \left(F_L^+\right)_{i,j} + (\mathbf{v}\cdot\mathbf{n}_\alpha)_+ \displaystyle\int_{\partial L_\alpha} \varphi_i^L(\boldsymbol{\eta})\varphi_j^L(\boldsymbol{\eta})\, d\boldsymbol{\eta}$
- $F_{L,\alpha}^-$ are four 10×6 matrices defined by the following assembly algorithm:
  for $\alpha \leftarrow 0$ to 3 **do**
    $R_\alpha \leftarrow \nu(L, \alpha)$
    for $k \leftarrow 0$ to 5 **do**
      $i \leftarrow \mathsf{f2c}(\alpha, k)$
      for $l \leftarrow 0$ to 5 **do**
        $j \leftarrow \mathsf{f2c}(\alpha, l)$
        $\left(F_{L,\alpha}^-\right)_{i,l} \leftarrow (\mathbf{v}\cdot\mathbf{n}_\alpha)_- \displaystyle\int_{\partial L_\alpha} \varphi_i^{R_\alpha}(\boldsymbol{\eta})\varphi_j^L(\boldsymbol{\eta})\, d\boldsymbol{\eta}$

The main point is that if cell $R_\alpha = \nu(L, \alpha)$ is such that $\mathbf{v}\cdot\mathbf{n} > 0$ on ∂L_α, then the matrix $F_{L,\alpha}^-$ vanishes, and therefore the values of f_R^n in this cell R_α are not needed to compute the values of f_L^n in cell L. In practice, consider the computation of f_L^n in a cell L. For a classical implicit scheme, computing f_L^n requires simultaneously computing f_R^n for each neighbor R of cell L. However, in the scheme (3.7), computing f_L^n only requires the knowledge of f_R^n in cells R upwind from L, i.e. for which $\mathbf{v}\cdot\mathbf{n} > 0$. Reciprocally, to compute f_R^n for such cells R, knowing f_L^n is unnecessary. Therefore, the computation of f^n in a cell is completely decoupled from that of f^n in its neighbor cells. Moreover, since the transport velocity $\mathbf{v}$ is constant, such dependencies can be computed once and for all during the preprocessing phase. This procedure is explained in the next section.

Remark. *Note that, for a classical explicit scheme, the value at each node is updated on its own. In the present method, a very small* 10×10 *linear system is solved in each tetrahedron, provided that the mesh is traversed in the correct way. Therefore, the computational cost for each iteration is quite similar to the cost of*

the explicit scheme. Hence, the present method is termed "quasi-explicit", meaning it is an implicit equation, but solved in an explicit way.

3.2. **Parallel downwind algorithm.** One time step of the implicit DG scheme consists in computing $f^{n+1} := (f_L^{n+1})_{L\in\mathcal{M}}$, the distribution function at time $(n+1)\Delta t$, from the distribution function f^n of the previous time step. From (3.7) it is clear that one has to solve a linear system, as in any implicit scheme. However, because the kinetic velocity $\mathbf{v}$ is constant and because we use the upwind numerical flux, the linear system is triangular. It can thus be solved cell by cell, by simply sweeping the mesh in the direction of the velocity vector. The sweep algorithm relies on the construction of a Directed Acyclic Graph (a DAG) that we define in section 3.2.1. The resolution of the transport equation is explained in section 3.2.2. This mesh sweeping technique is well suited to parallel computing, and we highlight this application in section 3.2.3.

3.2.1. *Reformulation of the mesh as a graph.* Let L and R be two adjacent cells ($\partial L \cap \partial R \neq \emptyset$), with R the neighbor of L through its face α, i.e. $R = \nu(L,\alpha)$. We say that L is *upwind* with respect to R if $\mathbf{v}\cdot\mathbf{n}_\alpha > 0$ on $\partial L \cap \partial R = \partial L_\alpha$. For a cell L, the solution depends only on the values of f in the upwind cells. For a given constant velocity $\mathbf{v}$ we can build a DAG $\mathcal{G}$ to represent the dependencies between the cells. Each vertex of the graph corresponds to a cell of the mesh. Each edge of the graph is associated to a face between two adjacent cells. If the cell L is upwind with respect to R, then the edge associated to $\partial L \cap \partial R$ is oriented from L to R.

We also consider two additional fictitious cells: the "upwind" cell corresponds to the part of the boundary $\partial\mathcal{D}$ where the velocity $\mathbf{v}$ enters the computational domain and the "downwind" cell corresponds to the part of the boundary $\partial\mathcal{D}$ where the velocity $\mathbf{v}$ exits the computational domain.

The dependency graph for a simple two-dimensional mesh and a given constant velocity is represented on figure 3.3.

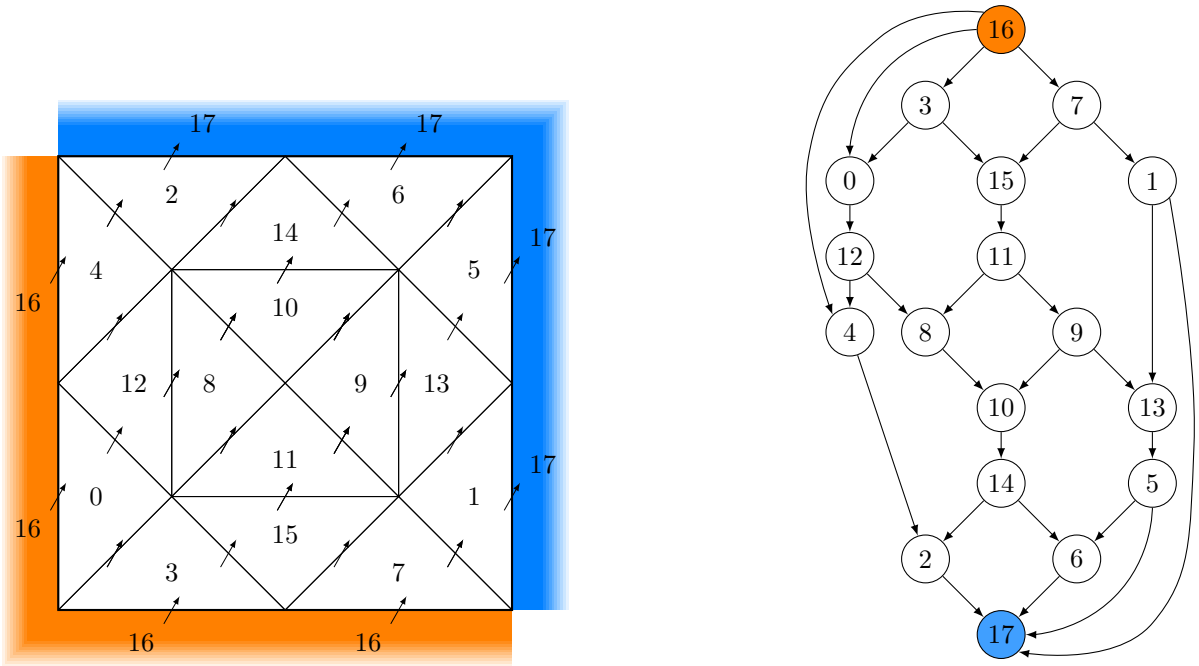


Figure 3.3. Left panel: simple 2D mesh, where the constant velocity $\mathbf{v}$ is represented with an arrow through each edge, and where the upwind and downwind boundary conditions are respectively represented with orange and blue colors. The true cells are numbered from 0 to 15, and the "upwind" and "downwind" cells are respectively numbered 16 and 17. Right panel: dependency graph of the mesh, according to the constant velocity $\mathbf{v}$. The vertices of the graph are the cells of the mesh, and the edges of the graph are the edges of the mesh, The graph edges are directed according to the direction of $\mathbf{v}$ through the corresponding mesh edge.

The construction can be generalized to any unstructured mesh with flat faces. The flatness condition and the fact that the velocity $\mathbf{v}$ is constant ensures that the velocity crosses each face in only one direction. Therefore, the graph does not contain any loop, and it is indeed a DAG. For more details, we refer to [4].

3.2.2. *Solving a transport equation using the graph.* We now state the algorithm to solve one transport equation for a given constant velocity $\mathbf{v}$.

- **Graph ordering**: Compute a topological ordering of the dependency graph $\mathcal{G}$. Let $\mathcal{N}$ be the set of vertices of $\mathcal{G}$. Let $n_{\mathcal{N}}$ be the number of vertices of $\mathcal{G}$, defined by $n_{\mathcal{N}} = \#\mathcal{N}$. Let us recall that $n_{\mathcal{N}} = n_t + 2$, where n_t is the number of cells in the mesh, because of the fictitious "upwind" and "downwind" cells. The ordering is a bijection
$$\sigma : \{1, \dots, n_{\mathcal{N}}\} \to \mathcal{N}$$
such that, if there is an oriented path from A to B, then $\sigma(B) > \sigma(A)$.
- **Linear system preparation**: Perform the assembly, LU decomposition and storage of the local linear system given in (3.8). These computations can also be redone at each time step to save memory at the expense of CPU time; this possibility is explored further in section 7.
- **Transport equation resolution**: At each time step, the following algorithm is applied to each cell in topological order.
  (1) For each upwind face α of L, extract the face values
$$\begin{bmatrix} f^{n+1}_{\mathsf{ext\_node}(L,\alpha,0)} \\ \vdots \\ f^{n+1}_{\mathsf{ext\_node}(L,\alpha,5)} \end{bmatrix}.$$
  (2) Solve the local linear system (3.8). Its unknowns correspond to $f^{n+1}_{L,j}$ in a given cell L, which are computed from the values $f^{n+1}_{R,j}$ in the upwind neighbor cells.

3.2.3. *Parallelization.* As it turns out, it is possible to parallelize large chunks of this transport solver, even though the solver as a whole is not parallelizable. Indeed, because of the dependency graph, we cannot perform all the computations in parallel. For instance, for the mesh of figure 3.3, it is necessary to compute the solution in cells 3 and 7 first. Then, cells 0, 15 and 1 can be computed, etc. Let us emphasize that, in this case, cells 3 and 7 can be computed independently in parallel, like cells 0, 15 and 1, and subsequent downwind cells. Therefore, ordering the graph allows us to subdivide the mesh in several parallel regions, which depend on the transport velocity. In this section, we write the possible parallel optimizations more rigorously.

We assume that the renumbering σ described in section 3.2.2 has been computed. Let us consider a set of cells of the form
$$\mathcal{P} = \{\sigma(k), \sigma(k+1), \dots, \sigma(k+l)\},$$
where k and l are non-negative integers. We shall say that $\mathcal{P}$ is a *parallel region* of size l if there is no edge of the DAG joining two cells inside $\mathcal{P}$. This means that the cells inside $\mathcal{P}$ can be computed completely independently, and in parallel. The algorithm to construct the parallel regions is given below.

$p \leftarrow 0, \quad k_p \leftarrow 0, \quad l_p \leftarrow 0$
while $k_p + l_p < n_{\mathcal{N}}$ **do**
  $\mathcal{P}_p = \emptyset$
  while cell $\sigma(k_p + l_p)$ does not depend on the cells in $\mathcal{P}_p$ **do**
    $\mathcal{P}_p \leftarrow \mathcal{P}_p \cup \{\sigma(k_p + l_p)\}$
    $l_p \leftarrow l_p + 1$
  end while
  $p \leftarrow p + 1, \quad k_p \leftarrow k_{p-1} + l_{p-1}, \quad l_p \leftarrow 0$
end while

Once the n_p parallel regions are constructed, the transport equation resolution from section 3.2.2 is modified as follows.

- **Transport equation resolution**: At each time step, the following algorithm is applied.
  for $p \leftarrow 0$ to $n_p - 1$ **do**
    for each cell L in the parallel region $\mathcal{P}_p$ **do in parallel**
      (1) For each upwind face α of L, extract the face values
$$\begin{bmatrix} f^{n+1}_{\mathsf{ext\_node}(L,\alpha,0)} \\ \vdots \\ f^{n+1}_{\mathsf{ext\_node}(L,\alpha,5)} \end{bmatrix}.$$

(2) Solve the local linear system (3.8).

The parallel efficiency heavily relies on the quality of the topological sort. Indeed, this sorting is generally not unique. A building block of this algorithm is how the graph is visited. The visiting algorithm can be a Depth-First Search (DFS) or a Breadth-First Search (BFS), for instance. In many applications, the DFS is algorithm is preferred, because it is more memory efficient. However, for our application, we have observed a much better parallel efficiency of the topological sorting if a BFS is used instead. See section 7.2 for practical evaluations of the two options.

The parallelization is done by the `rayon`[1] library of the Rust language: once the serial version of the above algorithm is correctly written, the library ensures a correct parallelization. We have also tested other implementations in previous works. For more details on the implementation, we refer for instance to [4], where the algorithm is parallelized with the StarPU runtime, or to [12] where we use a specialized DAG (Direct Acyclic Graph) clustering algorithm.

3.3. **Full matrix assembly.** Another way to practically to apply the implicit DG scheme is to perform a full assembly of the associated linear system. First, arrange all the unknowns $f_{L,i}^{n+1}$ in a single vector Φ^{n+1}. The implicit DG scheme can be written in the following matrix form

$$\frac{\Phi^{n+1} - \Phi^n}{\Delta t} + \mathbb{K}\Phi^{n+1} = \mathcal{B}^n,$$

where $\mathcal{B}^n$ is a vector arising from the approximation of the boundary conditions. One time step of the DG algorithm then corresponds to the resolution of the linear system

$$\mathbb{M}\Phi^{n+1} = \Phi^n + \Delta t \mathcal{B}^n,$$

for the unknown vector Φ^{n+1}.

From what we have seen above, we know that, up to a permutation, the matrix $\mathbb{M}$ is actually a block-triangular matrix and that the size of the diagonal blocks is 10×10. This kind of matrix frequently arises in numerical circuit analysis. Special software libraries have been developed for solving efficiently the associated linear systems. We can mention for instance KLU, from the SuiteSparse library [22]. Accelerated parallel versions of the KLU algorithm also exist [7, 15].

We have thus also implemented a second version of the transport DG solver, where the matrix $\mathbb{M}$ is fully assembled and stored in memory. This obviously induces a higher memory consumption. But the programming is simpler and the task of renumbering the unknowns for discovering the triangular structure of $\mathbb{M}$ is delegated to the KLU library. We compare, in section 7.1, the two approaches in terms of memory occupation and efficiency.

Finally, in order to compare our method to the classical DG approach, we have also implemented the standard explicit DG method to directly solve (2.1). The spatial basis functions are the same as above. The spatial approximation is then exactly the same as in [23]. We thus apply an exact quadrature to the integral terms of the DG formulation. In this third implementation, the time integration is performed by the low-storage third-order Runge-Kutta (RK3) scheme from [46]. This choice is motivated by the fact that the RK3 scheme is the lowest order RK scheme that leads to a CFL condition where the limit stability time step Δt is proportional to the smallest cell size h:

$$\Delta t \sim h.$$

For instance, the RK2 scheme would lead to a CFL condition of the form

$$\Delta t \sim h^{3/2}.$$

The resulting approximation is often referred to as the RKDG method. For a discussion on the RKDG method and its variants, we refer, among others, to [17, 33, 30].

3.4. **Summary of the algorithms.** In the numerical experiments, according to the choice of the velocities in section 6.1.1, we call `D3Q4` the algorithm stemming from the full matrix assembly described in section 3.3, and we denote by `D3Q4P` the parallel version of the downwind algorithm described in section 3.2. These names are derived from the kinetic velocities described in section 6.1.1. The main features of these two algorithms, and of their implementations, are summarized in table 3.1.

[1] https://docs.rs/rayon

Table 3.1. Main differences between the algorithms and the implementation of `D3Q4` and `D3Q4P`.

Method	`D3Q4`	`D3Q4P`
Parallelization	one thread per v_i	deduced from mesh ordering
Memory usage	global DG matrix stored	local DG matrix computed on the fly
Libraries	`SuiteSparse-KLU` (LU-solver) `OpenMP` (parallelization)	`petgraph` (graph ordering) `rayon` (parallelization)
Written in	`C89`	`Rust`

4. Second-order accuracy in time

Equipped with the third-order accurate in space, second-order accurate in time solvers for the transport step (3.1) described in section 3, we turn to the resolution of the relaxation-source step (3.2). The algorithm used for this step is presented in section 4.1. Then, in section 4.2, we discuss and recall the scheme to solve the full kinetic equation (2.19).

4.1. Algorithm for the relaxation-source step.

Here, we propose an algorithm to solve the relaxation-source step (3.2). We assume that the kinetic variables $\mathbf{f}_k^n$ at time $n\Delta t$ are known, and the goal is to obtain the kinetic variables $\mathbf{f}_k^{n+1}$ at time $(n+1)\Delta t$. We first discuss in section 4.1.1 the time discretization of the ODE (3.2), and section 4.1.2 is devoted to a brief explanation of the simple parallelization procedure applied to this discretization.

4.1.1. *Time discretization.* Note that there are no space derivatives in the relaxation-source ODE (3.2). Therefore, the following procedure is applied to each interpolation node in the mesh, independently. For the sake of clarity, we temporarily drop the space indices. Also, note that the forthcoming developments provide a heuristic way to discretize the relaxation-source step, which stems from the abstract BGK representation (2.4). A more rigorous formulation relies on the Dirac comb introduced in section 2.3.

Summing (3.2) for $k \in \{0, \dots, d\}$ leads to

$$\partial_t \sum_{k=0}^{d} \mathbf{f}_k = \sum_{k=0}^{d} \mathbf{g}_k(\mathbf{u}) + \frac{1}{\tau}\left(\sum_{k=0}^{d} \mathbf{m}_k(\mathbf{u}) - \sum_{k=0}^{d} \mathbf{f}_k\right). \tag{4.1}$$

Arguing (2.3), (2.7) and (2.21), the relaxation term vanishes, and (4.1) becomes

$$\partial_t \mathbf{u} = \mathbf{s}(\mathbf{u}), \tag{4.2}$$

which is nothing but the split source term of (2.18). We solve (4.2) with the following Crank-Nicolson time stepping, which is second-order accurate in time:

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = \frac{\mathbf{s}(\mathbf{u}^n) + \mathbf{s}(\mathbf{u}^{n+1})}{2}. \tag{4.3}$$

To apply this procedure, we merely define $\mathbf{u}^n = \sum_k \mathbf{f}_k^n$, and (4.3) yields the value of $\mathbf{u}^{n+1}$.

The goal is now to obtain the updated relaxation variables $\mathbf{f}_k^{n+1}$ from $\mathbf{f}_k^n$, $\mathbf{u}^n$ and $\mathbf{u}^{n+1}$. To that end, we write the Crank-Nicolson time discretization applied to the relaxation-source step (3.2):

$$\frac{\mathbf{f}_k^{n+1} - \mathbf{f}_k^n}{\Delta t} = \frac{\mathbf{g}_k(\mathbf{u}^n) + \mathbf{g}_k(\mathbf{u}^{n+1})}{2} + \frac{1}{2\tau}\big(\mathbf{m}_k(\mathbf{u}^n) + \mathbf{m}_k(\mathbf{u}^{n+1}) - \mathbf{f}_k^n - \mathbf{f}_k^{n+1}\big). \tag{4.4}$$

Straightforward computations lead to the following expression of $\mathbf{f}_k^{n+1}$:

$$\mathbf{f}_k^{n+1} = \frac{1 - \dfrac{\Delta t}{2\tau}}{1 + \dfrac{\Delta t}{2\tau}} \mathbf{f}_k^n + \frac{\Delta t}{1 + \dfrac{\Delta t}{2\tau}} \frac{\mathbf{g}_k(\mathbf{u}^n) + \mathbf{g}_k(\mathbf{u}^{n+1})}{2} + \frac{\dfrac{\Delta t}{\tau}}{1 + \dfrac{\Delta t}{2\tau}} \frac{\mathbf{m}_k(\mathbf{u}^n) + \mathbf{m}_k(\mathbf{u}^{n+1})}{2}. \tag{4.5}$$

Now, following section 2.3, we set

$$\omega = \frac{\dfrac{\Delta t}{\tau}}{1 + \dfrac{\Delta t}{2\tau}}, \tag{4.6}$$

so that (4.5) reads

$$\mathbf{f}_k^{n+1} = (1-\omega)\mathbf{f}_k^n + \omega\tau\,\frac{\mathbf{g}_k(\mathbf{u}^n)+\mathbf{g}_k(\mathbf{u}^{n+1})}{2} + \omega\,\frac{\mathbf{m}_k(\mathbf{u}^n)+\mathbf{m}_k(\mathbf{u}^{n+1})}{2}. \tag{4.7}$$

In the framework of section 2.3, this equation reads

$$\mathbf{f}_k(x,t^+) = (1-\omega)\mathbf{f}_k(x,t^-) + \omega\tau\,\frac{\mathbf{g}_k(\mathbf{u}(x,t))+\mathbf{g}_k(\mathbf{u}(x,t-\Delta t))}{2} + \omega\,\frac{\mathbf{m}_k(\mathbf{u}(x,t))+\mathbf{m}_k(\mathbf{u}(x,t-\Delta t))}{2}.$$

In practice, to achieve the required second-order accuracy in time, we take $\omega = 2 - C\Delta t$, where $C > 0$ is a positive constant, which leads to $\tau = \mathcal{O}(\Delta t^2)$ according to (4.6). This remark enables us to redefine f_k^{n+1} according to (4.7) and up to $\mathcal{O}(\Delta t^2)$, as follows:

$$\mathbf{f}_k^{n+1} = (1-\omega)\mathbf{f}_k^n + \omega\,\frac{\mathbf{m}_k(\mathbf{u}^n)+\mathbf{m}_k(\mathbf{u}^{n+1})}{2}. \tag{4.8}$$

We have thus solved the ODE (3.2) with a second-order accurate in time discretization. Note that this resolution, which leads to the expression (4.8), does not require computing the expression (2.5) of the kinetic source term $\mathbf{g}_k(\mathbf{u})$; we only need to compute the expressions (2.8) of the equilibrium kinetic functions $\mathbf{m}_k$. Also, note that with a vanishing source term $\mathbf{s}(\mathbf{u}) = 0$, the source step (4.3) reduces to $\mathbf{u}^{n+1} = \mathbf{u}^n$, (4.8) becomes

$$\mathbf{f}_k^{n+1} = (1-\omega)\mathbf{f}_k^n + \omega\,\mathbf{m}_k(\mathbf{u}^n),$$

and we recover the expression (2.11) stemming from the over-relaxation procedure.

4.1.2. *Parallelization.* The parallelization of the relaxation-source step is straightforward. We simply apply for each node the relaxation formula (4.8) to the kinetic data. In the KLU implementation of the scheme, this step is parallelized thanks to a simple `OpenMP` loop. In the Rust implementation, we once again rely on `rayon`. Let us remark that, while embarrassingly parallel, the relaxation-source step induces many cache misses because the organization of the kinetic data into memory cannot be optimal both in the transport and in the relaxation-source steps.

4.2. **Palindromic time discretization of the kinetic equations.** Equipped with the solver from section 3 for the transport part (3.1) of the kinetic equations, and the procedure from section 4.1 for the relaxation-source part (3.2), we now summarize the full resolution of the kinetic equations (2.19). Following [21], we suggest a palindromic time integration. For the sake of clarity, let us denote by $\mathscr{T}(\Delta t, t)$ the transport solver from section 3 applied with a time step Δt and with boundary conditions taken at time t. Similarly, we denote by $\mathscr{R}(\Delta t)$ the relaxation-source procedure described in section 4.1 applied with a time step Δt. Note that the time-symmetric nature of the following palindromic time discretization implies that it remains second-order accurate in time. In addition, it is possible to reach any order of accuracy by simply increasing the number of steps of the palindromic splitting, even with the time-symmetric Crank-Nicolson building block. See for instance [40, 21].

The following algorithm summarizes the resolution of the kinetic equations (2.19).

Algorithm. *Assume that, for $k \in \{0,\dots,d\}$, the kinetic variables $\mathbf{f}_k^n$ are known at time $n\Delta t$. The following procedure computes the updated kinetic variables $\mathbf{f}_k^{n+1}$ at time $(n+1)\Delta t$ with a second-order accuracy in time:*

(1) Apply $\mathscr{T}(\frac{\Delta t}{2}, n\Delta t)$, with initial condition $\mathbf{f}_k^n$, to compute the transported kinetic variables $\underline{\mathbf{f}}_k^n$.

(2) Apply $\mathscr{R}(\Delta t)$ to compute the relaxed kinetic variables $\bar{\mathbf{f}}_k^{n+1}$, as follows:

(a) compute $\underline{\mathbf{u}}^n$ by summing $\underline{\mathbf{f}}_k^n$ for $k \in \{0,\dots,d\}$;

(b) compute $\bar{\mathbf{u}}^{n+1}$ by applying the source term according to (4.3):

$$\frac{\bar{\mathbf{u}}^{n+1}-\underline{\mathbf{u}}^n}{\Delta t} = \frac{\mathbf{s}(\underline{\mathbf{u}}^n)+\mathbf{s}(\bar{\mathbf{u}}^{n+1})}{2},$$

noting that, if $\mathbf{s}$ is zero, this step is skipped and $\bar{\mathbf{u}}^{n+1}$ is set equal to $\underline{\mathbf{u}}^n$;

(c) compute the relaxed kinetic variables $\bar{\mathbf{f}}_k^{n+1}$ according to (4.8):

$$\bar{\mathbf{f}}_k^{n+1} = (1-\omega)\underline{\mathbf{f}}_k^n + \omega\,\frac{\mathbf{m}_k(\underline{\mathbf{u}}^n)+\mathbf{m}_k(\bar{\mathbf{u}}^{n+1})}{2}.$$

(3) Apply $\mathscr{T}(\frac{\Delta t}{2}, (n+1)\Delta t)$, with initial condition $\bar{\mathbf{f}}_k^{n+1}$, to compute the updated kinetic variables $\mathbf{f}_k^{n+1}$.

Remark. *As written above, the previous algorithm requires two costly transport steps to advance time from* $n\Delta t$ *to* $(n+1)\Delta t$*. However, note that two consecutive time steps respectively end with* $\mathscr{T}(\frac{\Delta t}{2},(n+1)\Delta t)$ *and begin with* $\mathscr{T}(\frac{\Delta t}{2},(n+1)\Delta t)$*. In practice, we collapse these two transport steps into one, by using* $\mathscr{T}(\Delta t,(n+1)\Delta t)$ *instead. This leads to the following palindromic chain to go from time* 0 *to time* $N\Delta t$*, with* $N\in\mathbb{N}$*:*

$$\mathscr{T}\left(\frac{\Delta t}{2},0\right)\ \mathscr{R}(\Delta t)\ \mathscr{T}(\Delta t,\Delta t)\ \mathscr{R}(\Delta t)\ \mathscr{T}(\Delta t,2\Delta t)\ \cdots\ \mathscr{T}(\Delta t,(N-1)\Delta t)\ \mathscr{R}(\Delta t)\ \mathscr{T}\left(\frac{\Delta t}{2},N\Delta t\right).$$

5. CFL-less aspects

The presented method is claimed to be CFL-less, i.e. unconditionally L^2-stable. In order to measure this feature, we have to give a precise definition of the CFL number.

For one-dimensional problems, in the book of Hesthaven [33] on DG nodal methods, the form of the CFL condition is as follows (section 4.8, page 97):

$$\Delta t \leq \beta \frac{1}{\lambda_{\max}} h_{\min}, \tag{5.1}$$

where Δt is the step of the time integrator, $\lambda_{\max}$ is the maximal wave speed, $h_{\min}$ is the minimal distance between two interpolation points and β is the CFL number. In this paper, the maximal wave speed can be either the speed of light c or the kinetic scaling factor λ. From the sub-characteristic condition, we know that $\lambda > \sqrt{3}c$. In this paper, we will consider a CFL condition given by (5.1), with $h_{\min}$ defined by the smallest cell in the mesh, measured by

$$h_{\min} = \min_{L\in\mathcal{M}} \text{size}(L), \quad \text{where} \quad \text{size}(L) = \frac{\text{volume}(L)}{\text{surface}(\partial L)}. \tag{5.2}$$

The maximum CFL number such that the scheme remains stable depends on the time integration method. For explicit methods such as RK3 or RK4, it is of the order of unity, i.e. $\beta_{\max} \sim 1$.

In higher dimensions, the definition of the CFL number is less straightforward. In several papers (such as [17, 44]), one can find the condition

$$\Delta t \leq \gamma \frac{1}{\lambda_{\max}} \frac{\Delta x}{2r+1},$$

where Δx is the cell size and r the polynomial degree (we only consider the case $r = 2$ in this work). In this context, the CFL number is defined by

$$\gamma := \frac{\lambda_{\max}\Delta t}{\Delta x}(2r+1).$$

This definition is well suited for DG methods based on Gauss-Legendre points. For DG methods based on Legendre-Gauss-Lobatto (LGL) nodes, it appears that the stability condition is less constraining [30], and that a more adapted definition is

$$\theta := \frac{\lambda_{\max}\Delta t}{\Delta x}(r+1).$$

In dimension higher than one, there are very few rigorous results on the stability condition of the DG LGL method on unstructured meshes. What is observed in practice (see [13, 44]) is a more constraining stability condition when the dimension increases. In this paper, we give numerical results for very large values of β. The scheme remains stable at these high CFL numbers. We insist on the fact that our scheme is CFL-free, and that the time step is only restricted by the required accuracy. When the wave velocities do not depend on the solution – which is the case for linear PDEs – the scheme is stable whatever the time step. In this work we have observed that, with our definition (5.1) of the CFL number, the classical explicit `RK3DG` scheme is stable up to $\beta \simeq 1.85$.

6. Numerical applications

This section is dedicated to the numerical applications of the method described above. First, in section 6.1, we present the models used in these experiments, we fix the kinetic velocities $\mathcal{V}$, and we give the expressions of the equilibrium kinetic functions $\mathbf{m}_k$ induced by this choice. Then, in section 6.2, we perform an error analysis on two tests cases, one based on Maxwell's equations and the other on the wave equation, to provide a validation of our numerical scheme. This shows that our method is generic enough to accommodate multiple hyperbolic systems. Afterwards, in section 6.3 we illustrate the "CFL-less" aspect of the method and its advantage over an explicit `RK3DG` method when the mesh presents some cells which are way smaller than the variation scales of the studied solution. Finally, in section 6.4, we discuss the source term treatment, and give a realistic simulation of a resistive wire.

For the sake of simplicity, all the numerical experiments are performed on the unit cube $\mathcal{D} = [0,1]^3$. The exact solution is denoted by $\mathbf{u}^{\text{inc}}(\mathbf{x},t)$. In addition, we set $\omega = 2 - 10^{-12}$ to ensure both stability and second-order time accuracy. To estimate the convergence rates of our scheme, we define, at the final time t_{end}, the discrete L^2 error using the mass matrix M_L from (3.6), as follows:

$$e(h_{\min},\Delta t) = \frac{1}{m}\sum_{k=1}^{m}\sqrt{\sum_{L\in\mathcal{M}} e_L^{\intercal} M_L e_L}, \tag{6.1}$$

where $h_{\min}$ is defined in (5.2), and where e_L is the vector given by

$$e_{k,L} = \begin{pmatrix} u^n_{k,L,0} - u^{\text{inc}}_k(\mathbf{x}_{L,0},t_{\text{end}}) \\ \vdots \\ u^n_{k,L,9} - u^{\text{inc}}_k(\mathbf{x}_{L,9},t_{\text{end}}) \end{pmatrix}, \tag{6.2}$$

which represents the error at each of the 10 nodes $\mathbf{x}_{L,i}$ of cell L. With this error definition, for a given time step Δt and for two spatial discretizations characterized by $h_{\min}$ and $h'_{min} = h_{\min}/2$, the spatial order of accuracy μ is then computed using the regression:

$$\mu = \frac{\log(e(h'_{min},\Delta t)/e(h_{\min},\Delta t))}{\log(2)}.$$

In a same way, the temporal order of accuracy κ is determined by considering two time steps Δt and $\Delta t' = \Delta t/2$ for a fixed $h_{\min}$:

$$\kappa = \frac{\log(e(h_{\min},\Delta t')/e(h_{\min},\Delta t))}{\log(2)}.$$

Remark. *All our numerical experiments are performed in three dimensions of space and based on unstructured meshes that are composed of first geometrical order tetrahedra (i.e. with straight edges). For the sake of readability and uniformity in this paper, the numerical fields are always depicted in two dimensions of space. The two-dimensional representations are obtained by slicing the physical domain $\mathcal{D}$ with the* `ParaView` *software* [3]. *Our experiments are performed on the unit cube $\mathcal{D} = [0,1]^3$, which is sliced at $x_3 = 0.5$ for the visualization of the results in sections 6.2 and 6.3. An illustration is given in figure 6.1. Additionally, in the context of the spatial error analysis, the series of meshes is specifically built using a "refine by splitting" technique that allows refining a given unstructured mesh by a factor of two on the whole domain. These mesh generation functionalities are provided by* `Gmsh`, *see* [31].

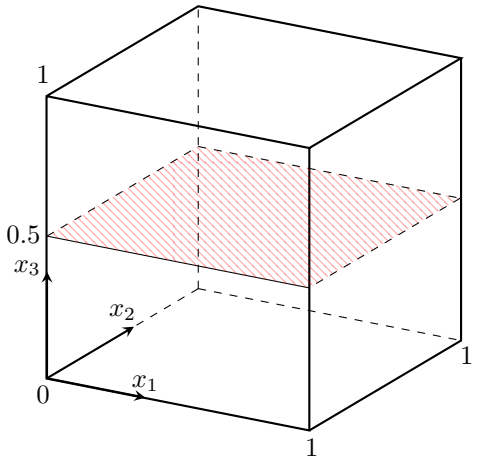


Figure 6.1. Illustration, here on the unit cube $[0,1]^3$, of the setup used to depict the numerical fields. All the simulations are performed in 3D but the data are displayed as a 2D field by using a slice (here in red) through the computational domain.

6.1. **Models and kinetic velocities.** As a first step, we fix the expressions of the kinetic velocities $\mathbf{v}_k$ in section 6.1.1, and we give the resulting simple formula for the equilibrium kinetic functions $\mathbf{m}_k(\mathbf{u})$. Then, we describe the three-dimensional models used in these numerical experiments: Maxwell's equations in section 6.1.2, and the wave equation in section 6.1.3.

6.1.1. *Choice of the kinetic velocities.* For the sake of simplicity, we use the following values for the kinetic velocities $\mathbf{v}_k$ from (2.2):

$$v_0 = \begin{pmatrix}\lambda\\ \lambda\\ \lambda\end{pmatrix}, \quad v_1 = \begin{pmatrix}\lambda\\ -\lambda\\ -\lambda\end{pmatrix}, \quad v_2 = \begin{pmatrix}-\lambda\\ \lambda\\ -\lambda\end{pmatrix}, \quad v_3 = \begin{pmatrix}-\lambda\\ -\lambda\\ \lambda\end{pmatrix}.$$

This choice corresponds to the outer normals to the four faces of the reference tetrahedron depicted in figure 3.2; analogously to Lattice-Boltzmann literature (see for instance [43]), we name them "D3Q4". The parameter λ is set to $\sqrt{3}$ to satisfy the subcharacteristic condition, see for instance [14, 2, 9].

With these expressions of the velocities, the relations (2.8) yield, after straightforward computations, the following simple expressions for the equilibrium kinetic functions $\mathbf{m}_k(\mathbf{u})$:

$$\mathbf{m}_k(\mathbf{u}) = \frac{\mathbf{u}}{4} + \frac{\mathbf{q}(\mathbf{u}, \mathbf{v}_k)}{4\lambda^2}.$$

6.1.2. *Three-dimensional Maxwell's equations.* Let us consider, for $\mathbf{x}$ in a spatial domain $\mathcal{D} \subset \mathbb{R}^3$ and for $t > 0$, the following non-dimensional Maxwell's equations

$$\begin{cases}\partial_t \mathbf{E} - \boldsymbol{\nabla} \times \mathbf{H} = -\sigma \mathbf{E},\\ \partial_t \mathbf{H} + \boldsymbol{\nabla} \times \mathbf{E} = 0,\end{cases} \tag{6.3}$$

where $\mathbf{E}(\mathbf{x}, t) = (E_1, E_2, E_3)^\intercal \in \mathbb{R}^3$ is the electric field, $\mathbf{H}(\mathbf{x}, t) = (H_1, H_2, H_3)^\intercal \in \mathbb{R}^3$ is the magnetic field, and $\sigma(\mathbf{x}) \geq 0$ is the electrical conductivity. For this system, the $m = 6$ conservative variables and the flux in a direction $\mathbf{n} = (n_1, n_2, n_3)^\intercal \in \mathbb{R}^3$ are respectively

$$\mathbf{u}(\mathbf{x}, t) = (E_1, E_2, E_3, H_1, H_2, H_3)^\intercal$$

and

$$\mathbf{q}(\mathbf{u}, \mathbf{n}) = \left(-(\mathbf{n} \times \mathbf{H})^T,\ (\mathbf{n} \times \mathbf{E})^T\right)^\intercal.$$

For our validation experiments, the test case relies on solving (6.3) on the unit cube $\mathcal{D} = [0, 1]^3$, until the final time $t_{\text{end}} = 1$ and with the following initial and boundary conditions:

$$\begin{cases}\mathbf{u}(\mathbf{x}, 0) = \mathbf{u}^{\text{inc}}(\mathbf{x}, 0), & \forall \mathbf{x} \in \mathcal{D},\\ \mathbf{u}(\mathbf{x}, t) = \mathbf{u}^{\text{inc}}(\mathbf{x}, t), & \forall \mathbf{x} \in \partial\mathcal{D},\ \forall t \leq t_{\text{end}},\end{cases}$$

where $\mathbf{u}^{\text{inc}}$ is an exact solution of (6.3), to be prescribed for each experiment. In this case of the non-dimensional Maxwell's equations, we take $\lambda_{\max} = 1$ in the expression (5.1) of the time step Δt. In this way we can compare the CFL numbers of the `RK3DG` and the `D3Q4(P)` algorithms.

6.1.3. *Three-dimensional wave equation.* The second system that we consider models the propagation, for $t \leq t_{\text{end}}$, of a three-dimensional wave in a homogeneous domain $\mathcal{D} = [0, 1]^3$. Mathematically, the problem reads

$$\begin{cases}\partial_{tt} w(\mathbf{x}, t) - c^2 \Delta w(\mathbf{x}, t) = 0, & \forall (\mathbf{x}, t) \in \mathcal{D},\ \forall t \leq t_{\text{end}},\\ w(\mathbf{x}, 0) = g(\mathbf{x}, 0), & \forall \mathbf{x} \in \mathcal{D},\\ \partial_t w(\mathbf{x}, 0) = 0, & \forall \mathbf{x} \in \mathcal{D},\\ w(\mathbf{x}, t) = g(\mathbf{x}, t), & \forall \mathbf{x} \in \partial\mathcal{D},\ \forall t \leq t_{\text{end}},\end{cases} \tag{6.4}$$

with $w : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}$ the unknown function, $g : \mathbb{R}^4 \to \mathbb{R}$ a known exact solution and c the wave velocity. For our applications, we set the wave velocity to $c \equiv 1$, which leads to $\lambda_{\max} = 1$. To comply with the formalism described in section 2, the system (6.4) is rewritten into a system of conservation laws using the following definitions of the conserved variables and flux:

$$\mathbf{u}(\mathbf{x}, t) = (\partial_t w,\ \partial_1 w,\ \partial_2 w,\ \partial_3 w)^\intercal$$

and

$$\mathbf{q}(\mathbf{u}, \mathbf{n}) = (-c^2 \mathbf{n} \cdot \boldsymbol{\nabla}_{\mathbf{x}} w,\ -n_1 \partial_t w,\ -n_2 \partial_t w,\ -n_3 \partial_t w)^\intercal.$$

One can compute the exact time-domain solution of (6.4) through Kirchhoff's formula, namely

$$g(\mathbf{x}, t) = \frac{1}{4\pi} \partial_t \left(t \int_{|\mathbf{y}|=1} g(\mathbf{x} + ct\mathbf{y}, 0) ds(\mathbf{y}) \right). \tag{6.5}$$

The integral term in (6.5) is numerically evaluated with an accurate Lebedev quadrature [39] of order $p = 27$ which uses 302 points on the unit sphere $\mathbb{S}^2$. Finally, the initial condition g is chosen as a compactly supported function of the form

$$g(\mathbf{x}, t = 0) = \begin{cases} \left(1 - \dfrac{|\mathbf{x} - \mathbf{x}_0|^2}{\eta^2}\right)^k & \text{if } \dfrac{|\mathbf{x} - \mathbf{x}_0|^2}{\eta^2} \leq 1, \\ 0 & \text{otherwise,} \end{cases} \tag{6.6}$$

with $\eta \in \mathbb{R}$, $\mathbf{x}_0 \in \mathbb{R}^3$ and $k \in \mathbb{N}$. The coefficient η enables us to control the diameter of the initial condition at $t = 0$. In our simulation, we set $\eta = 0.20$ and $k = 6$ to ensure sufficient regularity on g, and we take $\mathbf{x}_0 = (0.5, 0.5, 0.5)^\intercal$.

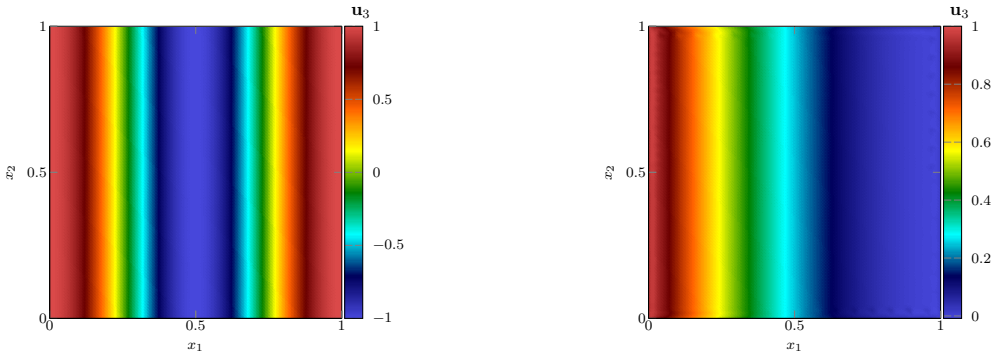
6.2. **Validation experiments.** In this section, we propose a few experiments to ensure that our method yields the correct approximate solution. First, in section 6.2.1, we discuss the approximation of Maxwell's equations, and provide a study of the order of accuracy, both in time and space. Then, in section 6.2.2, we highlight the generality of our method by considering a numerical approximation of the wave equation.

6.2.1. *Three-dimensional Maxwell's equations.* This first set of experiments focuses on Maxwell's equations (6.3) with a vanishing conductivity source term, i.e. $\sigma = 0$. We consider the following simple exact time-domain solution satisfying (6.3):

$$\mathbf{u}^{\text{inc}}(\mathbf{x}, t) = \begin{pmatrix} 0 \\ 0 \\ f(x_1 - t) \\ 0 \\ -f(x_1 - t) \\ 0 \end{pmatrix}, \tag{6.7}$$

with $f : \mathbb{R} \to \mathbb{R}$ a given function, which is nothing but a plane wave solution.

In order to evaluate the spatial order of the scheme, we choose a low frequency plane wave by setting $f(s) = \cos(2\pi s)$ in (6.7). As an illustration, the third component of the exact solution is displayed in figure 6.2. To ensure that the error component due to time integration vanishes, the time step is set to a very small value $\Delta t = 10^{-4}$. The spatial convergence results are shown in table 6.1a and figure 6.3a. As observed for instance in [32, 6], we found a spatial order of accuracy close to $\mu \simeq p + 1$ when using a DG $p = 2$ approximation, which validates the spatial part of the scheme. The temporal order is evaluated by taking $f(s) = s^2$ in (6.7). Taking such a quadratic function ensures an exact spatial integration of the solution thanks to the DG $p = 2$ approximation, which therefore allows us to isolate the error component due to time integration. The temporal convergence results are shown in table 6.1b and figure 6.3b. We observe that the temporal order is close to 2, which is consistent with the Crank-Nicolson method we are using and thus validates the temporal integration part of the scheme.



(a) Case 1: $f(s) = cos(2\pi s)$ **(b)** Case 2: $f(s) = s^2$

Figure 6.2. Maxwell test case from section 6.2.1: depiction of the third component of the exact solution $\mathbf{u}_3(\mathbf{x}, t) = E_3(\mathbf{x}, t)$, sliced at $x_3 = 0.5$. In each panel, we have taken a different value for the function $f(s)$ in (6.7). The left and right panels respectively illustrate the exact solutions used in the study of the spatial and temporal orders of accuracy.

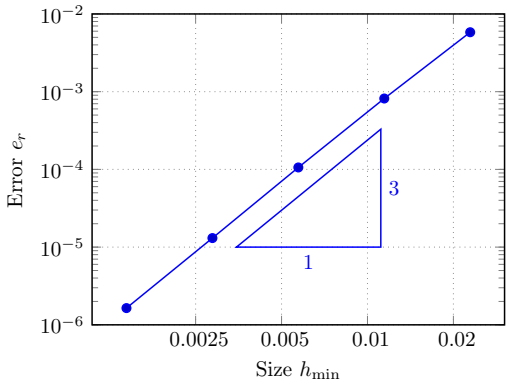
Table 6.1. Maxwell test case from section 6.2.1: study of the order of accuracy. The tables contain the values of the error e_r, as well as the values of the spatial order of accuracy μ with respect to the mesh size $h_{\min}$ (left table) and the temporal order of accuracy κ with respect to the time step Δt (right table).

Size $h_{\min}$	Error e_r	Order μ
$2.290 \cdot 10^{-2}$	$5.820 \cdot 10^{-3}$	—
$1.145 \cdot 10^{-2}$	$8.168 \cdot 10^{-4}$	2.8329
$5.725 \cdot 10^{-3}$	$1.059 \cdot 10^{-4}$	2.9474
$2.863 \cdot 10^{-3}$	$1.307 \cdot 10^{-5}$	3.0182
$1.431 \cdot 10^{-3}$	$1.642 \cdot 10^{-6}$	2.9973

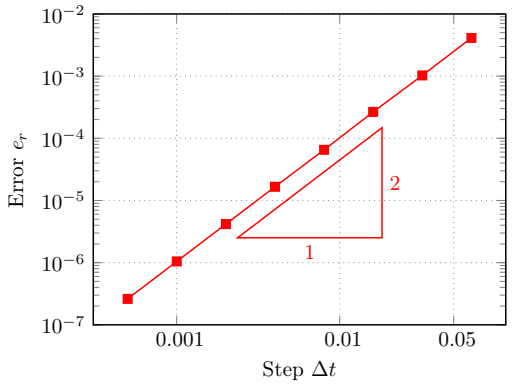
(a) Spatial convergence.

Step Δt	Error e_r	Order κ
$6.415 \cdot 10^{-2}$	$4.114 \cdot 10^{-3}$	—
$3.208 \cdot 10^{-2}$	$1.026 \cdot 10^{-3}$	2.0030
$1.604 \cdot 10^{-2}$	$2.658 \cdot 10^{-4}$	1.9495
$8.019 \cdot 10^{-3}$	$6.531 \cdot 10^{-5}$	2.0247
$4.009 \cdot 10^{-3}$	$1.661 \cdot 10^{-5}$	1.9750
$2.005 \cdot 10^{-3}$	$4.176 \cdot 10^{-6}$	1.9920
$1.002 \cdot 10^{-3}$	$1.046 \cdot 10^{-6}$	1.9980
$5.010 \cdot 10^{-4}$	$2.615 \cdot 10^{-7}$	1.9995

(b) Temporal convergence.



(a) Spatial convergence.



(b) Temporal convergence.

Figure 6.3. Maxwell test case from section 6.2.1: plot in `loglog` scale of the numerical convergence rates. The error e_r is displayed with respect to the mesh size $h_{\min}$ in the left panel, and with respect to the time step Δt in the right panel.

6.2.2. *Three-dimensional wave equation.* To insist on the generic aspect of the present method, we now perform an additional time order validation by considering the wave equation (6.4) and its exact solution (6.5).

With the aim of studying the time order of the scheme, the simulations are performed on a very fine mesh. To study the behavior of our scheme both before and after the wave front came into contact with the boundaries of the domain, the error (6.1) is evaluated at two different final times, $t_{\text{end}} = 0.3$ and $t_{\text{end}} = 0.5$. In figure 6.4, the numerical values of $u_1 = \partial_t w$ (top panels) and $u_2 = \partial_1 w$ (bottom panels) are depicted. We observe no perturbations due to the boundary, and the numerical solution seems to be in accordance with the exact solution. To quantify this consistency, we report in table 6.2, and we display in figure 6.5, the values of the error e_r and the temporal order κ with respect to the time step and both final times. In each case, we observe that the temporal order remains approximately equal to 2, which further validates the consistency of our approach.

6.3. **Unconditional stability.** It is commonplace to see appearing, when generating an unstructured mesh composed of tetrahedra, cells whose size can be several orders of magnitude smaller than the largest one. Depending on the problem under consideration and the physical solution, such finely discretized areas may not be necessary for accurate simulations. An example of this behavior is found when the spatial variations take place on scales much larger than the smallest cells of the mesh. A solution would be to modify the mesh to remove these unnecessarily refined areas. However, this is often impossible in an industrial context with regard to the time and the cost that such modifications would impose. With a classical explicit scheme, handling this type of configuration requires, because of the CFL condition, the use of very small time steps. As stated above, the method presented in this study is unconditionally stable and thus offers the possibility to increase the time step according to the nature of the solution and the desired accuracy.

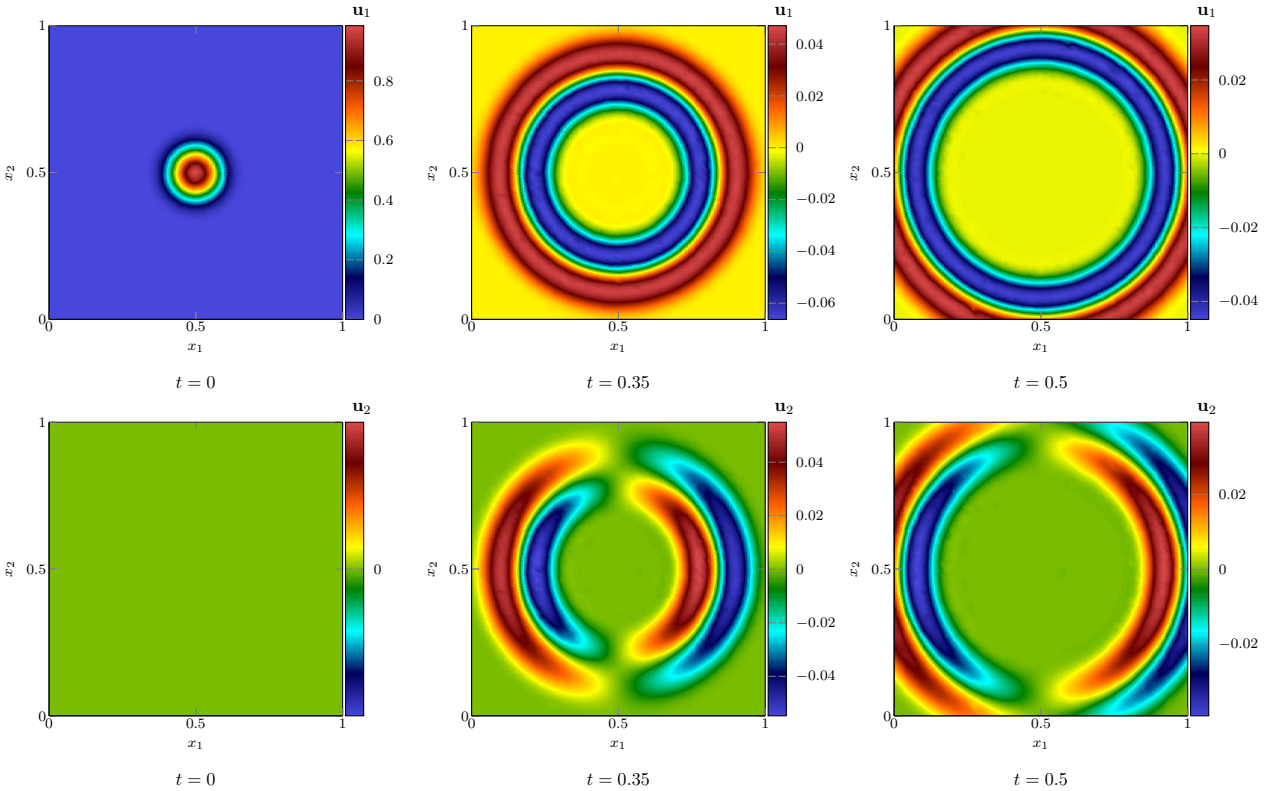


Figure 6.4. Wave equation test case from section 6.2.2: depiction of the first two components of the numerical approximations of $\mathbf{u}_1(\mathbf{x}, t) = \partial_t w(\mathbf{x}, t)$ (top panels) and $\mathbf{u}_2(\mathbf{x}, t) = \partial_1 w(\mathbf{x}, t)$ (bottom panels), sliced at $x_3 = 0.5$. From left to right, the solutions are depicted at time $t = 0$, $t = 0.35$ (before the wave hits the boundary) and $t = 0.5$ (after the wave has hit the boundary).

Table 6.2. Wave equation test case from section 6.2.2: study of the temporal order of accuracy. The table contains the values of the error e_r, as well as the values of the temporal order of accuracy κ with respect to the time step Δt. These errors and orders of accuracy are collected at time $t = 0.35$ and $t = 0.5$, respectively before and after the wave has hit the domain boundary.

	$t_{\text{end}} = 0.35$		$t_{\text{end}} = 0.50$	
Step Δt	Error e_r	Order κ	Error e_r	Order κ
$1.343 \cdot 10^{-2}$	$9.091 \cdot 10^{-3}$	—	$1.144 \cdot 10^{-2}$	—
$6.713 \cdot 10^{-3}$	$2.904 \cdot 10^{-3}$	1.6467	$3.520 \cdot 10^{-3}$	1.7012
$3.357 \cdot 10^{-3}$	$7.673 \cdot 10^{-4}$	1.9601	$8.788 \cdot 10^{-4}$	1.9559
$1.678 \cdot 10^{-3}$	$2.026 \cdot 10^{-4}$	1.9813	$2.327 \cdot 10^{-4}$	1.9716

For this numerical experiment, we once again consider the Maxwell test from section 6.2.1 with $f(s) = \cos(2\pi\nu s)$ in (6.7). It represents the propagation in the unit cube of a plane wave of frequency ν, characterized by the function $f(s)$. For our simulations, we consider $t_{\text{end}} = 1$, $\nu \in \{1, 2\}$, and two mesh configurations. The first one, which we call $\mathcal{M}_1$, is based on a classical mesh of the unit cube with a refinement $r_c = 16$ imposed on its edges. The second one, called $\mathcal{M}_2$, is based on the geometric inclusion of a small torus in the center of the unit cube. This torus has a large radius $R = 0.1$ and a cross-sectional radius $r = 0.01$. We impose a refinement of $r_c = 8$ on the edges of the cube and a one of $r_t = 128$ on the torus circumference.

Remark. *The presence of the torus is only intended to force the generation of a conformal mesh with very small elements in a delimited area of space. In this test case, the dielectric properties of the cube and of the torus remain absolutely identical (i.e. vacuum). To get a better idea of the employed refinement, representations of the two meshes (sliced at $x_2 = 0.5$) are given in figure 6.6. In this figure, we clearly see the strong variation*

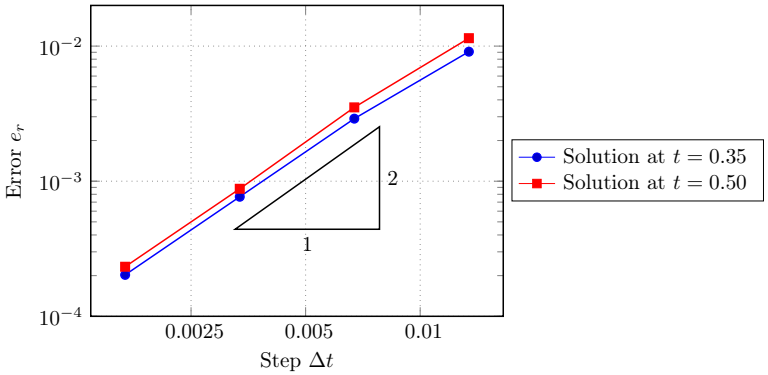


Figure 6.5. Wave equation from section 6.2.1: plot in `loglog` scale of the temporal convergence rates. The error e_r is displayed with respect to the time step Δt, at time $t = 0.35$ and $t = 0.5$.

of the cell size induced by the torus presence. For $\mathcal{M}_1$, the ratio between the largest and smallest cell size is about 4.5*, while this ratio is around* 60 *for $\mathcal{M}_2$, with the cell size defined by* (5.2)*.*

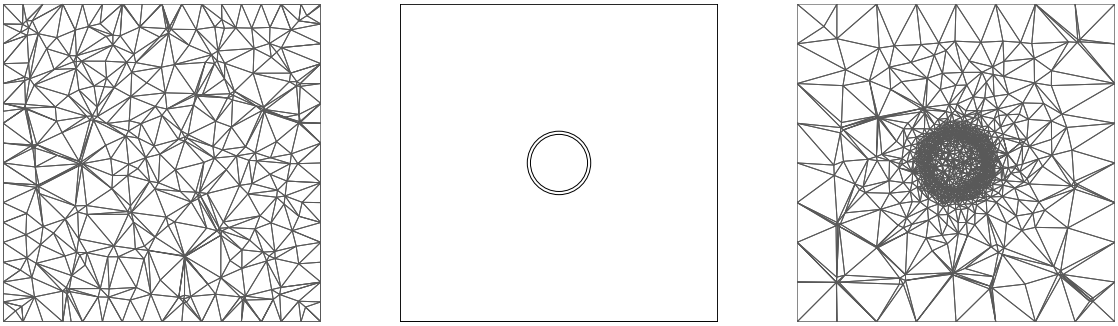


Figure 6.6. Depiction of the two meshes under consideration in section 6.3, sliced at $x_2 = 0.5$. Left panel: mesh $\mathcal{M}_1$, with uniformly spaced unstructured tetrahedra. Middle panel: geometry for the mesh $\mathcal{M}_2$, with a torus within the cubic domain. Right panel: mesh $\mathcal{M}_2$, with uniformly spaced unstructured tetrahedra at the boundary, and with a local refinement at the center of the mesh.

To illustrate the interest of our method, we report in tables 6.3 and 6.4 the results obtained with the `D3Q4P` implementation and the one achieved with an explicit `RK3DG` method. The errors between the exact and the numerical solutions are reported with respect to different CFL numbers, i.e. values of β in (5.1). Using the setup described in figure 6.1, we depict for both meshes in figures 6.7 and 6.8 the third component of the numerical solution. These plots display the spatial variation of E_3 for the two frequencies $\nu \in \{2, 5\}$ and for several values of the CFL number β.

The starting point of this study was first to numerically determine the largest CFL value β for which the `RK3DG` remained stable. On the present test case and as indicated in section 5, the highest value we found was $\beta = 1.85$. After this threshold, the error quickly grows until it explodes for $\beta = 1.87$. Looking at tables 6.3 and 6.4 we first notice that for values of $\beta \leq 1.85$, both methods have converged in time, and their respective errors e_r are of the same order of magnitude. For small time steps, the error is dominated by the spatial accuracy that our scheme is able to achieve on the largest cells of the mesh. For a same given mesh and small time step, the error e_r is also impacted by the spatial variations of the solution. For the high frequency $\nu = 5$, we observe a slightly larger error than for the low frequency $\nu = 2$.

For larger values of the CFL number ($\beta > 1.85$) and as claimed in the previous sections, we observe in tables 6.3 and 6.4 that our relaxation scheme remains stable even in the case $\beta = 185$. On figure 6.7, we display the progressive degradation of the solution as the CFL value goes up ($\beta = 1.85$, 37 and 92 from left to right). However the solution never explodes. On the second configuration (i.e. with the locally refined torus), h_{min} is ten times smaller than the one in the first configuration. The `RK3DG` CFL condition directly reflects this variation on the time step. Here, this drastic constraint imposed on the time step is clearly unnecessary to accurately approximate the solution. As shown in 6.4 for both cases $\nu \in \{2, 5\}$, our relaxation scheme is

able to produce almost equivalent results with a CFL value $\beta = 18.5$, compared to the value $\beta = 1.85$ needed for the RK3DG scheme. On figure 6.8 and for the case $\nu = 2$, we note a barely visible deterioration for $\beta = 185$. For the case $\nu = 5$, the solution still looks very similar for $\beta = 18.5$, but becomes deformed for $\beta = 185$.

Tables 6.3 and 6.4 also contain the CPU times of each simulation for sequential (1 thread) and parallel (24 threads) runs. Since the algorithms, the languages, and the optimizations are different in both implementations, we would like to emphasize the fact that we do not seek to directly compare the execution times of both methods. Indeed, the objective is only to give an order of magnitude of the execution time. Nevertheless, the results in tables 6.3 and 6.4 illustrate the efficiency of the D3Q4P method, thereby placing it in the category of explicit methods despite it being implicit and unconditionally stable. Those results confirm thus the unconditional stability of our method, and its interest when one wishes to adapt the simulation time step in function of the nature of the solution, without having to deal with a constraining CFL condition imposed by the underlying mesh.

Table 6.3. Results for the mesh $\mathcal{M}_1$ described in section 6.3. For this mesh, we get $h_{\min} \simeq 2.28 \times 10^{-3}$. The numerical test is done with the exact solution from section 6.2.1, with $f(s) = \cos(\nu\pi s)$, $\nu \in \{2, 5\}$. We collect the error e_r and the CPU time with respect to the value of the CFL number, for both values of ν and for the RK3DG and D3Q4P methods.

			Error e_r		CPU (s)	
Method	CFL β	Δt	$\nu = 2$	$\nu = 5$	1 thread	24 threads
RK3DG	0.37	0.00084	0.00032	0.00609	360.01	72.54
D3Q4P	0.37	0.00084	0.00046	0.00627	96.27	15.53
RK3DG	0.93	0.00211	0.00032	0.00610	146.81	29.19
D3Q4P	0.93	0.00211	0.00047	0.00657	38.63	6.52
RK3DG	1.85	0.00422	0.00032	0.00609	77.32	16.07
D3Q4P	1.85	0.00422	0.00062	0.00891	19.23	3.26
D3Q4P	3.70	0.00845	0.00162	0.02397	9.92	1.64
D3Q4P	9.25	0.02112	0.00960	0.14851	3.95	0.63
D3Q4P	18.50	0.04223	0.03990	0.42444	2.00	0.33
D3Q4P	37.00	0.08447	0.14919	0.34411	1.02	0.17
D3Q4P	92.50	0.21117	0.25771	0.67218	0.45	0.08
D3Q4P	185.00	0.42234	0.45671	0.49513	0.29	0.05

Table 6.4. Results for the mesh $\mathcal{M}_2$ described in section 6.3. For this mesh, we get $h_{\min} \simeq 2.47 \times 10^{-4}$. The numerical test is done with the exact solution from section 6.2.1, with $f(s) = \cos(\nu\pi s)$, $\nu \in \{2, 5\}$. We collect the error e_r and the CPU time with respect to the value of the CFL number, for both values of ν and for the RK3DG and D3Q4P methods.

			Error e_r		CPU (s)	
Method	CFL β	Δt	$\nu = 2$	$\nu = 5$	1 thread	24 threads
RK3DG	0.37	0.00009	0.00070	0.01238	4,607.95	785.28
D3Q4P	0.37	0.00009	0.00103	0.01467	1,524.45	234.48
RK3DG	0.93	0.00023	0.00070	0.01238	2,189.76	384.79
D3Q4P	0.93	0.00023	0.00103	0.01467	613.44	90.84
RK3DG	1.85	0.00046	0.00070	0.01238	1,121.96	212.60
D3Q4P	1.85	0.00046	0.00103	0.01467	304.41	45.14
D3Q4P	3.70	0.00091	0.00103	0.01468	153.09	22.40
D3Q4P	9.25	0.00228	0.00104	0.01479	61.60	8.96
D3Q4P	18.50	0.00456	0.00115	0.01619	30.76	4.53
D3Q4P	37.00	0.00912	0.00210	0.02992	15.34	2.46
D3Q4P	92.50	0.02281	0.01107	0.16589	6.17	0.92
D3Q4P	185.00	0.04562	0.04509	0.40344	3.10	0.48

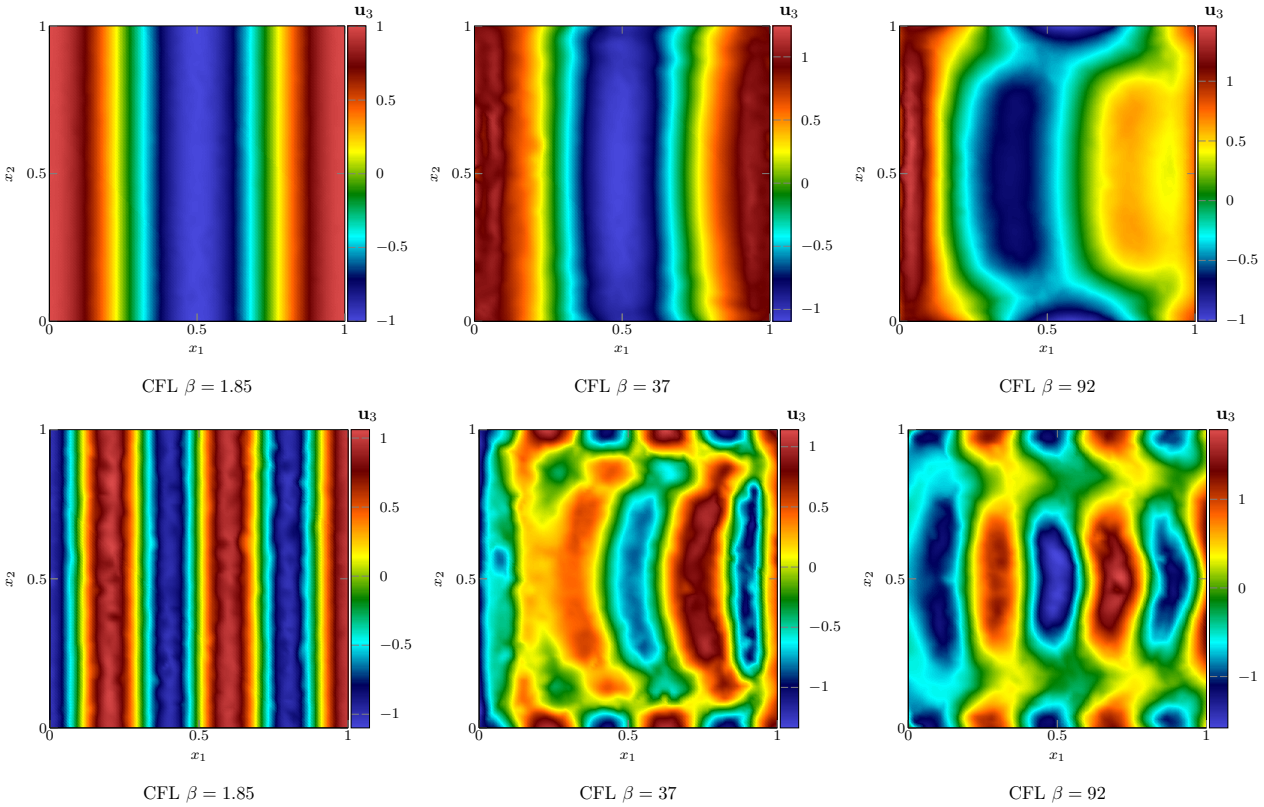


Figure 6.7. Maxwell test case from section 6.3: graphical representation of the third component $\mathbf{u}_3(\mathbf{x}, t) = E_3(\mathbf{x}, t)$ of the approximate solution, sliced at $x_3 = 0.5$, using the first mesh $\mathcal{M}_1$. Top panels: $\nu = 2$ in $f(s)$; bottom panels: $\nu = 5$ in $f(s)$. From left to right, the CFL coefficients are 1.85, 37 and 92.

6.4. **Numerical experiments with a source term.** We finally turn to numerical experiments involving a source term, namely the conductivity term $\sigma > 0$ in (6.3). We first study the order of accuracy in time in section 6.4.1, before performing the more complex simulation of a resistive wire in section 6.4.2.

6.4.1. *Order of accuracy in time.* We introduce the following exact solution of Maxwell's equations with conductivity (6.3):

$$\mathbf{u}_\sigma^{\text{inc}}(\mathbf{x}, t) = \begin{pmatrix} (\psi_1(x_1) + \psi_2(x_2, x_3))e^{-\sigma t} \\ (\psi_1(x_2) + \psi_2(x_3, x_1))e^{-\sigma t} \\ (\psi_1(x_3) + \psi_2(x_1, x_2))e^{-\sigma t} \\ (\partial_1\psi_2(x_3, x_1) + \partial_2\psi_2(x_1, x_2))\frac{1-e^{-\sigma t}}{\sigma} \\ (\partial_1\psi_2(x_1, x_2) + \partial_2\psi_2(x_2, x_3))\frac{1-e^{-\sigma t}}{\sigma} \\ (\partial_1\psi_2(x_2, x_3) + \partial_2\psi_2(x_3, x_1))\frac{1-e^{-\sigma t}}{\sigma} \end{pmatrix}, \tag{6.8}$$

with $\psi_1 \in \mathcal{C}^\infty(\mathbb{R}, \mathbb{R})$ and $\psi_2 \in \mathcal{C}^\infty(\mathbb{R}^2, \mathbb{R})$ such that $(\partial_1^2 + \partial_2^2)\psi_2 \equiv 0$.

To validate the second-order accuracy in time of the scheme, we first consider a solution that does not depend on space in order to eliminate potential boundary issues, and we then study a solution with a space dependence with a special treatment of the boundary error. We take the final time $t_{\text{end}} = 0.25$ and the conductivity $\sigma = 1$. In addition, to avoid errors due to a poor spatial approximation, the experiments are performed on a very fine mesh, with a refinement $r_c = 72$ imposed on its edges, and which contains 544030 elements.

To make $\mathbf{u}_\sigma^{\text{inc}}$ independent of space, we take

$$\psi_1(x_1) = 1 \text{ and } \psi_2(x_2, x_3) = 1. \tag{6.9}$$

To ensure that $\mathbf{u}_\sigma^{\text{inc}}$ depends on space, we take

$$\psi_1(x_1) = x_1^3 \text{ and } \psi_2(x_2, x_3) = x_2^2(3x_3 - x_2) + x_3^2(3x_2 - x_3). \tag{6.10}$$

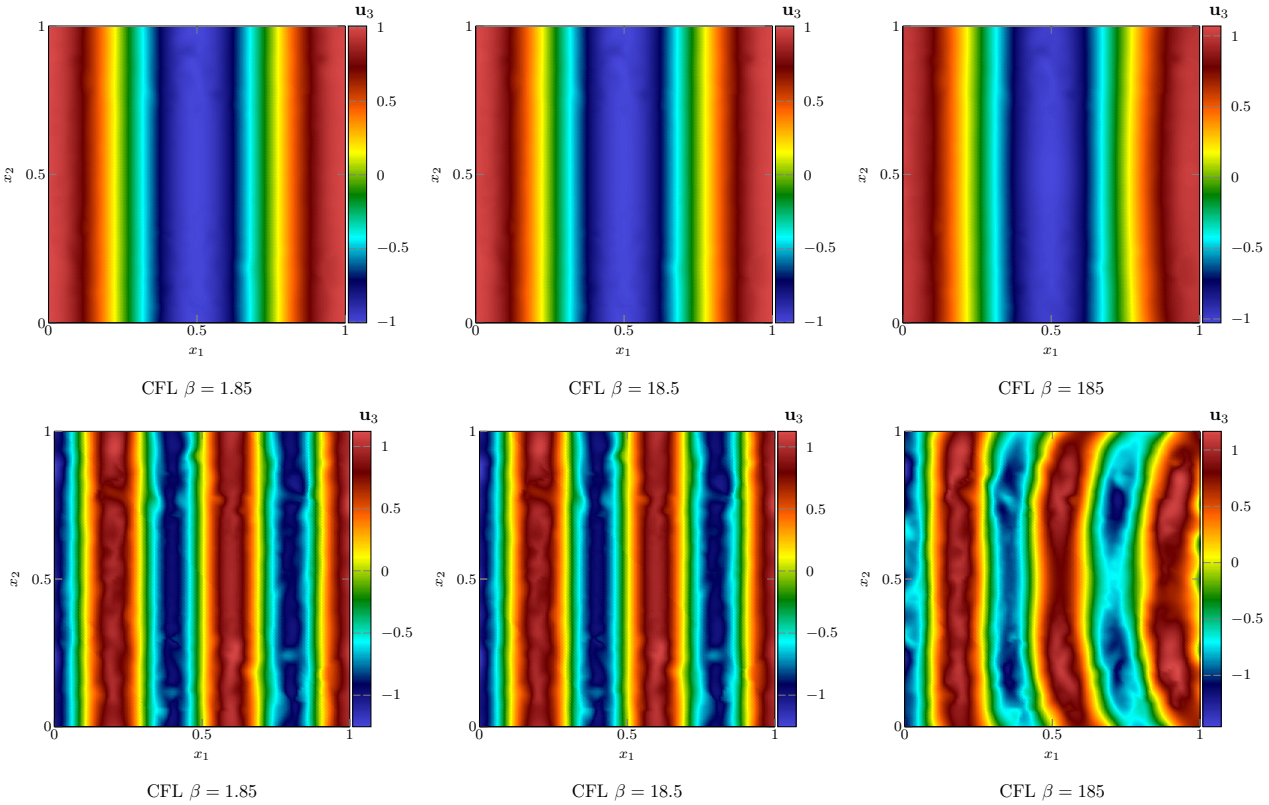


Figure 6.8. Maxwell test case from section 6.3: graphical representation of the third component $\mathbf{u}_3(\mathbf{x}, t) = E_3(\mathbf{x}, t)$ of the approximate solution, sliced at $x_3 = 0.5$, using the second mesh $\mathcal{M}_2$. Top panels: $\nu = 2$ in $f(s)$; bottom panels: $\nu = 5$ in $f(s)$. From left to right, the CFL coefficients are 1.85, 18.5 and 185.

In this case where the solution depends on space, well-known issues at the domain boundaries occur, see for instance [27, 28]. To avoid such issues, we temporarily modify the definition (6.2) of the pointwise error, used in the global error formula (6.1), as follows:

$$e_{k,L} = \begin{pmatrix} \left(u^n_{k,L,0} - u^{\text{inc}}_k(\mathbf{x}_{L,0}, t_{\text{end}})\right) g(\mathbf{x}_{L,0}, 0) \\ \vdots \\ \left(u^n_{k,L,9} - u^{\text{inc}}_k(\mathbf{x}_{L,9}, t_{\text{end}})\right) g(\mathbf{x}_{L,9}, 0) \end{pmatrix},$$

where the function g is given by (6.6), with $\mathbf{x}_0 = (0.5, 0.5, 0.5)^\intercal$ and $\eta = 0.5$. This slight modification allows us to ensure that the error is zero close to the boundaries, and thus to only measure the error within a sphere of diameter η centred at the domain center.

The errors are reported in table 6.5 and error lines are displayed in figure 6.9. In both cases, we observe a second-order convergence in time.

6.4.2. *Simulation of an antenna.* For this simulation, we consider a conductive antenna inside a vacuum. The antenna, a cylinder of length 0.5 and diameter 0.05, is placed inside the cubic domain $\mathcal{D} = [0, 1]^3$, and it is oriented along the x_3-axis. The mesh is coarse in the vacuum domain, and refined around the antenna. For more clarity, the geometry and mesh are depicted in figure 6.10. Note that the ratio between the largest and smallest cell size is about 40, with the cell size defined by (5.2). To represent a conductive antenna, we take $\sigma \neq 0$ in the antenna and $\sigma = 0$ elsewhere.

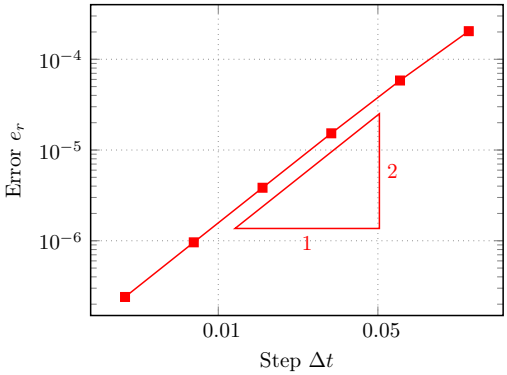
Table 6.5. Maxwell test case with conductivity from section 6.4.1: study of the order of accuracy in time. The tables contain the values of the error e_r, as well as the values of the temporal order of accuracy κ with respect to the time step Δt. Left table: ψ_1 and ψ_2 given by (6.9); right table: ψ_1 and ψ_2 given by (6.10).

Step Δt	Error e_r	Order κ
0.125	$2.041 \cdot 10^{-4}$	—
0.0625	$5.834 \cdot 10^{-5}$	1.8064
0.03125	$1.528 \cdot 10^{-5}$	1.9329
0.015625	$3.839 \cdot 10^{-6}$	1.9928
0.007813	$9.607 \cdot 10^{-7}$	1.9985
0.003906	$2.401 \cdot 10^{-7}$	2.0004

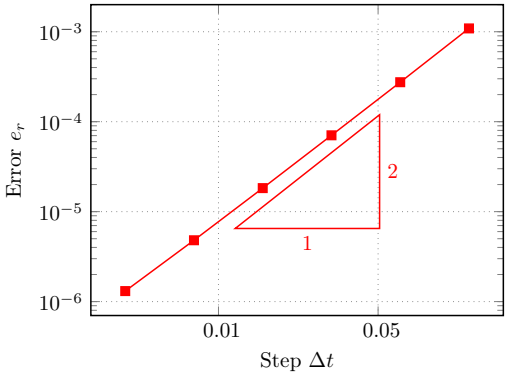
(a) Temporal convergence, case (6.9).

Step Δt	Error e_r	Order κ
0.125	$1.088 \cdot 10^{-3}$	—
0.0625	$2.746 \cdot 10^{-4}$	1.9869
0.03125	$7.055 \cdot 10^{-5}$	1.9605
0.015625	$1.826 \cdot 10^{-5}$	1.9500
0.007813	$4.806 \cdot 10^{-6}$	1.9257
0.003906	$1.308 \cdot 10^{-6}$	1.8773

(b) Temporal convergence, case (6.10).



(a) Temporal convergence, case (6.9).



(b) Temporal convergence, case (6.10).

Figure 6.9. Maxwell test case with conductivity from section 6.4.1: plot in `loglog` scale of the numerical convergence rates. The error e_r is displayed with respect to the time step Δt. Left panel: ψ_1 and ψ_2 given by (6.9); right panel: ψ_1 and ψ_2 given by (6.10).

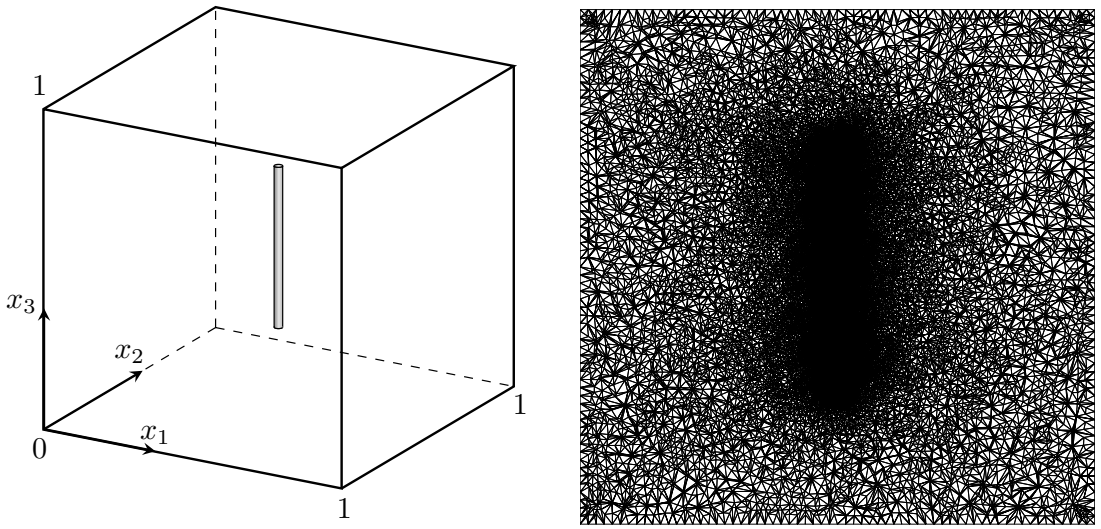


Figure 6.10. Antenna simulation from section 6.4.2: depiction of the geometry (left panel) and the mesh sliced at $x_1 = 0.5$ (right panel).

We take the following exact solution of Maxwell's equations (6.3) with $\sigma = 0$:

$$\mathbf{u}_{\mathrm{w}}^{\mathrm{inc}}(\mathbf{x}, t) = \begin{pmatrix} 0 \\ 0 \\ \psi(x_2 - x_c - t) \\ \psi(x_2 - x_c - t) \\ 0 \\ 0 \end{pmatrix}, \tag{6.11}$$

where ψ is the following compactly supported function (i.e. bump function):

$$\psi(x) = \begin{cases} \exp\left(1 - \dfrac{1}{1 - \frac{|x|}{\eta}}\right) & \text{if } |x| < \eta, \\ 0 & \text{otherwise,} \end{cases}$$

with $\eta = 0.25$ the diameter of the bump (i.e. the size of the support) and $x_c = -0.25$ the initial location of the bump. Note that, initially, the bump is out of the space domain $\mathcal{D} = [0,1]^3$. This allows us to have initially a vanishing exact solution inside the domain, before its perturbation by the incoming wave. The goal of this experiment is to study the interaction of this incoming wave with the antenna. Indeed, (6.11) is no longer an exact solution of (6.3) as soon as $\sigma \neq 0$, which will highlight the conductive behavior of the antenna. Small distortions due to the mesh size can be seen in the figures. This is caused by two reasons: first, the mesh is quite coarse away from the antenna; second, nonzero data span a small interval around 0, making a certain degree of numerical fluctuations appear in the figures.

The CFL number β is set to 20, which is about 11 times larger than the CFL prescribed by the fully explicit DG scheme. Further decreasing the value of β does not lead to a better approximation, which once again demonstrates the relevance of our CFL-less procedure in meshes with large variations in cell size. We show results for a non-stiff ($\sigma \sim 1$) and stiff ($\sigma \gg 1$) source term.

6.4.2.1. *Non-stiff source term.* We start with a non-stiff source term. The numerical results with $\sigma = 5$ in the antenna are reported in figure 6.11, and they show good agreement with the expected results, especially at such large values of the CFL number.

In figure 6.11a, we take the final time to be $t_{\text{end}} = 0.75$, i.e. the peak of the wave is on the antenna. In the left and middle panels, we display E_1 and E_2, i.e. the electric field in the x_1 and x_2 directions, and the right panel contains the magnetic field H_2 in the x_2 direction. As expected, we clearly observe that the magnetic field rotates around the antenna, which creates two electric charges of opposite signs, concentrated at the ends of the antenna.

In figures 6.11b and 6.11c, we respectively take $t_{\text{end}} = 1.5$ and $t_{\text{end}} = 15$, which means that the wave has left the domain. From left to right, we depict E_1, E_2 and E_3. As expected, we observe that the electric charge tends to return to equilibrium as time advances. In these panels, we do not display the magnetic field, which is zero (up to numerical errors) in the whole domain.

To further study the return to equilibrium, we display in figure 6.12, with respect to time, the value of E_3 in the mesh element closest to the center of the antenna. We observe that initial impulsion reaches the antenna for $0.25 \leq t \leq 0.75$; this impulsion, which elevates E_3 to 1, is not fully represented in figure 6.12 for scaling reasons. Then, when the wave has passed the antenna, we observe a return to equilibrium for $t \geq 0.75$. This relaxation towards the equilibrium $E_3 = 0$ is slower for larger values of σ.

6.4.2.2. *Stiff source term.* We now consider a stiff source term. Namely, we take $\sigma = 10^{12}$ to simulate a Perfect Electric Conductor (PEC). In this case, the electromagnetic field should not penetrate the antenna. Note that the stiffness induced by the source term is not translated into a stiffness in the numerical scheme. Indeed, recall that the only place the source term appears in the scheme is via (4.3), which yields, in the present case:

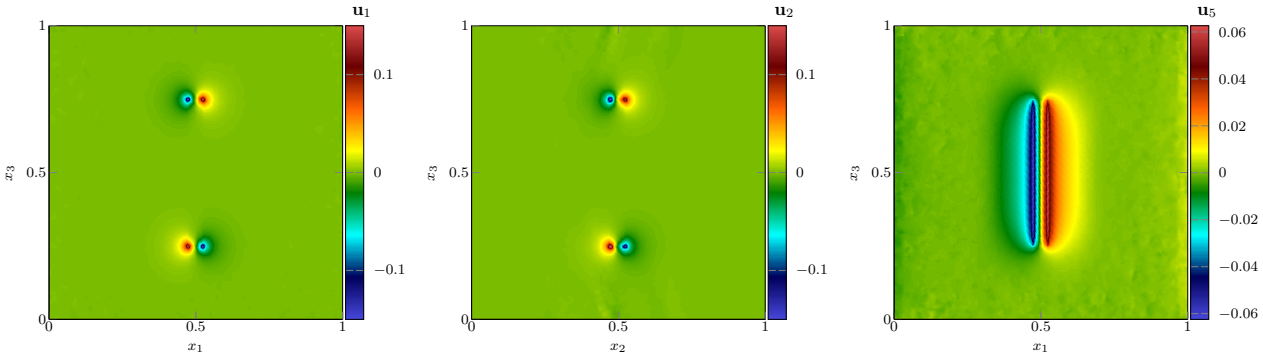
$$\begin{cases} E^{n+1} = \mu E^n, \\ H^{n+1} = H^n, \end{cases} \quad \text{where} \quad \mu = \frac{1 - \sigma\dfrac{\Delta t}{2}}{1 + \sigma\dfrac{\Delta t}{2}}.$$

When the source term is stiff, i.e. $\sigma \to \infty$, we get $\mu \to -1$, and thus $E^{n+1} \to -E^n$, which simulates the correct behavior in the PEC without the need for any additional numerical treatment.

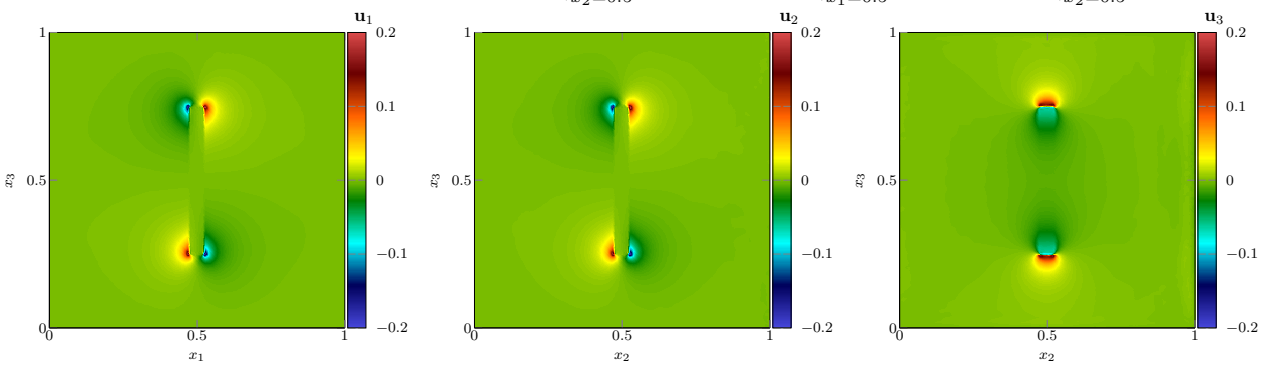
This observation is confirmed in figure 6.13, where we indeed observe that the electric field remains zero inside the antenna (left and middle panels). In addition, we note in the right panel that the incoming magnetic field – the right-traveling wave, which comes from the left of the domain, being clearly visible to the right of the antenna – did not induce a magnetic field within the antenna.

7. Performance and parallelism

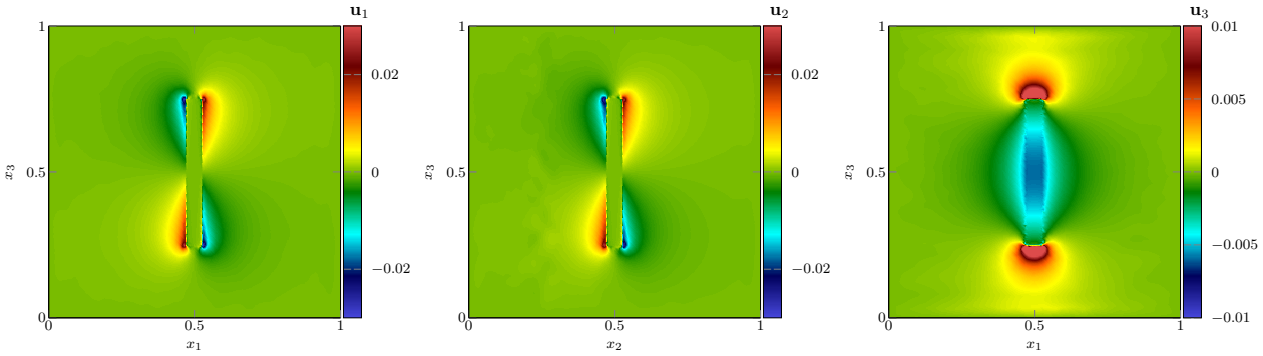
In this section, we evaluate the performance and parallel scalability of the algorithm described in section 3. We seek to compare the `D3Q4` and `D3Q4P` implementations, as described in section 3.4.



(a) Solution at time $t = 0.75$; left panel: $E_1\big|_{x_2=0.5}$; middle panel: $E_2\big|_{x_1=0.5}$; right panel: $H_2\big|_{x_2=0.5}$.



(b) Solution at time $t = 1.5$; left panel: $E_1\big|_{x_2=0.5}$; middle panel: $E_2\big|_{x_1=0.5}$; right panel: $E_3\big|_{x_1=0.5}$.



(c) Solution at time $t = 15$; left panel: $E_1\big|_{x_2=0.5}$; middle panel: $E_2\big|_{x_1=0.5}$; right panel: $E_3\big|_{x_2=0.5}$.

Figure 6.11. Antenna simulation from section 6.4.2: depictions of the solution at times $t = 0.75$, $t = 1.5$ and $t = 15$ (from top to bottom). Note that the color scale, as well as the slicing plane, are not shared by the nine plots.

To that end, we first compare in section 7.1 the memory footprint and the parallel scalability of the two algorithms. Then, in section 7.2, we discuss two ways to order the mesh graph, and their effects on parallel efficiency.

These comparisons are made on a server with an Intel Xeon E5-2680 v3 processor (24 physical cores, 2.50 GHz). We consider tetrahedral meshes of the unit cube $\mathcal{D} = [0, 1]^3$, represented by a single integer, called the "refinement", which is nothing but the number of tetrahedra on each side of the unit cube. These meshes are once again generated using `gmsh`.

7.1. **Scalability and memory footprint.** Recall that, although the `D3Q4` and `D3Q4P` implementations solve the same problem, they have different specificities. For the sake of completeness, we recall them in table 3.1.

We expect the `D3Q4P` implementation, thanks to its superior scalability, to be much faster than the `D3Q4` one on larger meshes and with a larger number of threads. Also, we expect the memory footprint of the `D3Q4P` implementation to be lower than that `D3Q4` one, since it is not necessary to store the global DG matrix.

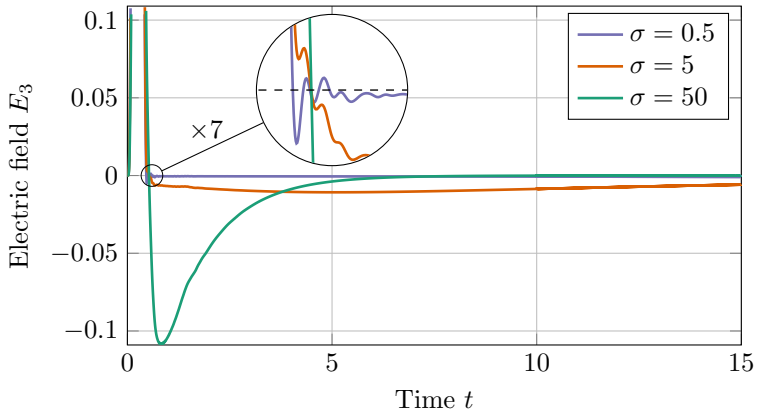


Figure 6.12. Antenna simulation from section 6.4.2: graph of E_3 with respect to time. We observe a relaxation towards the equilibrium $E_3 = 0$ as time increases, quicker for smaller values of σ inside the antenna. The zoom displays the quick relaxation when $\sigma = 0.5$ in the antenna; in the zoomed in graph, the dashed line represents $E_3 = 0$, and the minimum of E_3 is around 5×10^{-3}.

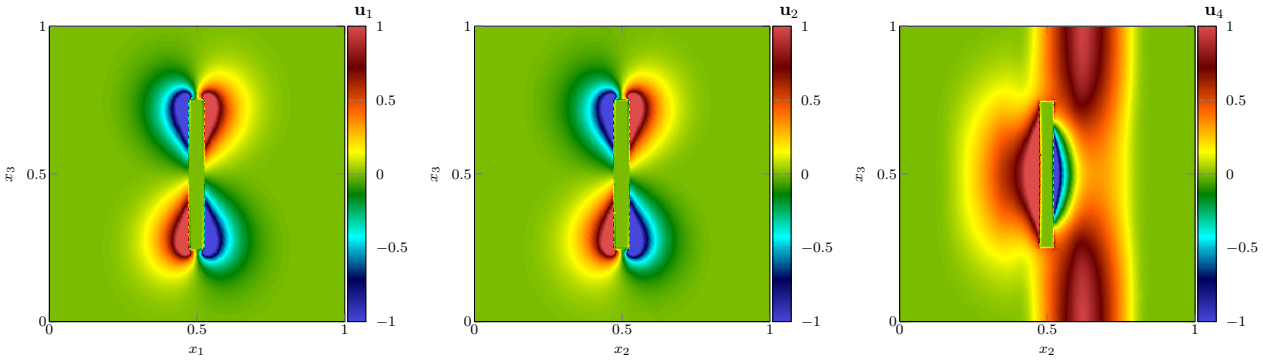


Figure 6.13. Antenna simulation from section 6.4.2 with a stiff source term ($\sigma = 10^{12}$): depictions of the solution at time $t = 0.375$ (left panel: $E_1\big|_{x_2=0.5}$; middle panel: $E_2\big|_{x_1=0.5}$; right panel: $H_1\big|_{x_1=0.5}$).

Remark. *Note that, to improve the CPU time of the serial implementation of* `D3Q4P`*, it is possible to store the local matrices instead of computing them on the fly. This drastically increases the memory footprint (since* $d + 1 = 4$ *matrices of size* 10×10 *have to be stored for each tetrahedron), but also decreases the CPU time of the serial code. However, in the context of the parallel implementation, storing these matrices barely decreases the CPU time, since it also decreases the work done by each thread. Therefore, storing the local matrices in the* `D3Q4P` *implementation is not useful, since it only increases memory consumption for a small decrease in CPU time.*

These expectations are confirmed in table 7.1, where we compare the `D3Q4` and `D3Q4P` methods for refinement levels 8, 16, 32 and 48, on the Maxwell experiment from section 6.2.1 and with 1000 time steps. We limit the `D3Q4P` method to 4 threads for a fair comparison with the `D3Q4` implementation. We remark that the heap memory footprint of the `D3Q4P` implementation is much lower than that of the `D3Q4` method, as expected. The CPU time taken by the serial version of the `D3Q4P` is also lower than that of the `D3Q4` method, which may be explained by more cache misses due to having to recover the global DG matrix in memory. This would also explain the better scalability of the `D3Q4P` method.

To further explore this scalability, we turn to table 7.2, where we compare the serial and parallel versions of the `D3Q4P` implementation, this time not limited to 4 threads but using the full 24 physical threads. We first note that the `D3Q4` implementation consumes about 10 times more heap memory than the `D3Q4P` method for larger meshes. We also remark that, although the parallel code runs much faster than the serial code, the scalability is not perfect on 24 threads. This is possibly due to the fact that tasks have to be distributed to the parallel threads at the start of each parallel region. Therefore, since larger meshes contain larger parallel regions, we expected the scalability to increase as the mesh refinement increases. We study the effects of the graph ordering algorithm on the size of the parallel regions in the next section.

Table 7.1. CPU times and memory footprints, for several refinements levels; comparison between the serial and parallel versions of the `D3Q4` and `D3Q4P` codes, with 4 threads.

refinement		`D3Q4` (it/s)			`D3Q4P` (it/s)			heap (MB)	
level	elements	serial	parallel	scalability	serial	parallel	scalability	`D3Q4`	`D3Q4P`
8	1808	24.67	55.55	2.251	72.58	231.9	3.195	21.41	11.85
16	9199	4.87	7.69	1.579	11.34	44.88	3.958	389.6	42.50
32	56967	0.69	1.029	1.491	1.698	5.938	3.497	2483	266.5
48	175138	0.22	0.334	1.518	0.531	1.859	3.501	7554	808.9

Table 7.2. CPU times, for several refinements levels; comparison between the serial and parallel versions of the `D3Q4P` code, with 24 threads.

refinement		it/s		µs/dof/it			
level	elements	serial	parallel	serial	parallel	scalability	heap
8	1808	72.58	346.1	0.425	0.089	4.769	11.85 MB
16	9199	11.34	102.2	0.569	0.063	9.012	42.50 MB
32	56967	1.698	20.19	0.664	0.056	11.89	266.5 MB
48	175138	0.531	7.753	0.718	0.049	14.60	808.9 MB
64	386806	0.236	3.579	0.747	0.049	15.17	1.777 GB
72	544030	0.165	2.531	0.765	0.050	15.34	2.515 GB

7.2. **Graph ordering.** According to section 3.2.3, one has to choose an algorithm to perform the topological sort of the graph $\mathcal{G}$. Two approaches have been discussed: Breadth-First Search (BFS) and Depth-First Search (DFS). Intuitively, the BFS seems more suited to the current problem. In the example of figure 3.3, we display an example of topological sorts based on BFS and DFS in figure 7.1, as well as the parallel regions resulting from these orderings. It is clear that the DFS ordering, below the BFS ordering, is not well suited at all to the current problems, since it results in more, smaller parallel regions. This remark is confirmed in table 7.3, where we display the number of parallel regions, as well as the average and the maximal number of cells in each region, resulting from BFS and DFS orderings of the meshes already deployed in this section. The DFS ordering is computed with Kosaraju's algorithm from [1], implemented in the `petgraph`[2] library of the Rust language, while the BFS ordering is computed thanks to Kahn's algorithm from [35].

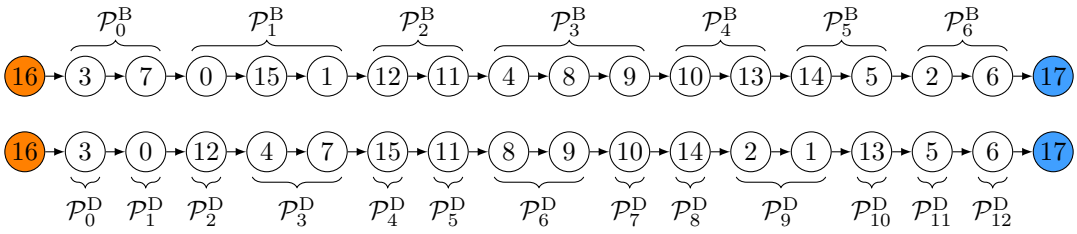


Figure 7.1. Comparison of Breadth-First Search (BFS, above) and Depth-First Search (DFS, below) orderings of the graph from figure 3.3. For each ordering, the regions that can be treated in parallel are displayed; $\mathcal{P}_p^{\mathrm{B}}$ represents the parallel region stemming from the BFS order, while $\mathcal{P}_p^{\mathrm{D}}$ represents the ones obtained with the DFS order. It is apparent that BFS is more suited to the current graph, since it yields fewer and larger parallel regions.

8. Conclusion and further work

In this work, we have proposed a new Discontinuous Galerkin (DG) approximation of a hyperbolic system, based on a vectorial kinetic representation. The time stepping is quasi-explicit but is not constrained by a CFL stability condition. We have applied this method to Maxwell's equations and to the wave equation.

[2] https://docs.rs/petgraph

Table 7.3. Distribution of parallel regions with respect to the topological sorting algorithm used. We present the results using the first kinetic velocity $v_0 = (1, 1, 1)$, but the conclusions are similar with the other three. We have denoted by n_p the number of parallel regions, $n_p^{\max}$ the number of elements in the largest parallel region, and n_p^{avg} the average number of elements in a parallel region.

refinement		BFS order			DFS order		
level	elements	n_p	n_p^{avg}	$n_p^{\max}$	n_p	n_p^{avg}	$n_p^{\max}$
8	1808	73	24	42	1331	1	5
16	9199	152	60	130	7026	1	5
32	56967	299	190	431	43695	1	6
48	175138	444	394	934	135021	1	6
64	386806	592	653	1475	298925	1	8
72	544030	676	804	1939	420828	1	7

We have proposed two implementations of the kinetic DG method. The first is based on a classical assembly of the linear system of the DG transport matrix; this linear system is triangular and can be solved by an adequate matrix solver, such as KLU. The second implementation is based on a downwind algorithm that is more memory efficient and has good parallelization opportunities.

In several numerical experiments, we have shown that the method has the expected features: excellent parallel efficiency, low memory footprint like an explicit scheme, and good accuracy. The main application of the method is to treat the case of meshes with small cells to compute low frequency solutions.

There remains several extensions to be explored in order for our approach to be fully useful in practical applications. First it would be interesting to couple it with another classical explicit RKDG method. This would make it possible to couple an explicit scheme, in regions where the cells are large, to the kinetic method, in refined regions. Other points that still deserve some work are related to the treatment of source terms and boundary conditions. Some strategies to deal with these issues are discussed in [21, 25]. Finally, another important improvement is related to parallelism. For the moment, the method is well adapted to shared-memory multiprocessing (generally addressed by OpenMP). The particular structure of the downwind algorithm makes it difficult to extend it efficiently to distributed-memory multiprocessing (generally addressed by MPI). To be able to use MPI, we could for instance adopt a hybrid Implicit-Explicit (IMEX) strategy such as the one given in [23].

Acknowledgement

The authors extend their thanks to the ANR-20-ASTC-0028 M2CEM for financial support.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data structures and algorithms*. Addison-Wesley, Reading, Mass, 1983.
[2] D. Aregba-Driollet and R. Natalini. Discrete Kinetic Schemes for Multidimensional Systems of Conservation Laws. *SIAM J. Numer. Anal.*, 37(6):1973–2004, 2000.
[3] U. Ayachit. *The ParaView guide : updated for ParaView version 4.3*. Kitware, Clifton Park, New York, 2015.
[4] J. Badwaik, M. Boileau, D. Coulette, E. Franck, Ph. Helluy, C. Klingenberg, L. Mendoza, and H. Oberlin. Task-Based Parallelization of an Implicit Kinetic Scheme. *ESAIM: Proceedings and Surveys*, 63:60–77, 2018.
[5] M. Baudin, C. Berthon, F. Coquel, R. Masson, and Q. H. Tran. A relaxation method for two-phase flow models with hydrodynamic closure law. *Numer. Math.*, 99(3):411–440, 2004.
[6] L. Berardocco, M. Kronbichler, and V. Gravemeier. A hybridizable discontinuous Galerkin method for electromagnetics with a view on subsurface applications. *Comput. Methods Appl. Mech. Engrg.*, 366:113071, 2020.
[7] J. D. Booth, N. D. Ellingwood, H. K. Thornquist, and S. Rajamanickam. Basker: Parallel sparse LU factorization utilizing hierarchical parallelism and data layouts. *Parallel Comput.*, 68:17–31, 2017.
[8] F. Bouchut. Construction of BGK Models with a Family of Kinetic Entropies for a Given System of Conservation Laws. *J. Stat. Phys.*, 95(1/2):113–170, 1999.
[9] F. Bouchut. A reduced stability condition for nonlinear relaxation to conservation laws. *J. Hyperbolic Differ. Equ.*, 01(01):149–170, 2004.
[10] F. Bourdel, P.-A. Mazet, and Ph. Helluy. Resolution of the non-stationary or harmonic Maxwell equations by a discontinuous finite element method. Application to an E.M.I. (electromagnetic impulse) case. In *10th international conference on computing methods in applied sciences and engineering*, pages 405–422. Nova Science Publishers, Inc. Commack, NY, USA, 1992.
[11] B. Bramas, Ph. Helluy, L. Mendoza, and B. Weber. Optimization of a discontinuous Galerkin solver with OpenCL and StarPU. *Int. J. Finite Vol.*, 15(1):1–19, 2020.

[12] B. Bramas and A. Ketterlin. Improving parallel executions by increasing task granularity in task-based runtime systems using acyclic DAG clustering. *PeerJ Comput. Sci.*, 6:e247, 2020.
[13] N. Castel, G. Cohen, and M. Duruflé. Application of discontinuous Galerkin spectral method on hexahedral elements for aeroacoustic. *J. Comp. Acous.*, 17(02):175–196, 2009.
[14] G.-Q. Chen, C. D. Levermore, and T.-P. Liu. Hyperbolic conservation laws with stiff relaxation terms and entropy. *Comm. Pure Appl. Math.*, 47(6):787–830, 1994.
[15] X. Chen, Y. Wang, and H. Yang. NICSLU: An Adaptive Sparse Matrix Solver for Parallel Circuit Simulation. *IEEE Trans. Comput. Aid. D.*, 32(2):261–274, 2013.
[16] B. Cockburn and C.-W. Shu. The Runge–Kutta Discontinuous Galerkin Method for Conservation Laws V: Multidimensional Systems. *J. Comput. Phys.*, 141(2):199–224, apr 1998.
[17] B. Cockburn and C.-W. Shu. Runge–Kutta discontinuous Galerkin methods for convection-dominated problems. *J. Sci. Comput*, 16(3):173–261, 2001.
[18] G. Cohen and S. Pernet. *Finite Element and Discontinuous Galerkin Methods for Transient Wave Equations*. Scientific Computation. Springer Netherlands, 2017.
[19] F. Coquel and B. Perthame. Relaxation of Energy and Approximate Riemann Solvers for General Pressure Laws in Fluid Dynamics. *SIAM J. Numer. Anal.*, 35(6):2223–2249, 1998.
[20] D. Coulette, C. Courtès, E. Franck, and L. Navoret. Vectorial Kinetic Relaxation Model with Central Velocity. Application to Implicit Relaxations Schemes. *Commun. Comput. Phys.*, 27(4):976–1013, 2020.
[21] D. Coulette, E. Franck, Ph. Helluy, M. Mehrenberger, and L. Navoret. High-order implicit palindromic discontinuous Galerkin method for kinetic-relaxation approximation. *Comput. & Fluids*, 190:485–502, 2019.
[22] T. A. Davis and E. P. Natarajan. Algorithm 907: KLU, a direct sparse solver for circuit simulation problems. *ACM T. Math. Software*, 37(3):1–17, 2010.
[23] S. Descombes, S. Lanteri, and L. Moya. Locally Implicit Time Integration Strategies in a Discontinuous Galerkin Method for Maxwell's Equations. *J. Sci. Comput*, 56(1):190–218, 2012.
[24] V. Dolejší and M. Feistauer. *Discontinuous Galerkin Method*, volume 48 of *Springer Series in Computational Mathematics*. Springer International Publishing, July 2015.
[25] F. Drui, E. Franck, Ph. Helluy, and L. Navoret. An analysis of over-relaxation in a kinetic approximation of systems of conservation laws. *CR Mécanique*, 347(3):259–269, 2019.
[26] F. Dubois. Equivalent partial differential equations of a lattice Boltzmann scheme. *Comput. Math. Appl.*, 55(7):1441–1449, 2008.
[27] L. Einkemmer and A. Ostermann. Overcoming Order Reduction in Diffusion-Reaction Splitting. Part 1: Dirichlet Boundary Conditions. *SIAM J. Sci. Comput.*, 37(3):A1577–A1592, 2015.
[28] L. Einkemmer and A. Ostermann. Overcoming Order Reduction in Diffusion-Reaction Splitting. Part 2: Oblique Boundary Conditions. *SIAM J. Sci. Comput.*, 38(6):A3741–A3757, 2016.
[29] L. Fezoui, S. Lanteri, S. Lohrengel, and S. Piperno. Convergence and stability of a discontinuous Galerkin time-domain method for the 3D heterogeneous Maxwell equations on unstructured meshes. *ESAIM Math. Model. Numer. Anal.*, 39(6):1149–1176, nov 2005.
[30] G. Gassner and D. A. Kopriva. A Comparison of the Dispersion and Dissipation Errors of Gauss and Gauss–Lobatto Discontinuous Galerkin Spectral Element Methods. *SIAM J. Sci. Comput.*, 33(5):2560–2579, 2011.
[31] C. Geuzaine and J.-F. Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *Internat. J. Numer. Methods Engrg.*, 79(11):1309–1331, 2009.
[32] J. Hesthaven and T. Warburton. Discontinuous Galerkin methods for the time-domain Maxwell's equations. *ACES Newsletter*, 19:10–29, 2004.
[33] J. S. Hesthaven and T. Warburton. *Nodal Discontinuous Galerkin Methods*. Springer New York, 2008.
[34] S. Jin and Z. P. Xin. The relaxation schemes for systems of conservation laws in arbitrary space dimensions. *Comm. Pure Appl. Math.*, 48(3):235–276, 1995.
[35] A. B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, 1962.
[36] A. Klöckner, T. Warburton, J. Bridge, and J. S. Hesthaven. Nodal discontinuous Galerkin methods on graphics processors. *J. Comput. Phys.*, 228(21):7863–7882, nov 2009.
[37] D. Komatitsch, G. Erlebacher, D. Göddeke, and D. Michéa. High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster. *J. Comput. Phys.*, 229(20):7692–7714, oct 2010.
[38] P. Lasaint and P. A. Raviart. On a Finite Element Method for Solving the Neutron Transport Equation. In *Mathematical Aspects of Finite Elements in Partial Differential Equations*, pages 89–123. Elsevier, 1974.
[39] V. I. Lebedev. Quadratures on a sphere. *USSR Comp. Math. Math+*, 16(2):10–24, 1976.
[40] R. I. McLachlan and G. R. W. Quispel. Splitting methods. *Acta Numer.*, 11:341–434, jan 2002.
[41] D. A. Di Pietro and A. Ern. *Mathematical Aspects of Discontinuous Galerkin Methods*, volume 69 of *Mathématiques et Applications*. Springer Berlin Heidelberg, 2012.
[42] W. H. Reed and T. R. Hill. Triangular mesh methods for the neutron transport equation. Technical report, Los Alamos Scientific Lab., N. Mex. (USA), 1973.
[43] S. Succi. *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*. Oxford University Press, Oxford New York, 2001.
[44] T. Toulorge and W. Desmet. CFL Conditions for Runge–Kutta discontinuous Galerkin methods on triangular grids. *J. Comput. Phys.*, 230(12):4657–4678, 2011.
[45] G. B. Whitham. *Linear and Nonlinear Waves*. John Wiley & Sons, Inc., 1999.
[46] J.H Williamson. Low-storage Runge-Kutta schemes. *J. Comput. Phys.*, 35(1):48–56, 1980.

Appendix A. Lagrange basis on the reference tetrahedron

We consider the reference tetrahedron whose nodes are numbered following figure 3.2. The coordinates of the nodes, as well as the associated Lagrange basis functions, are given in table A.1.

Table A.1. Point coordinates and Lagrange basis functions on the reference tetrahedron.

point	coordinates	basis function
P_0	$(0,0,0)$	$\varphi_0(x_1,x_2,x_3) = (x_1+x_2+x_3-1)(2x_1+2x_2+2x_3-1)$
P_1	$(1,0,0)$	$\varphi_1(x_1,x_2,x_3) = x_1(2x_1-1)$
P_2	$(0,1,0)$	$\varphi_2(x_1,x_2,x_3) = x_2(2x_2-1)$
P_3	$(0,0,1)$	$\varphi_3(x_1,x_2,x_3) = x_3(2x_3-1)$
P_4	$(\frac{1}{2},0,0)$	$\varphi_4(x_1,x_2,x_3) = -4x_1(x_1+x_2+x_3-1)$
P_5	$(\frac{1}{2},\frac{1}{2},0)$	$\varphi_5(x_1,x_2,x_3) = 4x_1x_2$
P_6	$(0,\frac{1}{2},0)$	$\varphi_6(x_1,x_2,x_3) = -4x_2(x_1+x_2+x_3-1)$
P_7	$(0,0,\frac{1}{2})$	$\varphi_7(x_1,x_2,x_3) = -4x_3(x_1+x_2+x_3-1)$
P_8	$(0,\frac{1}{2},\frac{1}{2})$	$\varphi_8(x_1,x_2,x_3) = 4x_2x_3$
P_9	$(\frac{1}{2},0,\frac{1}{2})$	$\varphi_9(x_1,x_2,x_3) = 4x_3x_1$

Appendix B. Derivation of the discontinuous Galerkin solver for the transport equation

To determine the coefficients $f_{L,i}(t)$ in the discontinuous Galerkin framework, one multiplies the PDE (3.3) by the basis function φ_j^L for all $\forall j \in \{0,\dots,9\}$, before integrating the resulting equation on L. In our case, we get

$$\forall L \in \mathcal{M},\ \forall j \in \{0,\dots,9\},\ \int_L \partial_t f(\mathbf{x},t)\varphi_j^L(\mathbf{x})\,d\mathbf{x} + \int_L \mathbf{v}\cdot\boldsymbol{\nabla} f(\mathbf{x},t)\varphi_j^L(\mathbf{x})\,d\mathbf{x} = 0.$$

Arguing the divergence theorem yields

$$\int_L \partial_t f(\mathbf{x},t)\varphi_j^L(\mathbf{x})\,d\mathbf{x} - \int_L f(\mathbf{x},t)\,\mathbf{v}\cdot\boldsymbol{\nabla}\varphi_j^L(\mathbf{x})\,d\mathbf{x} + \int_{\partial L} f(\boldsymbol{\eta},t)\,\varphi_j^L(\boldsymbol{\eta})\,\mathbf{v}\cdot\mathbf{n}\,d\boldsymbol{\eta} = 0, \tag{B.1}$$

where $\mathbf{n}$ is the outer normal to ∂L and $\boldsymbol{\eta} \in \partial L$.

We now consider the behavior of each integral in (B.1) under the basis expansion (3.4). For the first integral, we get:

$$\begin{aligned}
\int_L \partial_t f(\mathbf{x},t)\varphi_j^L(\mathbf{x})\,d\mathbf{x} &= \int_L \sum_{i=0}^{9} \partial_t f_{L,i}(t)\varphi_i^L(\mathbf{x})\varphi_j^L(\mathbf{x})\,d\mathbf{x} \\
&= \sum_{i=0}^{9} \partial_t f_{L,i}(t)\left(\int_L \varphi_i^L(\mathbf{x})\varphi_j^L(\mathbf{x})\,d\mathbf{x}\right) \\
&\simeq \sum_{i=0}^{9} \frac{f_{L,i}^{n+1} - f_{L,i}^{n}}{\Delta t}\left(\int_L \varphi_i^L(\mathbf{x})\varphi_j^L(\mathbf{x})\,d\mathbf{x}\right),
\end{aligned}$$

where we have set $f_{L,i}^n \coloneqq f_{L,i}(n\Delta t)$, and where we have approximated the time derivative $\partial_t f_{L,i}(t)$ using a first-order implicit Euler discretization. The second integral becomes:

$$\begin{aligned}
\int_L f(\mathbf{x},t)\,\mathbf{v}\cdot\boldsymbol{\nabla}\varphi_j^L(\mathbf{x})\,d\mathbf{x} &= \int_L \sum_{i=0}^{9} f_{L,i}(t)\varphi_i^L(\mathbf{x})\,\mathbf{v}\cdot\boldsymbol{\nabla}\varphi_j^L(\mathbf{x})\,d\mathbf{x} \\
&= \sum_{i=0}^{9} f_{L,i}(t)\left(\int_L \varphi_i^L(\mathbf{x})\,\mathbf{v}\cdot\boldsymbol{\nabla}\varphi_j^L(\mathbf{x})\,d\mathbf{x}\right) \\
&\simeq \sum_{i=0}^{9} f_{L,i}^{n+1}\left(\int_L \varphi_i^L(\mathbf{x})\,\mathbf{v}\cdot\boldsymbol{\nabla}\varphi_j^L(\mathbf{x})\,d\mathbf{x}\right),
\end{aligned}$$

where we have used the implicit Euler discretization to set $f_{L,i}(t) \simeq f_{L,i}((n+1)\Delta t) = f_{L,i}^{n+1}$.

For the third integral, we leverage the fact that the cell L is a tetrahedron with 4 faces. For $\alpha \in \{0, \dots, 3\}$, we denote by ∂L_α the α-th face of L, and by $\mathbf{n}_\alpha$ the outer normal vector to ∂L_α, as depicted in figure 3.1. Therefore, the third integral reads:

$$\int_{\partial L} f(\boldsymbol{\eta}, t)\, \varphi_j^L(\boldsymbol{\eta})\, \mathbf{v} \cdot \mathbf{n}\, d\boldsymbol{\eta} = \sum_{\alpha=0}^{3} \int_{\partial L_\alpha} f(\boldsymbol{\eta}, t)\, \mathbf{v} \cdot \mathbf{n}_\alpha\, \varphi_j^L(\boldsymbol{\eta})\, d\boldsymbol{\eta}. \tag{B.2}$$

We now need to provide an approximation of the flux $f(\boldsymbol{\eta}, t)\, \mathbf{v} \cdot \mathbf{n}_\alpha$ through the face ∂L_α. Denoting by R_α the neighboring tetrahedron of L through the face ∂L_α, we choose the following classical upwind discretization:

$$f(\boldsymbol{\eta}, t)\, \mathbf{v} \cdot \mathbf{n}_\alpha \simeq f_L(\boldsymbol{\eta}, t)\, (\mathbf{v} \cdot \mathbf{n}_\alpha)_+ + f_{R_\alpha}(\boldsymbol{\eta}, t)\, (\mathbf{v} \cdot \mathbf{n}_\alpha)_-,$$

with $(\mathbf{v} \cdot \mathbf{n}_\alpha)_+ = \max(0, \mathbf{v} \cdot \mathbf{n}_\alpha)$ and $(\mathbf{v} \cdot \mathbf{n}_\alpha)_- = \min(0, \mathbf{v} \cdot \mathbf{n}_\alpha)$. Applying this approximation to (B.2), together with the implicit Euler time discretization, yields the following sequence of approximations:

$$\begin{aligned}
\int_{\partial L} f(\boldsymbol{\eta}, t)\, \varphi_j^L(\boldsymbol{\eta})\, \mathbf{v} \cdot \mathbf{n}\, d\boldsymbol{\eta} &\simeq \sum_{\alpha=0}^{3} \int_{\partial L_\alpha} \left[f_L(\boldsymbol{\eta}, t)\, (\mathbf{v} \cdot \mathbf{n}_\alpha)_+ + f_{R_\alpha}(\boldsymbol{\eta}, t)\, (\mathbf{v} \cdot \mathbf{n}_\alpha)_- \right] \varphi_j^L(\boldsymbol{\eta})\, d\boldsymbol{\eta} \\
&= \sum_{\alpha=0}^{3} \int_{\partial L_\alpha} \left[\left(\sum_{i=0}^{9} f_{L,i}(t) \varphi_i^L(\boldsymbol{\eta}) \right) (\mathbf{v} \cdot \mathbf{n}_\alpha)_+ \right. \\
&\qquad \left. + \left(\sum_{i=0}^{9} f_{R_\alpha,i}(t) \varphi_i^{R_\alpha}(\boldsymbol{\eta}) \right) (\mathbf{v} \cdot \mathbf{n}_\alpha)_- \right] \varphi_j^L(\boldsymbol{\eta})\, d\boldsymbol{\eta} \\
&\simeq \sum_{i=0}^{9} f_{L,i}^{n+1} \left[\sum_{\alpha=0}^{3} \left(\int_{\partial L_\alpha} \varphi_i^L(\boldsymbol{\eta}) \varphi_j^L(\boldsymbol{\eta})\, d\boldsymbol{\eta} \right) (\mathbf{v} \cdot \mathbf{n}_\alpha)_+ \right] + \\
&\quad \sum_{i=0}^{9} f_{R_\alpha,i}^{n+1} \left[\sum_{\alpha=0}^{3} \left(\int_{\partial L_\alpha} \varphi_i^{R_\alpha}(\boldsymbol{\eta}) \varphi_j^L(\boldsymbol{\eta})\, d\boldsymbol{\eta} \right) (\mathbf{v} \cdot \mathbf{n}_\alpha)_- \right].
\end{aligned}$$

Plugging these three integral terms into (B.1) yields the DG solver (3.5).