CFL-less and parallel kinetic schemes

Pierre Gerhard[*†], Philippe Helluy[*†], Victor Michel-Dansac[†*], Bruno Weber[‡]

April 21, 2022
Séminaire M2N, **CNAM, Paris**

[*]IRMA, Université de Strasbourg
[†]Université de Strasbourg, CNRS, Inria, IRMA
[‡]AxesSim, Illkirch-Graffenstaden

Vectorial kinetic representation of systems of balance laws

Algorithm for the relaxation-source step

Algorithm for the transport step

Numerical experiments

Distributed-memory parallelization

Conclusion and perspectives

Generic systems of conservation laws

We consider a generic **system of conservation laws** in d space dimensions:

$$\partial_t \mathbf{u} + \sum_{i=1}^{d} \partial_i \mathbf{q}^i(\mathbf{u}) = 0, \quad \mathbf{x} \in \mathbb{R}^d, \quad t > 0,$$

where:

- $\mathbf{u} \in \mathbb{R}^m$ is the vector of m unknowns,
- $\mathbf{q}^i$ are the (smooth) physical flux functions.

For $\mathbf{n} = (n^1, \ldots, n^d) \in \mathbb{R}^d$, the physical flux in direction $\mathbf{n}$ is defined by

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

The homogeneous system is **hyperbolic**, i.e. the Jacobian matrix $\nabla_{\mathbf{u}} \mathbf{q}(\mathbf{u}, \mathbf{n})$ of the flux is diagonalizable with real eigenvalues.

Objectives

This work concerns the derivation of a numerical method to approximate systems of conservation laws, that:

- is high-order accurate,
- is stable without a CFL condition: CFL-less property,
- has the complexity of an explicit scheme,
- can be parallelized.

To achieve this, we select the vectorial kinetic relaxation framework:

- T. Platkowski and R. Illner, (1988), R. Natalini, (1998),
- F. Bouchut, (1999), D. Aregba-Driollet and R. Natalini, (2000),
- **D. Coulette et al., Comput. & Fluids (2019)**,
- ...

Minimal vectorial kinetic representation

The goal is the approximate the (nonlinear) system of conservation laws

$$\partial_t \mathbf{u} + \sum_{i=1}^{d} \partial_i \mathbf{q}^i(\mathbf{u}) = 0, \quad u \in \mathbb{R}^m,$$

when τ goes to 0, with the following (linear) systems of kinetic equations:

$$\forall k \in [\![0, d]\!],\ \partial_t \mathbf{f}_k + \mathbf{v}_k \cdot \boldsymbol{\nabla} \mathbf{f}_k = \frac{1}{\tau}(\mathbf{m}_k(\mathbf{u}) - \mathbf{f}_k),$$

where we have defined a set of constant vectors, the kinetic velocities:

$$\mathcal{V} = (\mathbf{v}_k)_{k \in [\![0,d]\!]}.$$

Minimal vectorial kinetic representation

The goal is the approximate the (nonlinear) system of conservation laws

$$\partial_t \mathbf{u} + \sum_{i=1}^{d} \partial_i \mathbf{q}^i(\mathbf{u}) = 0, \quad u \in \mathbb{R}^m,$$

when τ goes to 0, with the following (linear) systems of kinetic equations:

$$\forall k \in [\![0, d]\!],\ \partial_t \mathbf{f}_k + \mathbf{v}_k \cdot \nabla \mathbf{f}_k = \frac{1}{\tau}(\mathbf{m}_k(\mathbf{u}) - \mathbf{f}_k),$$

where we have defined a set of constant vectors, the kinetic velocities:

$$\mathcal{V} = (\mathbf{v}_k)_{k \in [\![0,d]\!]}.$$

For $k \in [\![0, d]\!]$, we have defined:

- $\mathbf{f}_k \in \mathbb{R}^m$ the $(d+1)$ kinetic unknowns;
- $\mathbf{m}_k : \mathbb{R}^m \to \mathbb{R}^m$ the $(d+1)$ kinetic equilibrium functions;
- τ a (small) relaxation time.

Next step: Derive conditions on $\mathbf{m}_k(\mathbf{u})$, knowing $\mathcal{V}$.

Conditions on the kinetic equilibrium functions

Summing the kinetic equations over k, and taking $\tau \to 0$, we wish to recover the conservation laws.

We first make the following assumption:

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

Conditions on the kinetic equilibrium functions

Summing the kinetic equations over k, and taking $\tau \to 0$, we wish to recover the conservation laws.

We first make the following assumption:

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

We now sum the kinetic equations over k.

$$\partial_t \left(\sum_{k=0}^{d} \mathbf{f}_k \right) + \sum_{k=0}^{d} \mathbf{v}_k \cdot \nabla \mathbf{f}_k = \frac{1}{\tau} \left(\sum_{k=0}^{d} \mathbf{m}_k(\mathbf{u}) - \sum_{k=0}^{d} \mathbf{f}_k \right)$$

$$\implies \partial_t \mathbf{u} + \sum_{k=0}^{d} \mathbf{v}_k \cdot \nabla \mathbf{f}_k = \frac{1}{\tau} \left(\sum_{k=0}^{d} \mathbf{m}_k(\mathbf{u}) - \mathbf{u} \right)$$

To recover the conservation laws, the source term must vanish, and the kinetic equilibrium functions $\mathbf{m}_k$ must satisfy:

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

Conditions on the kinetic equilibrium functions

Formally, taking $\tau \to 0$ in the kinetic equations

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

leads to $\mathbf{f}_k = \mathbf{m}_k(\mathbf{u})$.

Injecting this formal limit in the summed kinetic equations, we get:

$$\partial_t \mathbf{u} + \sum_{k=0}^{d} \mathbf{v}_k \cdot \nabla \mathbf{f}_k = 0 \implies \partial_t \mathbf{u} + \sum_{k=0}^{d} \mathbf{v}_k \cdot \nabla \mathbf{m}_k(\mathbf{u}) = 0.$$

Conditions on the kinetic equilibrium functions

Formally, taking $\tau \to 0$ in the kinetic equations

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

leads to $\mathbf{f}_k = \mathbf{m}_k(\mathbf{u})$.

Injecting this formal limit in the summed kinetic equations, we get:

$$\partial_t \mathbf{u} + \sum_{k=0}^{d} \mathbf{v}_k \cdot \nabla \mathbf{f}_k = 0 \implies \partial_t \mathbf{u} + \sum_{k=0}^{d} \mathbf{v}_k \cdot \nabla \mathbf{m}_k(\mathbf{u}) = 0.$$

With $\mathbf{v}_k = (v_k^i)_{i \in [\![1,d]\!]}$, we obtain:

$$\sum_{k=0}^{d} \mathbf{v}_k \cdot \nabla \mathbf{m}_k(\mathbf{u}) = \sum_{k=0}^{d} \left(\sum_{i=1}^{d} v_k^i \, \partial_i \mathbf{m}_k(\mathbf{u}) \right) = \sum_{i=1}^{d} \partial_i \left(\sum_{k=0}^{d} v_k^i \mathbf{m}_k(\mathbf{u}) \right).$$

Conditions on the kinetic equilibrium functions

Formally, taking $\tau \to 0$ in the kinetic equations

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

leads to $\mathbf{f}_k = \mathbf{m}_k(\mathbf{u})$.

Injecting this formal limit in the summed kinetic equations, we get:

$$\partial_t \mathbf{u} + \sum_{k=0}^{d} \mathbf{v}_k \cdot \nabla \mathbf{f}_k = 0 \implies \partial_t \mathbf{u} + \sum_{k=0}^{d} \mathbf{v}_k \cdot \nabla \mathbf{m}_k(\mathbf{u}) = 0.$$

With $\mathbf{v}_k = (v_k^i)_{i \in [\![1,d]\!]}$, we obtain:

$$\sum_{k=0}^{d} \mathbf{v}_k \cdot \nabla \mathbf{m}_k(\mathbf{u}) = \sum_{k=0}^{d} \left(\sum_{i=1}^{d} v_k^i \, \partial_i \mathbf{m}_k(\mathbf{u}) \right) = \sum_{i=1}^{d} \partial_i \left(\sum_{k=0}^{d} v_k^i \mathbf{m}_k(\mathbf{u}) \right).$$

Using the conservation laws, we get new conditions on $\mathbf{m}_k$:

$$\partial_t \mathbf{u} + \sum_{i=1}^{d} \partial_i \left(\sum_{k=0}^{d} v_k^i \mathbf{m}_k(\mathbf{u}) \right) = 0 \implies \forall i \in [\![1, d]\!], \sum_{k=0}^{d} v_k^i \mathbf{m}_k(\mathbf{u}) = \mathbf{q}^i(\mathbf{u}).$$

Computing the kinetic equilibrium functions

Finally, the kinetic equilibrium functions satisfy:

$$\sum_{k=0}^{d} \mathbf{m}_k(\mathbf{u}) = \mathbf{u} \quad \text{and} \quad \forall i \in [\![1, d]\!],\ \sum_{k=0}^{d} v_k^i \mathbf{m}_k(\mathbf{u}) = \mathbf{q}^i(\mathbf{u}).$$

These conditions can be recast as a $(d+1) \times (d+1)$ linear system, whose unknowns are the $(d+1)$ vectors $\mathbf{m}_k(\mathbf{u}) \in \mathbb{R}^m$:

$$\begin{cases} \mathbf{m}_0(\mathbf{u}) & + & \dots & + & \mathbf{m}_d(\mathbf{u}) & = & \mathbf{u}, \\ v_0^1\, \mathbf{m}_0(\mathbf{u}) & + & \dots & + & v_d^1\, \mathbf{m}_d(\mathbf{u}) & = & \mathbf{q}^1(\mathbf{u}), \\ \vdots & & & & \vdots & & \vdots \\ v_0^d\, \mathbf{m}_0(\mathbf{u}) & + & \dots & + & v_d^d\, \mathbf{m}_d(\mathbf{u}) & = & \mathbf{q}^d(\mathbf{u}). \end{cases}$$

Computing the kinetic equilibrium functions

Finally, the kinetic equilibrium functions satisfy:

$$\sum_{k=0}^{d} \mathbf{m}_k(\mathbf{u}) = \mathbf{u} \quad \text{and} \quad \forall i \in [\![1, d]\!],\ \sum_{k=0}^{d} v_k^i \mathbf{m}_k(\mathbf{u}) = \mathbf{q}^i(\mathbf{u}).$$

These conditions can be recast as a $(d+1) \times (d+1)$ linear system, whose unknowns are the $(d+1)$ vectors $\mathbf{m}_k(\mathbf{u}) \in \mathbb{R}^m$:

$$(\mathbf{m}_0(\mathbf{u}), \mathbf{m}_1(\mathbf{u}), \dots, \mathbf{m}_d(\mathbf{u}))\, \mathbb{V} = (\mathbf{u}, \mathbf{q}^1(\mathbf{u}), \dots, \mathbf{q}^d(\mathbf{u})),$$

where the matrix $\mathbb{V} \in \mathcal{M}_{d+1}(\mathbb{R})$, assumed to be invertible, is defined by:

$$\mathbb{V} = \begin{pmatrix} 1 & v_0^1 & \dots & v_0^d \\ 1 & v_1^1 & \dots & v_1^d \\ \vdots & \vdots & & \vdots \\ 1 & v_d^1 & \dots & v_d^d \end{pmatrix}.$$

An example in 3D

We take the following velocities, called the D3Q4 model[1]:

$$\mathbf{v}_0 = \lambda \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad \mathbf{v}_1 = \lambda \begin{pmatrix} 1 \\ -1 \\ -1 \end{pmatrix}, \quad \mathbf{v}_2 = \lambda \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix}, \quad \mathbf{v}_3 = \lambda \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix},$$

which leads to

$$\mathbb{V} = \begin{pmatrix} 1 & \lambda & \lambda & \lambda \\ 1 & \lambda & -\lambda & -\lambda \\ 1 & -\lambda & \lambda & -\lambda \\ 1 & -\lambda & -\lambda & \lambda \end{pmatrix} \Longrightarrow \mathbb{V}^{-1} = \frac{1}{4\lambda} \begin{pmatrix} \lambda & \lambda & \lambda & \lambda \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}.$$

Solving the linear system yields the kinetic equilibrium functions:

$$\mathbf{m}_k(\mathbf{u}) = \frac{1}{4}\mathbf{u} + \frac{1}{4\lambda^2}\mathbf{q}(\mathbf{u}, \mathbf{v}_k).$$

[1] These velocities are orthogonal to the faces of a tetrahedron; we take $\lambda = \sqrt{3}$ for the unit sphere to be included in this tetrahedron (subcharacteristic condition).

Summary

We have successfully provided a vectorial kinetic relaxation approximation of the system of conservation laws:

$$\partial_t \mathbf{u} + \sum_{i=1}^{d} \partial_i \mathbf{q}^i(\mathbf{u}) = 0 \quad \leftrightsquigarrow \quad \forall k \in [\![0, d]\!],\ \partial_t \mathbf{f}_k + \mathbf{v}_k \cdot \boldsymbol{\nabla} \mathbf{f}_k = \frac{1}{\tau}(\mathbf{m}_k(\mathbf{u}) - \mathbf{f}_k).$$

Remark: Although the kinetic representation is linear, it contains $(d+1) \times m$ unknowns instead of m, as well as a relaxation source term.

Summary

We have successfully provided a vectorial kinetic relaxation approximation of the system of conservation laws:

$$\partial_t \mathbf{u} + \sum_{i=1}^{d} \partial_i \mathbf{q}^i(\mathbf{u}) = 0 \quad \leftrightsquigarrow \quad \forall k \in [\![0, d]\!],\ \partial_t \mathbf{f}_k + \mathbf{v}_k \cdot \boldsymbol{\nabla} \mathbf{f}_k = \frac{1}{\tau}(\mathbf{m}_k(\mathbf{u}) - \mathbf{f}_k).$$

Remark: Although the kinetic representation is linear, it contains $(d+1) \times m$ unknowns instead of m, as well as a relaxation source term.

Remark: The kinetic system is similar to the BGK approximation in gas dynamics, except that:

- the kinetic unknowns $\mathbf{f}_k$ are vectors and have no real physical meaning,
- there is a finite number of kinetic velocities,
- we are only interested in the limit where τ goes to 0.

Summary

We have successfully provided a vectorial kinetic relaxation approximation of the system of conservation laws:

$$\partial_t \mathbf{u} + \sum_{i=1}^{d} \partial_i \mathbf{q}^i(\mathbf{u}) = 0 \quad \leftrightsquigarrow \quad \forall k \in [\![0, d]\!],\ \partial_t \mathbf{f}_k + \mathbf{v}_k \cdot \boldsymbol{\nabla} \mathbf{f}_k = \frac{1}{\tau}(\mathbf{m}_k(\mathbf{u}) - \mathbf{f}_k).$$

Remark: Although the kinetic representation is linear, it contains $(d+1) \times m$ unknowns instead of m, as well as a relaxation source term.

Remark: The kinetic system is similar to the BGK approximation in gas dynamics, except that:

- the kinetic unknowns $\mathbf{f}_k$ are vectors and have no real physical meaning,
- there is a finite number of kinetic velocities,
- we are only interested in the limit where τ goes to 0.

Next step: What if the original system has a source term?

Generic systems of balance laws

We consider a generic **system of balance laws** in d space dimensions:

$$\partial_t \mathbf{u} + \sum_{i=1}^{d} \partial_i \mathbf{q}^i(\mathbf{u}) = \mathbf{s}(\mathbf{u}), \quad \mathbf{x} \in \mathbb{R}^d, \quad t > 0,$$

where:

- $\mathbf{u} \in \mathbb{R}^m$ is the vector of m unknowns,
- $\mathbf{q}^i$ are the (smooth) physical flux functions,
- $\mathbf{s}$ is the (smooth, potentially stiff) source term.

The homogeneous system is still supposed to be **hyperbolic**.

Minimal vectorial kinetic representation

The goal is the approximate the (nonlinear) system of balance laws

$$\partial_t \mathbf{u} + \sum_{i=1}^{d} \partial_i \mathbf{q}^i(\mathbf{u}) = \mathbf{s}(\mathbf{u}), \quad u \in \mathbb{R}^m,$$

with the following (linear) system of kinetic equations:

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

where $\mathbf{g}_k : \mathbb{R}^m \to \mathbb{R}^m$ are the $(d+1)$ kinetic source terms.

Next step: Derive conditions on $\mathbf{g}_k(\mathbf{u})$.

Computing the kinetic source terms

Recall that the kinetic variables and equilibrium functions satisfy:

$$\sum_{k=0}^{d} \mathbf{f}_k = \sum_{k=0}^{d} \mathbf{m}_k(\mathbf{u}) = \mathbf{u} \quad \text{and} \quad \forall i \in [\![1, d]\!], \sum_{k=0}^{d} v_k^i \mathbf{m}_k(\mathbf{u}) = \mathbf{q}^i(\mathbf{u}).$$

Summing the kinetic equations over k and taking the formal $\tau \to 0$ limit, we get:

$$\partial_t \left(\sum_{k=0}^{d} \mathbf{f}_k \right) + \sum_{k=0}^{d} \mathbf{v}_k \cdot \nabla \mathbf{f}_k = \sum_{k=0}^{d} \mathbf{g}_k(\mathbf{u}) + \frac{1}{\tau} \left(\sum_{k=0}^{d} \mathbf{m}_k(\mathbf{u}) - \sum_{k=0}^{d} \mathbf{f}_k \right),$$

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

For the kinetic equations to coincide with the balance laws, we need:

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

Computing the kinetic source terms

A Chapman-Enskog expansion performed in [D. Coulette et al., (2019)] shows that taking

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

cancels out some first-order terms.

In addition, we get

$$\begin{aligned}
\sum_{k=0}^{d} \mathbf{g}_k(\mathbf{u}) &= \left(\sum_{k=0}^{d} \nabla_{\mathbf{u}} \mathbf{m}_k(\mathbf{u}) \right) \mathbf{s}(\mathbf{u}) \\
&= \nabla_{\mathbf{u}} \left(\sum_{k=0}^{d} \mathbf{m}_k(\mathbf{u}) \right) \mathbf{s}(\mathbf{u}) \\
&= (\nabla_{\mathbf{u}} \mathbf{u})\, \mathbf{s}(\mathbf{u}) = \mathbf{s}(\mathbf{u}),
\end{aligned}$$

and the consistency condition is satisfied.

Summary

We have successfully provided a vectorial kinetic relaxation approximation

$$\forall k \in [\![0, d]\!],\ \partial_t \mathbf{f}_k + \mathbf{v}_k \cdot \boldsymbol{\nabla} \mathbf{f}_k = \mathbf{g}_k(\mathbf{u}) + \frac{1}{\tau}(\mathbf{m}_k(\mathbf{u}) - \mathbf{f}_k)$$

of the system of balance laws

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

Next step: Propose a numerical scheme to approximate the solutions of the vectorial kinetic representation.

We proceed with a splitting method:

- first, we treat the transport step $\partial_t \mathbf{f}_k + \mathbf{v}_k \cdot \boldsymbol{\nabla} \mathbf{f}_k = 0$;
- then, the relaxation-source step $\partial_t \mathbf{f}_k = \mathbf{g}_k(\mathbf{u}) + \frac{1}{\tau}(\mathbf{m}_k(\mathbf{u}) - \mathbf{f}_k)$.

Updating the physical variables

We seek a numerical method to approximate solutions to the equation

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

i.e. to find $\mathbf{f}_k^{n+1}$ at time t^{n+1}, knowing $\mathbf{f}_k^n$ and $\mathbf{u}^n$ at time t^n.

Updating the physical variables

We seek a numerical method to approximate solutions to the equation

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

i.e. to find $\mathbf{f}_k^{n+1}$ at time t^{n+1}, knowing $\mathbf{f}_k^n$ and $\mathbf{u}^n$ at time t^n.

First, summing this equation over k leads to

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

which yields, after arguing the properties of $\mathbf{f}_k$, $\mathbf{m}_k$ and $\mathbf{g}_k$:

$$\partial_t \mathbf{u} = \mathbf{s}(\mathbf{u}).$$

Applying the Crank-Nicolson time discretization leads to:

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

Over-relaxation

For clarity, we temporarily take $\mathbf{g}_k = 0$ and simplify the relaxation step:

$$\partial_t f = \frac{1}{\tau}(m(u) - f).$$

Over-relaxation

For clarity, we temporarily take $\mathbf{g}_k = 0$ and simplify the relaxation step:

$$\partial_t f = \frac{1}{\tau}(m(u) - f).$$

The first-order implicit Euler discretization yields:

$$f^{n+1} = f^n + \frac{\Delta t}{\tau}(m(u^{n+1}) - f^{n+1}),$$

$$f^{n+1} = \frac{1}{1 + \dfrac{\Delta t}{\tau}} f^n + \frac{\dfrac{\Delta t}{\tau}}{1 + \dfrac{\Delta t}{\tau}} m(u^{n+1}),$$

$$f^{n+1} = (1 - \omega) f^n + \omega\, m(u^{n+1}), \quad \text{with } \omega \xrightarrow[\tau \to 0]{} 1.$$

Over-relaxation

For clarity, we temporarily take $\mathbf{g}_k = 0$ and simplify the relaxation step:

$$\partial_t f = \frac{1}{\tau}(m(u) - f).$$

The first-order implicit Euler discretization yields:

$$f^{n+1} = (1-\omega)f^n + \omega\, m(u^{n+1}), \quad \text{with } \omega \xrightarrow[\tau\to 0]{} 1.$$

The second-order Crank-Nicolson discretization yields:

$$f^{n+1} = f^n + \frac{\Delta t}{\tau}\left(\frac{m(u^n) + m(u^{n+1})}{2} - \frac{f^n + f^{n+1}}{2}\right),$$

$$f^{n+1} = \frac{1 - \dfrac{\Delta t}{2\tau}}{1 + \dfrac{\Delta t}{2\tau}} f^n + \frac{\dfrac{\Delta t}{\tau}}{1 + \dfrac{\Delta t}{2\tau}} \frac{m(u^n) + m(u^{n+1})}{2},$$

$$f^{n+1} = (1-\omega)f^n + \omega \frac{m(u^n) + m(u^{n+1})}{2}, \quad \text{with } \omega \xrightarrow[\tau\to 0]{} 2.$$

Over-relaxation

For clarity, we temporarily take $\mathbf{g}_k = 0$ and simplify the relaxation step:

$$\partial_t f = \frac{1}{\tau}(m(u) - f).$$

The first-order implicit Euler discretization yields:

$$f^{n+1} = (1-\omega)f^n + \omega\, m(u^{n+1}), \quad \text{with } \omega \xrightarrow[\tau\to 0]{} 1.$$

The second-order Crank-Nicolson discretization yields:

$$f^{n+1} = (1-\omega)f^n + \omega \frac{m(u^n) + m(u^{n+1})}{2}, \quad \text{with } \omega \xrightarrow[\tau\to 0]{} 2.$$

In both cases, we get

$$f^{n+1} \simeq (1-\omega)f^n + \omega \frac{1}{\Delta t}\int_{t^n}^{t^{n+1}} m(u)\, dt, \quad \text{with } \omega \in \{1, 2\}.$$

Over-relaxation

Based on the value of ω, we get the following behavior:

- $\omega = 1 \implies f \leftarrow m(u)$, (*relaxation*)
- $\omega = 2 \implies f \leftarrow 2m(u) - f$. (*over-relaxation*)

We observe that:

- $\omega = 1 \implies$ first-order $\implies$ diffusion,
- $\omega = 2 \implies$ second-order $\implies$ oscillations.

In practice, we take $\omega = 2 - C\Delta t$ to remain second-order accurate but add some diffusion.

Remark: For Crank-Nicolson, we obtain:

$$\omega = \frac{\frac{\Delta t}{\tau}}{1 + \frac{\Delta t}{2\tau}} = 2 - C\Delta t \implies \tau = \frac{C}{4 - 2C\Delta t}\Delta t^2 = \mathcal{O}(\Delta t^2).$$

Updating the kinetic variables

We go back to the kinetic equations with source term. We apply the Crank-Nicolson time discretization:

$$\frac{\mathbf{f}_k^{n+1} - \mathbf{f}_k^n}{\Delta t} = \frac{\mathbf{g}_k(\mathbf{u}^n) + \mathbf{g}_k(\mathbf{u}^{n+1})}{2} + \frac{1}{\tau}\left(\frac{\mathbf{m}_k(\mathbf{u}^n) + \mathbf{m}_k(\mathbf{u}^{n+1})}{2} - \frac{\mathbf{f}_k^n + \mathbf{f}_k^{n+1}}{2}\right),$$

where $\mathbf{f}_k^n$, $\mathbf{u}^n$ and $\mathbf{u}^{n+1}$ are known and $\mathbf{f}_k^{n+1}$ is unknown.

Updating the kinetic variables

We go back to the kinetic equations with source term. We apply the Crank-Nicolson time discretization:

$$\frac{\mathbf{f}_k^{n+1} - \mathbf{f}_k^n}{\Delta t} = \frac{\mathbf{g}_k(\mathbf{u}^n) + \mathbf{g}_k(\mathbf{u}^{n+1})}{2} + \frac{1}{\tau}\left(\frac{\mathbf{m}_k(\mathbf{u}^n) + \mathbf{m}_k(\mathbf{u}^{n+1})}{2} - \frac{\mathbf{f}_k^n + \mathbf{f}_k^{n+1}}{2}\right),$$

where $\mathbf{f}_k^n$, $\mathbf{u}^n$ and $\mathbf{u}^{n+1}$ are known and $\mathbf{f}_k^{n+1}$ is unknown.

With $\omega = 2 - C\Delta t$ defined above, we obtain

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

Since $\tau = \mathcal{O}(\Delta t^2)$, the final update reads, up to $\mathcal{O}(\Delta t^2)$:

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

Summary

To obtain an approximate solution $\mathbf{f}_k^{n+1}$ to

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

at time t^{n+1} from $\mathbf{f}_k^n$ and $\mathbf{u}^n$, we compute:

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = \frac{\mathbf{s}(\mathbf{u}^n) + \mathbf{s}(\mathbf{u}^{n+1})}{2} \quad \text{and} \quad \mathbf{f}_k^{n+1} = (1-\omega)\mathbf{f}_k^n + \omega \, \frac{\mathbf{m}_k(\mathbf{u}^n) + \mathbf{m}_k(\mathbf{u}^{n+1})}{2}.$$

Summary

To obtain an approximate solution $\mathbf{f}_k^{n+1}$ to

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

at time t^{n+1} from $\mathbf{f}_k^n$ and $\mathbf{u}^n$, we compute:

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = \frac{\mathbf{s}(\mathbf{u}^n) + \mathbf{s}(\mathbf{u}^{n+1})}{2} \quad \text{and} \quad \mathbf{f}_k^{n+1} = (1-\omega)\mathbf{f}_k^n + \omega \, \frac{\mathbf{m}_k(\mathbf{u}^n) + \mathbf{m}_k(\mathbf{u}^{n+1})}{2}.$$

Remark: The expression $\mathbf{g}_k(\mathbf{u}) = (\nabla_{\mathbf{u}} \mathbf{m}_k(\mathbf{u}))\, \mathbf{s}(\mathbf{u})$ of the kinetic source term has not been used: no need to compute the flux Jacobian.

Remark: Finding $\mathbf{u}^{n+1}$ requires solving a potentially nonlinear system.

Next step: Propose an algorithm for the transport step $\partial_t \mathbf{f}_k + \mathbf{v}_k \cdot \nabla \mathbf{f}_k = 0$.

Transport step – setting up the problem

We wish to propose a numerical method to solve the equation

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

i.e. to find $\mathbf{f}_k^{n+1}$ at time t^{n+1}, knowing $\mathbf{f}_k^n$ at time t^n.

For simplicity, we treat the following general transport equation[2] with a constant velocity $\mathbf{v}$, of unknown f:

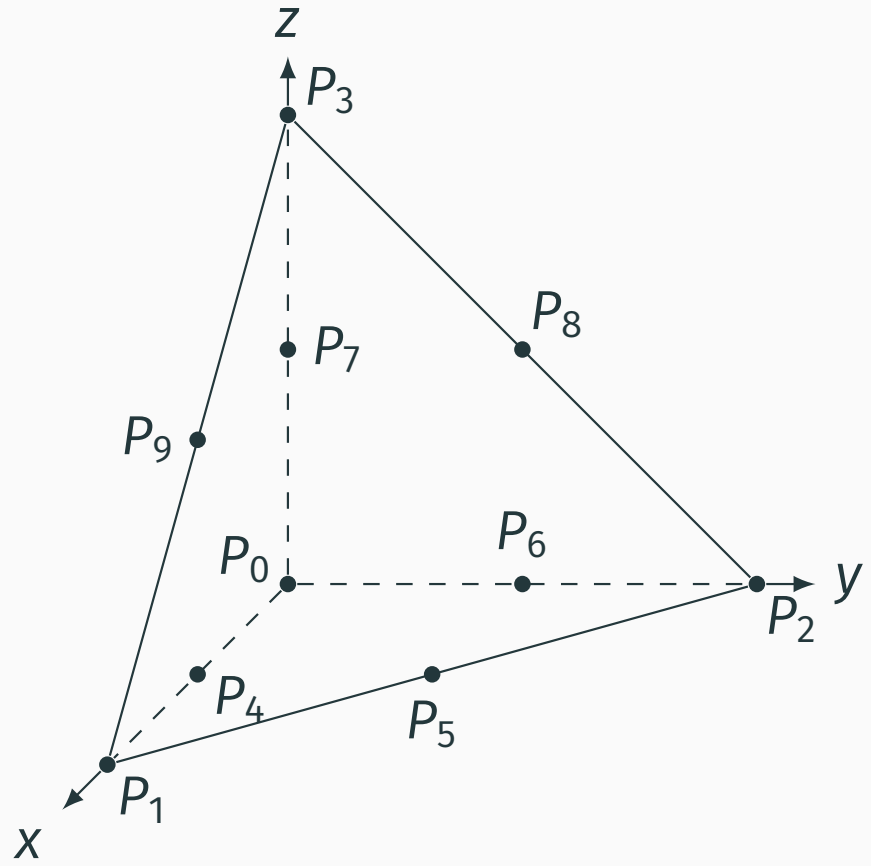
$$\partial_t f + \mathbf{v} \cdot \nabla f = 0.$$

To obtain a high-order scheme on an unstructured 3D grid, we select the Discontinuous Galerkin (DG) framework.

[2] Note that we have $d + 1$ such equations to solve (since there are $d + 1$ kinetic velocities) for m variables. These $(d + 1) \times m$ equations are decoupled.

Brief introduction to the Discontinuous Galerkin discretization

- $\mathcal{M}$: 3D unstructured mesh
- n_t: number of tetrahedra
- each tetrahedron has 10 nodes:
  4 vertices, 6 edge midpoints
- all tetrahedra have straight edges



In each tetrahedron L, we define 10 basis functions $(\varphi_i^L)_{i\in[\![0,9]\!]}$.

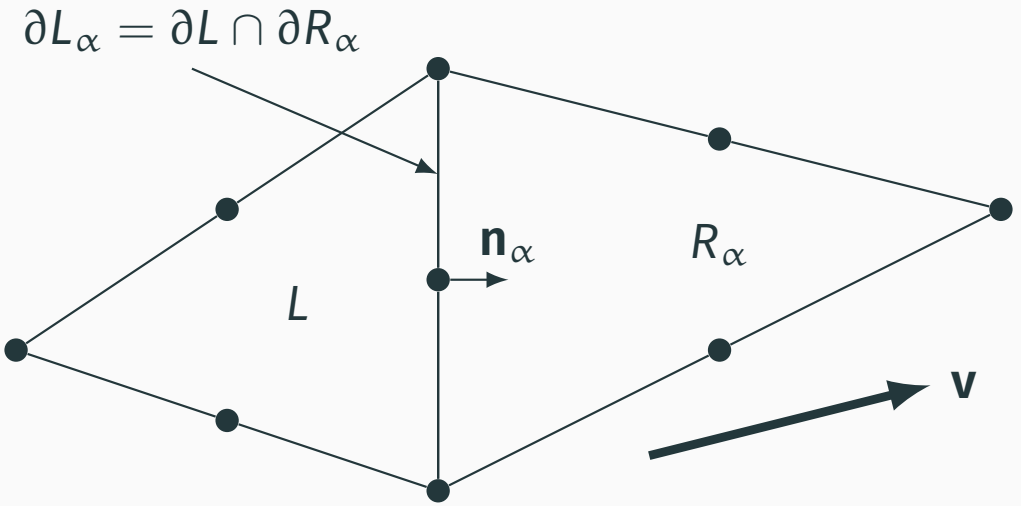
The unknown function f is approximated as follows at time t^n:

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

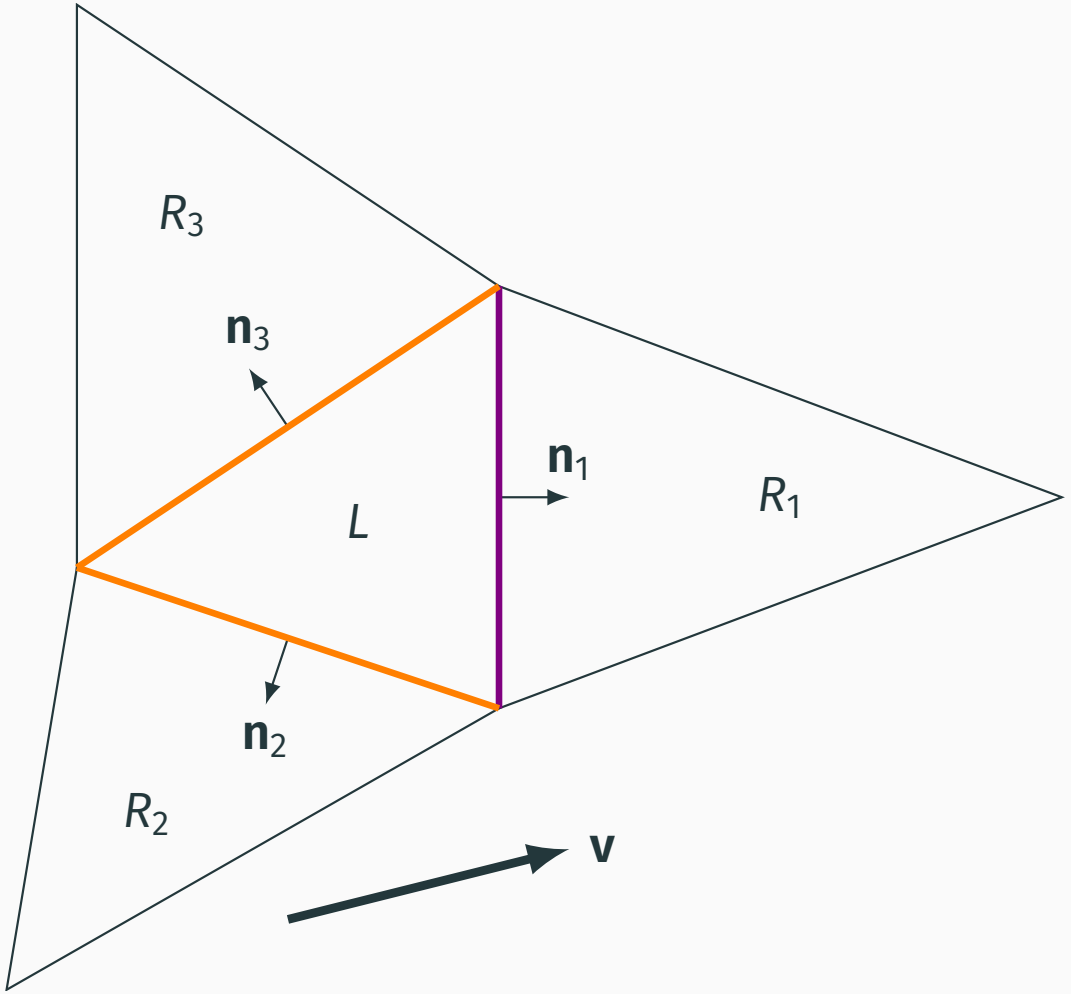
Brief introduction to the Discontinuous Galerkin discretization

With an implicit time discretization, integrating the transport equation against the basis functions yields, for all $j \in [\![0, 9]\!]$:

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



Downwind algorithm



$\mathbf{v} \cdot \mathbf{n}_1 > 0$:
downwind edge,
DG uses f_L^{n+1}

$\mathbf{v} \cdot \mathbf{n}_2 < 0, \quad \mathbf{v} \cdot \mathbf{n}_3 < 0$:
upwind edges,
DG uses $f_{R_2}^{n+1}$ and $f_{R_3}^{n+1}$

In this configuration, the DG algorithm is then recast as a linear system:

$$(M_L - \Delta t\, D_L + \Delta t\, F_L)\, f_L^{n+1} = M_L\, f_L^n + \Delta t\, F_{R_2}\, f_{R_2}^{n+1} + \Delta t\, F_{R_3}\, f_{R_3}^{n+1}.$$

Downwind algorithm

For any cell L, the DG algorithm reads:

$$(M_L - \Delta t\, D_L + \Delta t\, F_L^+)\, f_L^{n+1} = M_L\, f_L^n + \sum_{\alpha=0}^{3} \Delta t\, F_{R_\alpha}^-\, f_{R_\alpha}^{n+1}.$$

- This is a 10×10 linear system local to cell L.
- For all downwind edges, the matrix $F_{R_\alpha}^-$ vanishes.
- The matrix F_L^+ is built using the upwind edges.

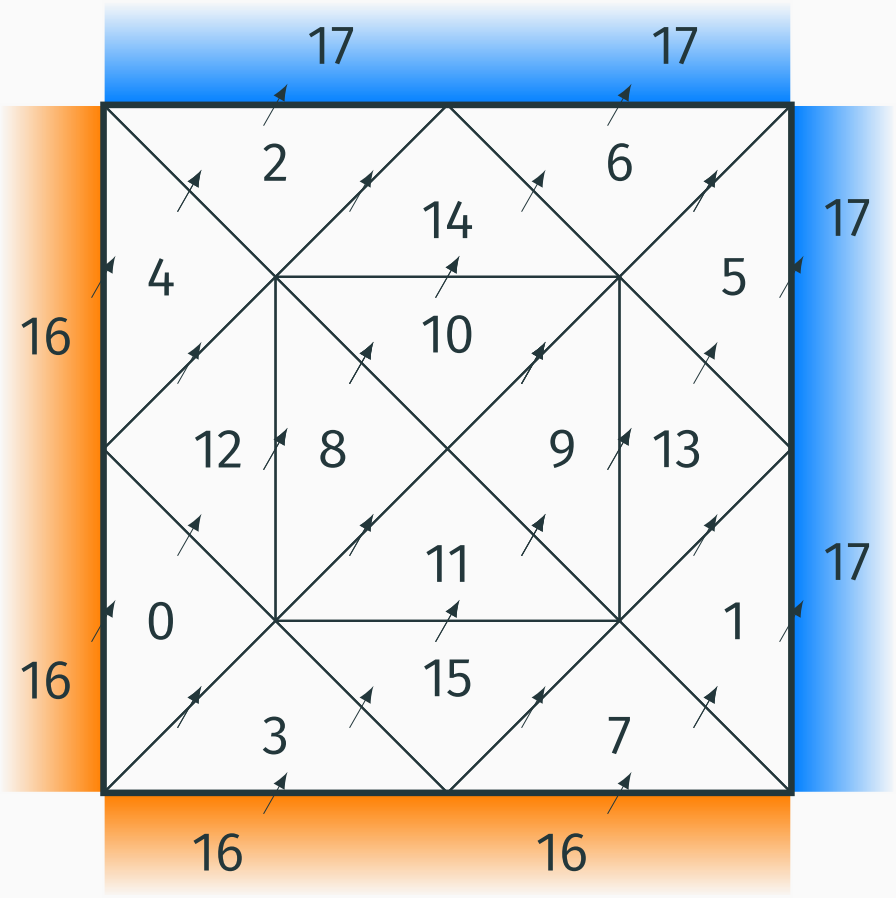
Consequence: If the upwind information is known, there is no implicit coupling between neighboring cells. Hence,

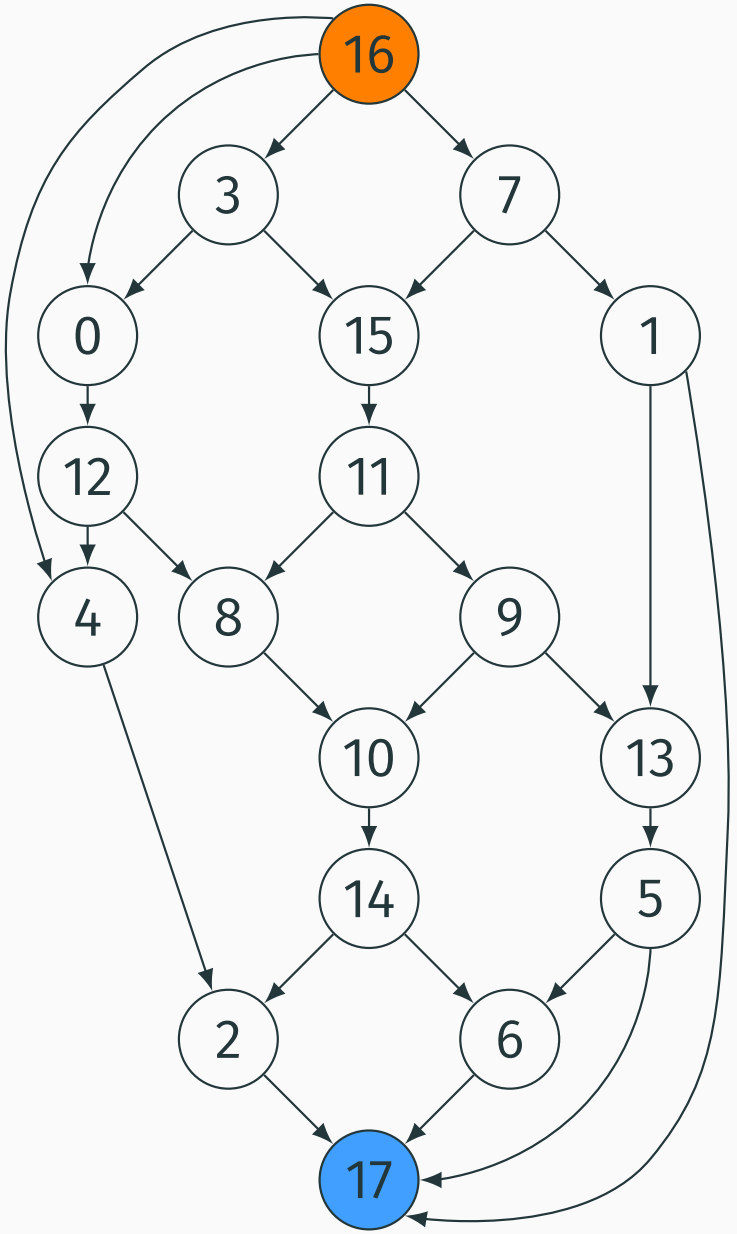
this implicit scheme has the complexity of an explicit scheme.

Remark: This is only possible because $\mathbf{v}$ is constant.

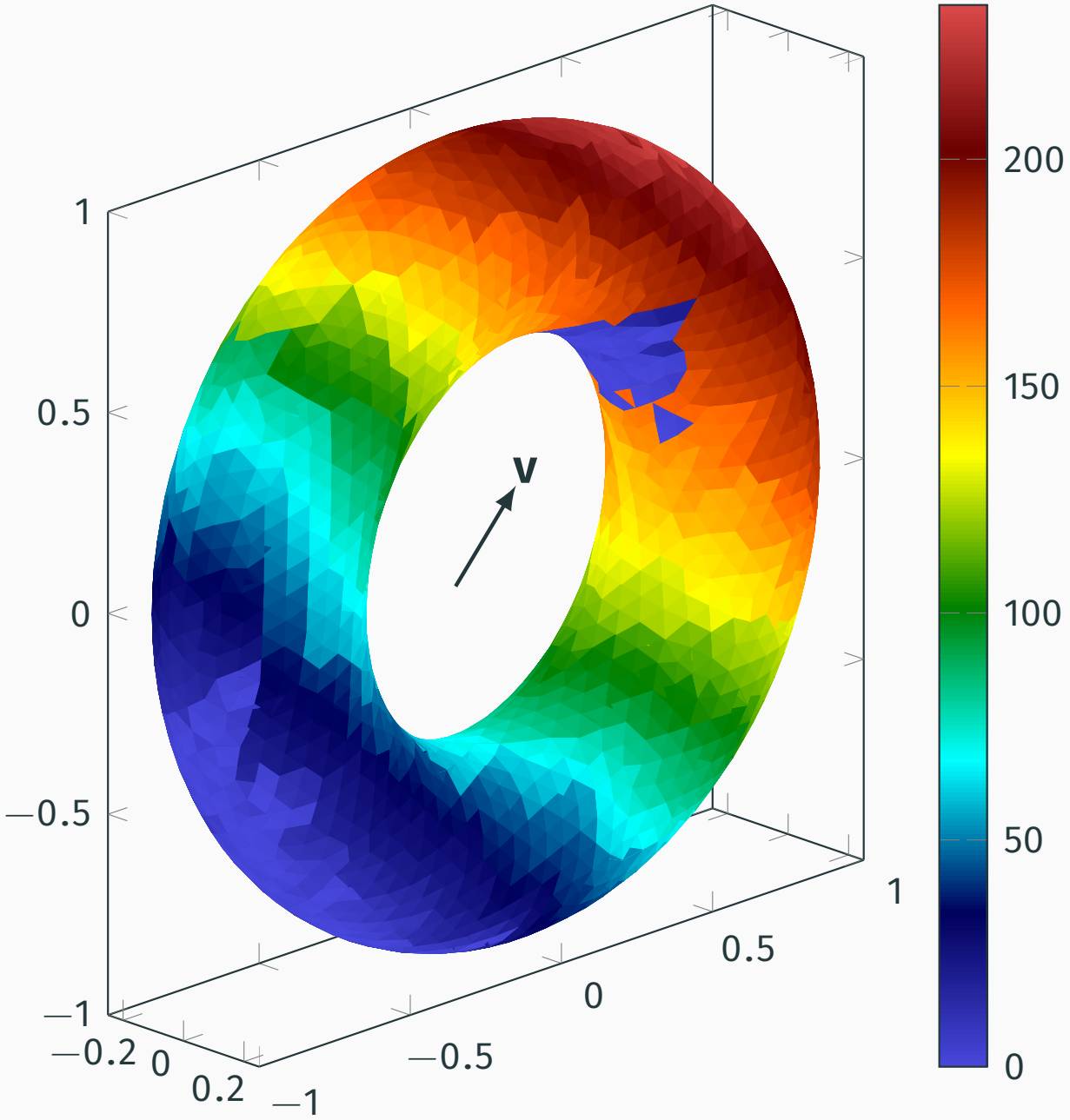
Next step: How to implement this algorithm in parallel?

Reformulation of the mesh as a graph





Visualization of the parallel zones for a full torus



Mesh of a **full torus**, showing the parallel zones, with $\mathbf{v} = (1, 1, 0)^\intercal$.

18340 elements;

235 zones treatable in parallel

Summary of the parallel algorithm

1. **Mesh preprocessing**:

   1.1 Recast the mesh $\mathcal{M}$ as a **graph**.

   1.2 For $k \in [\![0, d]\!]$, **sort** the graph associated to $\mathbf{v}_k$ using breadth-first search: this yields $(d+1)$ sorted meshes $\mathcal{S}_k$, each with n_p^k parallel regions $(\mathcal{P}_p^k)_{p \in [\![1, n_p^k]\!]}$.

2. **For each iteration of the time loop**[3]:

   2.1 Solve the **transport step** on the sorted meshes:

   `for` $k \in [\![0, d]\!]$, do `in parallel`:
   `for` $p \in [\![1, n_p^k]\!]$, do:
   `for` each cell L in the parallel region $\mathcal{P}_p^k$, do `in parallel`:
   solve the m local linear systems, all sharing the same matrix

   2.2 `for` each cell L in $\mathcal{M}$, `solve` the **relaxation-source step** `in parallel` (since it is local to the cell)

[3] In practice, we perform a palindromic Strang splitting.

A note on the CFL number

To run numerical experiments, we have to define the time step Δt.
In related work, we find:

1D nodal[4]	2D Gauss-Legendre[5]	2D Legendre-Gauss-Lobatto[6]
$\Delta t \leqslant \beta \frac{1}{\lambda} h$	$\Delta t \leqslant \gamma \frac{1}{\lambda} \frac{h}{2r+1}$	$\Delta t \leqslant \theta \frac{1}{\lambda} \frac{h}{r+1}$

with r the polynomial degree, λ the maximal wave speed, and

$$h = \min_{L \in \mathcal{M}} \frac{\text{volume}\,(L)}{\text{surface}\,(\partial L)}.$$

We will display results obtained with (very) large CFL numbers β.

[4] J. S. Hesthaven and T. Warburton, (2008)
[5] T. Toulorge and W. Desmet, *J. Comput. Phys.* (2011)
[6] G. Gassner and D. A. Kopriva, *SIAM J. Sci. Comput.* (2011)

A note on the implementation

The parallel algorithm is implemented using the **Rust language**.

- It is a recent language (2010), oriented towards **concision, speed and security**.
- Most common **bugs** (memory leaks, segmentation faults, uninitialized data, race conditions, ...) are **avoided at compile time**.
- There is **automatic shared-memory parallelization**: if the serial code works[7], then the parallel code outputs the same result.
- It is as **fast**[8] as C, C++ or `Fortran`.
- There is **no need for a makefile** and library inclusion is painless.
- There are **no native** scientific computing libraries, just wrappers. SIMD support is underway, and MPI is barely supported.

The code is called **KOUGLOFV**[9].

[7] and is implemented the right way...
[8] Gouy, Isaac. The Computer Language Benchmarks Game. Web.
[9] Kinetic schemes On Unstructured Grids for Large Optimized Finite Volume simulations

Maxwell's equations

We consider the non-dimensional Maxwell's equations:

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

where $E = (E_1, E_2, E_3)^\intercal$ and $H = (H_1, H_2, H_3)^\intercal$ are the electric and magnetic fields, and where σ is the electrical conductivity.

It is written under the present formalism by defining

$$\mathbf{u} = (E_1, E_2, E_3, H_1, H_2, H_3)^\intercal \quad \text{and} \quad \mathbf{q}(\mathbf{u}, \mathbf{n}) = (-(\mathbf{n} \times \mathbf{H})^\intercal, (\mathbf{n} \times \mathbf{E})^\intercal).$$

The initial and boundary conditions read, with $\Omega = [0, 1]^3 \subset \mathbb{R}^3$:

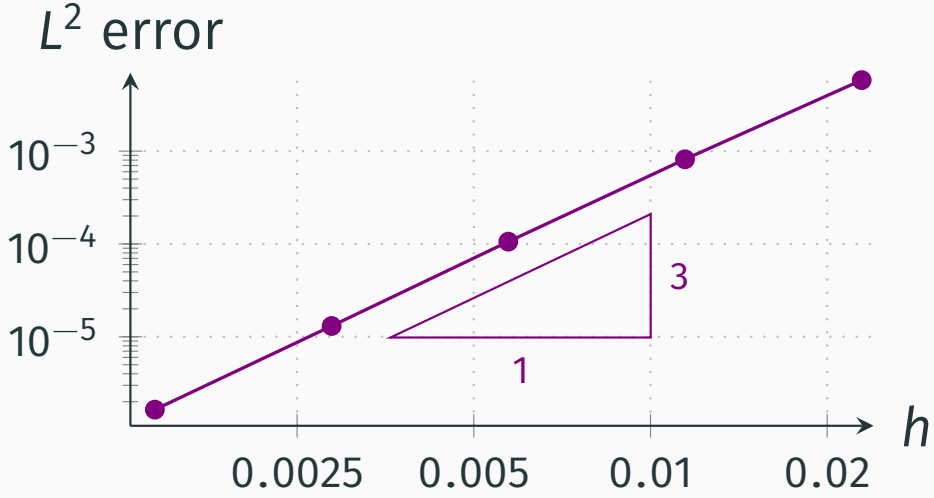
$$\begin{cases} \mathbf{u}(\mathbf{x}, 0) = \mathbf{u}_{\text{exact}}(\mathbf{x}, 0), & \forall \mathbf{x} \in \Omega, \\ \mathbf{u}(\mathbf{x}, t) = \mathbf{u}_{\text{exact}}(\mathbf{x}, t), & \forall \mathbf{x} \in \partial\Omega, \ \forall t \leqslant t_{\text{end}}, \end{cases}$$

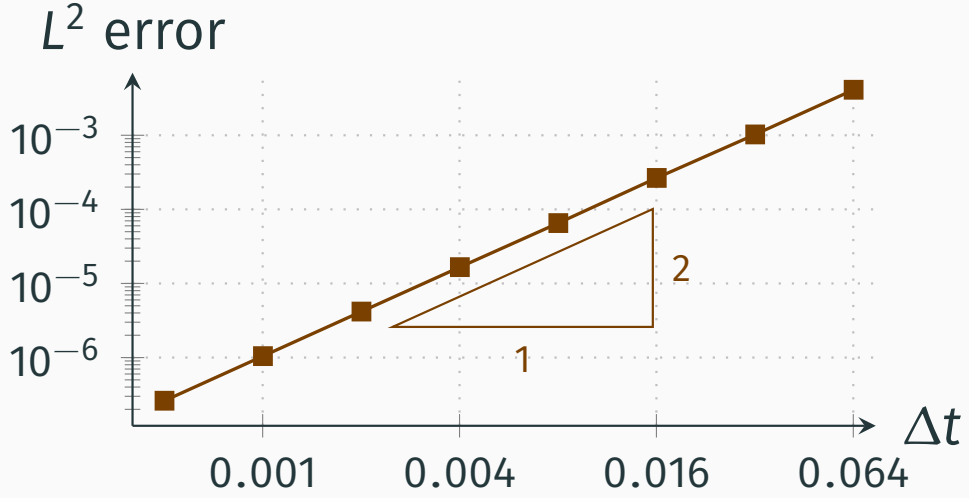
where $\mathbf{u}_{\text{exact}}$ is an exact solution of Maxwell's equations.

Order of accuracy

We evaluate the order of accuracy of the scheme:

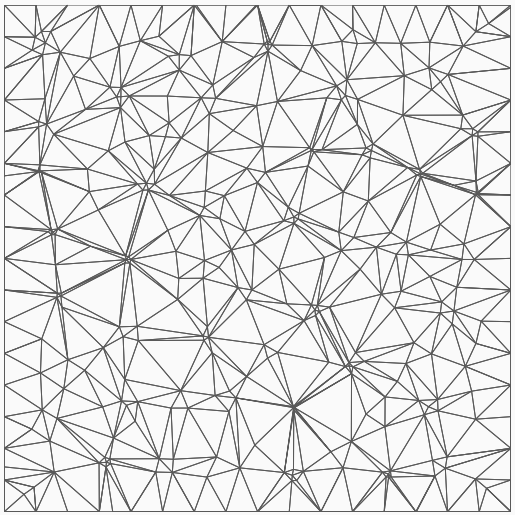
- in space with the exact solution being a low frequency wave with small Δt,
- in time with the exact solution being a quadratic function, for which the space integration is exact.



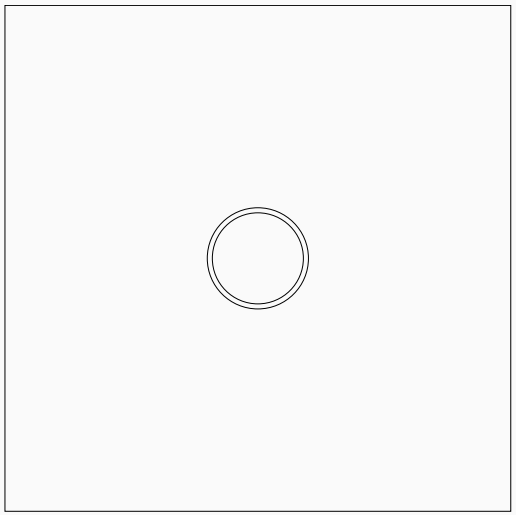


CFL-less aspects

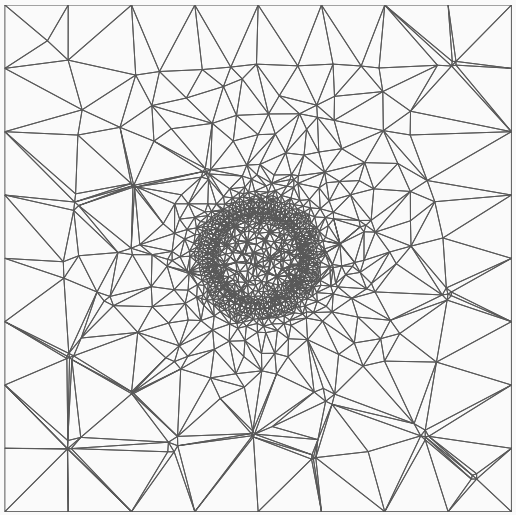
To test the stability of the method, we introduce two meshes and two exact solutions, plane waves with frequencies $\nu_s = 2$ and $\nu_f = 5$.



$\mathcal{M}_1$



$\mathcal{M}_2$ (geometry)

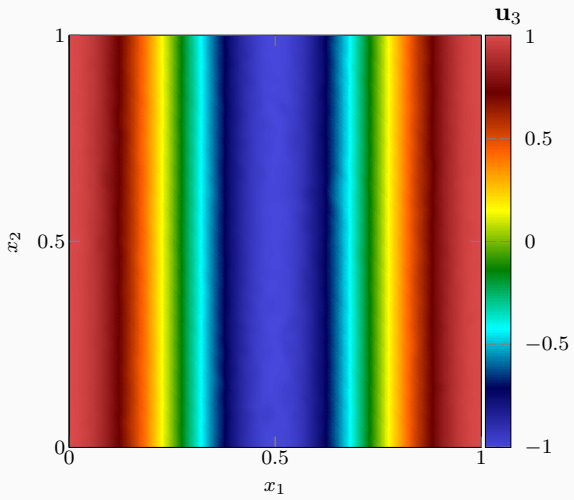


$\mathcal{M}_2$ (mesh)

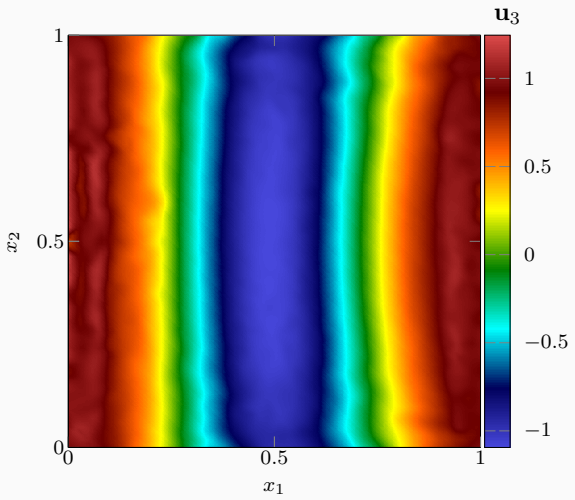
mesh	#tetrahedra	h	$\frac{\text{volume of largest cell}}{\text{volume of smallest cell}}$
$\mathcal{M}_1$	9199	$\simeq 2.28 \times 10^{-3}$	$\simeq 4.5$
$\mathcal{M}_2$	34280	$\simeq 2.47 \times 10^{-4}$	$\simeq 60$

CFL-less aspects: results on mesh $\mathcal{M}_1$

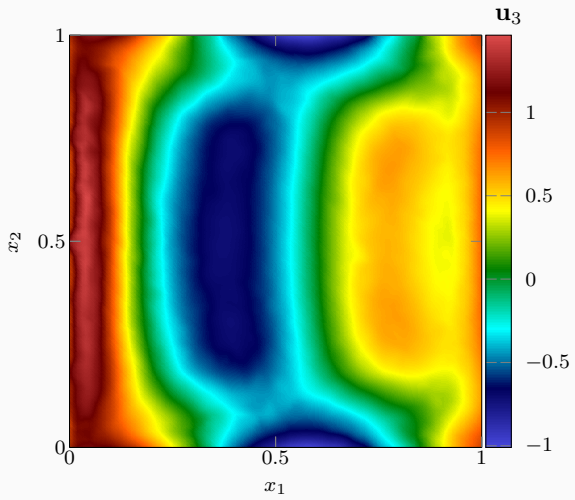
$\nu_s = 2$



CFL $\beta = 1.85$

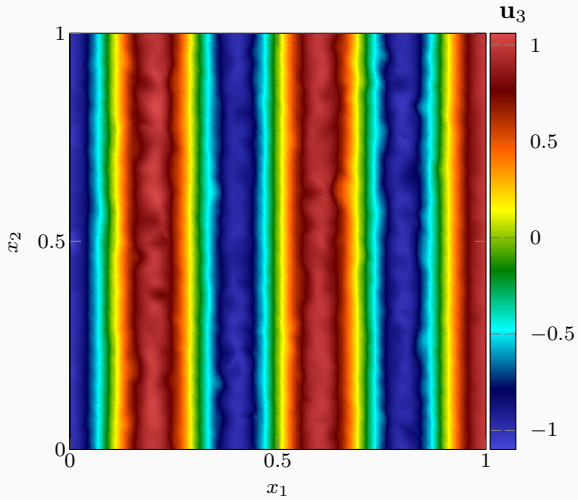


CFL $\beta = 37$

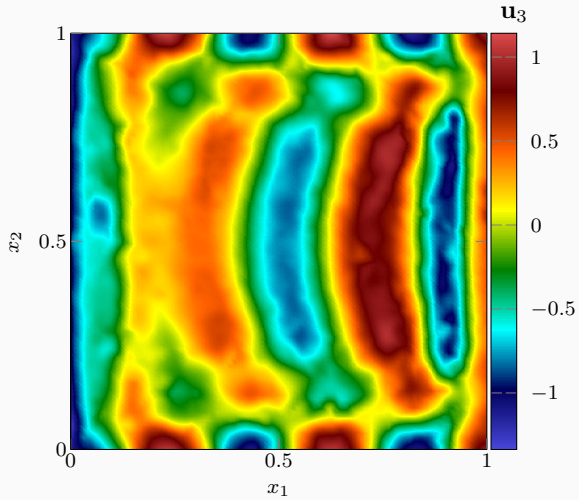


CFL $\beta = 92$

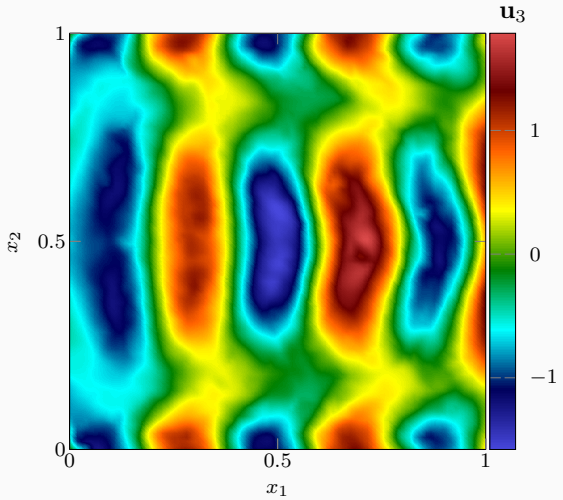
$\nu_s = 5$



CFL $\beta = 1.85$



CFL $\beta = 37$



CFL $\beta = 92$

CFL-less aspects: results on mesh $\mathcal{M}_1$

Code	CFL β	Δt	L^2 error	
			$\nu_s = 2$	$\nu_f = 5$
gdon	0.37	0.00084	0.00032	0.00609
KOUGLOFV	0.37	0.00084	0.00046	0.00627
gdon	0.93	0.00211	0.00032	0.00610
KOUGLOFV	0.93	0.00211	0.00047	0.00657
gdon	1.85	0.00422	0.00032	0.00609
KOUGLOFV	1.85	0.00422	0.00062	0.00891

CFL-less aspects: results on mesh $\mathcal{M}_1$

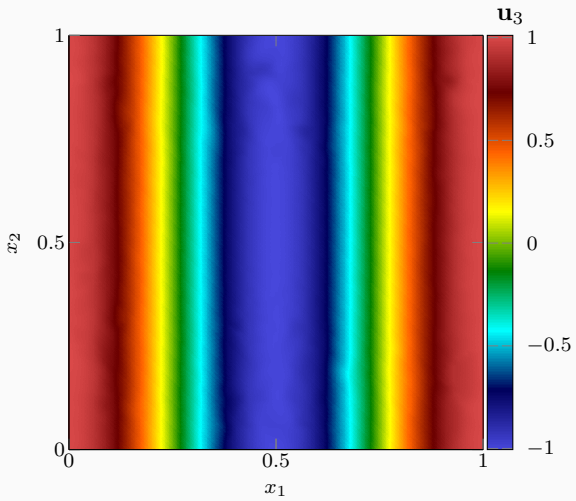
Code	CFL β	Δt	L^2 error	
			$\nu_s = 2$	$\nu_f = 5$
gdon	0.37	0.00084	0.00032	0.00609
KOUGLOFV	0.37	0.00084	0.00046	0.00627
gdon	0.93	0.00211	0.00032	0.00610
KOUGLOFV	0.93	0.00211	0.00047	0.00657
gdon	1.85	0.00422	0.00032	0.00609
KOUGLOFV	1.85	0.00422	0.00062	0.00891
KOUGLOFV	3.70	0.00845	0.00162	0.02397
KOUGLOFV	9.25	0.02112	0.00960	0.14851
KOUGLOFV	18.50	0.04223	0.03990	0.42444
KOUGLOFV	37.00	0.08447	0.14919	0.34411
KOUGLOFV	92.50	0.21117	0.25771	0.67218
KOUGLOFV	185.00	0.42234	0.45671	0.49513

CFL-less aspects: results on mesh $\mathcal{M}_1$

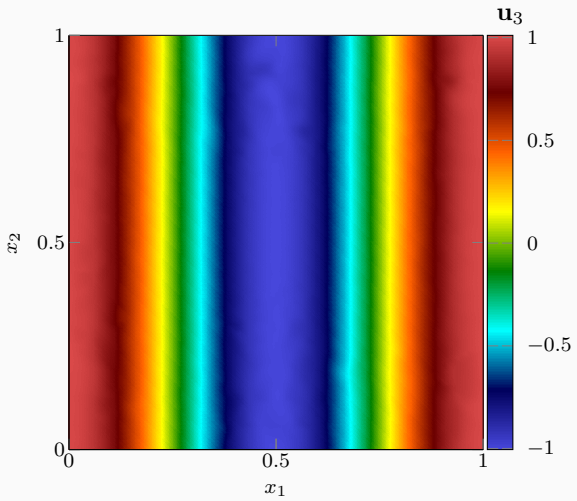
Code	CFL β	Δt	L^2 error		CPU (s)	
			$\nu_s = 2$	$\nu_f = 5$	1 thread	24 threads
gdon	0.37	0.00084	0.00032	0.00609	360.01	72.54
KOUGLOFV	0.37	0.00084	0.00046	0.00627	96.27	15.53
gdon	0.93	0.00211	0.00032	0.00610	146.81	29.19
KOUGLOFV	0.93	0.00211	0.00047	0.00657	38.63	6.52
gdon	1.85	0.00422	0.00032	0.00609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

CFL-less aspects: results on mesh $\mathcal{M}_2$

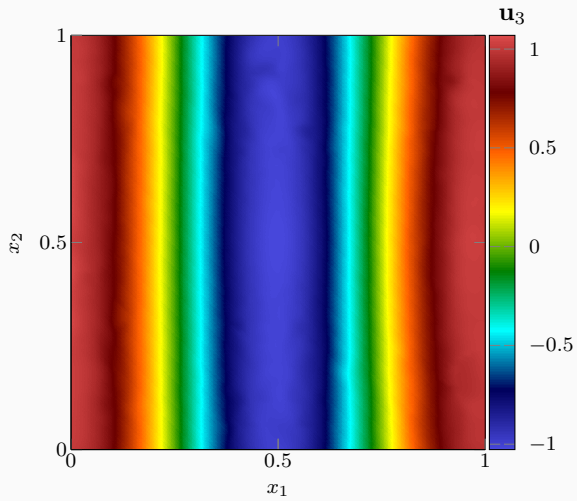
$\nu_s = 2$



CFL $\beta = 1.85$

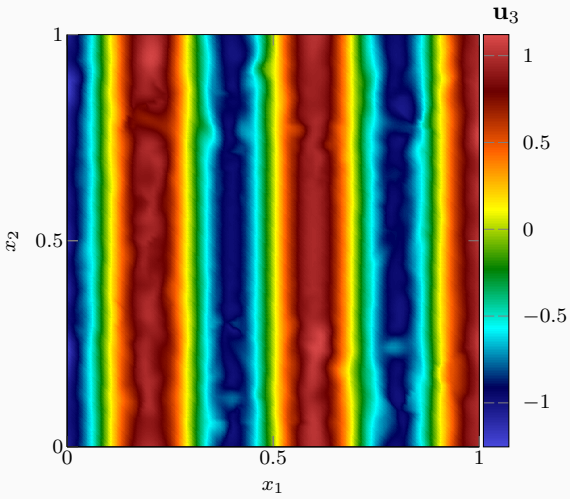


CFL $\beta = 18.5$

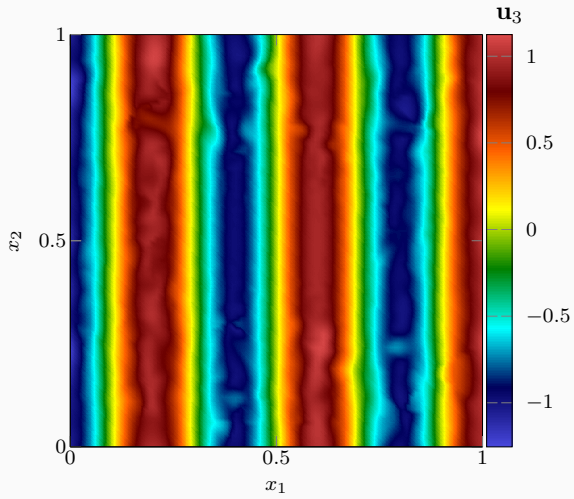


CFL $\beta = 185$

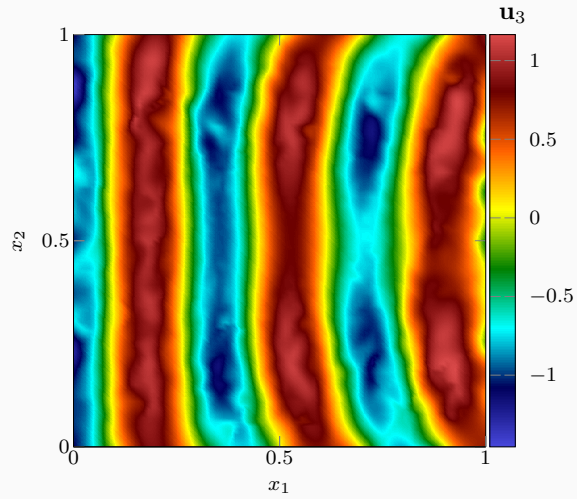
$\nu_s = 5$



CFL $\beta = 1.85$



CFL $\beta = 18.5$



CFL $\beta = 185$

CFL-less aspects: results on mesh $\mathcal{M}_2$

Code	CFL β	Δt	L^2 error $\nu_s = 2$	L^2 error $\nu_f = 5$
gdon	0.37	0.00009	0.00070	0.01238
KOUGLOFV	0.37	0.00009	0.00103	0.01467
gdon	0.93	0.00023	0.00070	0.01238
KOUGLOFV	0.93	0.00023	0.00103	0.01467
gdon	1.85	0.00046	0.00070	0.01238
KOUGLOFV	1.85	0.00046	0.00103	0.01467

CFL-less aspects: results on mesh $\mathcal{M}_2$

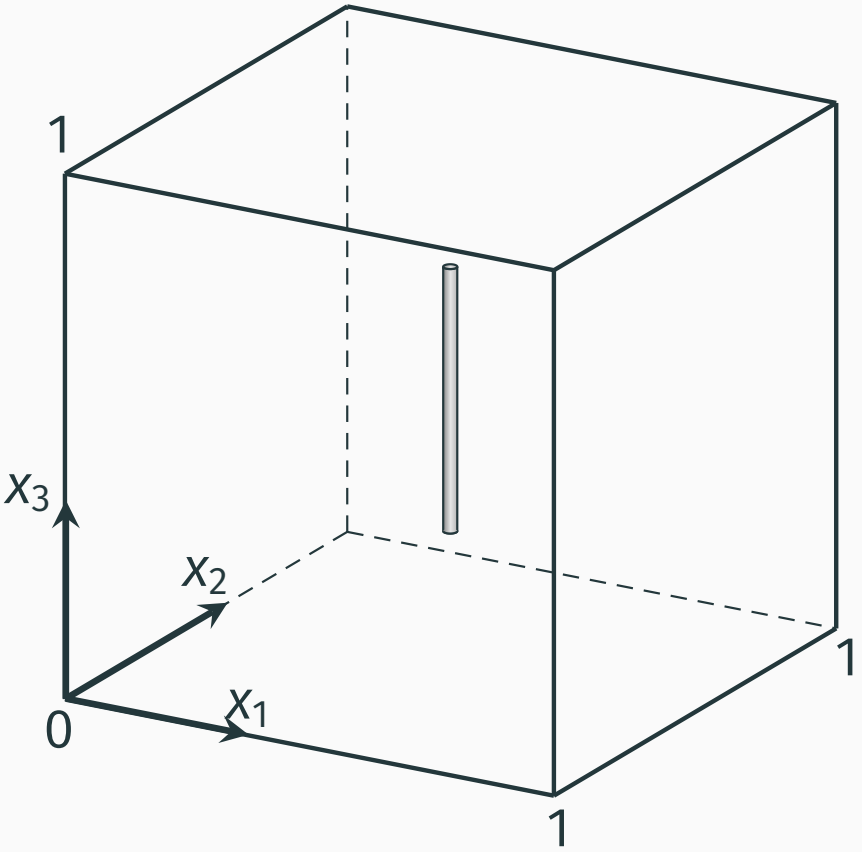
Code	CFL β	Δt	L^2 error	
			$\nu_s = 2$	$\nu_f = 5$
gdon	0.37	0.00009	0.00070	0.01238
KOUGLOFV	0.37	0.00009	0.00103	0.01467
gdon	0.93	0.00023	0.00070	0.01238
KOUGLOFV	0.93	0.00023	0.00103	0.01467
gdon	1.85	0.00046	0.00070	0.01238
KOUGLOFV	1.85	0.00046	0.00103	0.01467
KOUGLOFV	3.70	0.00091	0.00103	0.01468
KOUGLOFV	9.25	0.00228	0.00104	0.01479
KOUGLOFV	18.50	0.00456	0.00115	0.01619
KOUGLOFV	37.00	0.00912	0.00210	0.02992
KOUGLOFV	92.50	0.02281	0.01107	0.16589
KOUGLOFV	185.00	0.04562	0.04509	0.40344

CFL-less aspects: results on mesh $\mathcal{M}_2$

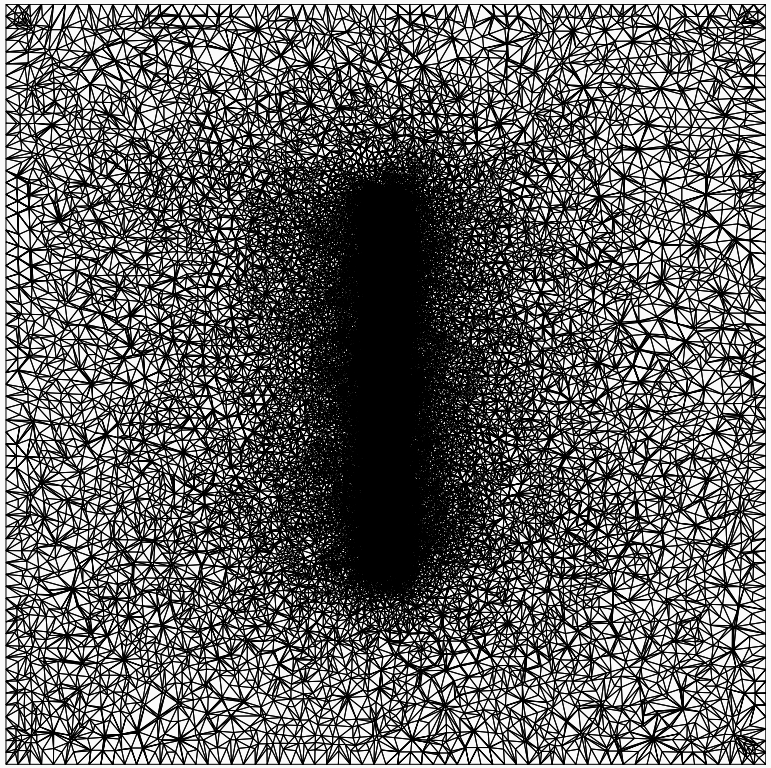
Code	CFL β	Δt	L^2 error		CPU (s)	
			$\nu_s = 2$	$\nu_f = 5$	1 thread	24 threads
gdon	0.37	0.00009	0.00070	0.01238	4,607.95	785.28
KOUGLOFV	0.37	0.00009	0.00103	0.01467	1,524.45	234.48
gdon	0.93	0.00023	0.00070	0.01238	2,189.76	384.79
KOUGLOFV	0.93	0.00023	0.00103	0.01467	613.44	90.84
gdon	1.85	0.00046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

Results with a source term – conductive antenna

We consider a conductive antenna in vacuum
($\sigma \neq 0$ in the antenna, $\sigma = 0$ elsewhere).
The mesh is made of about 516k elements; its largest/smallest ratio is 28.



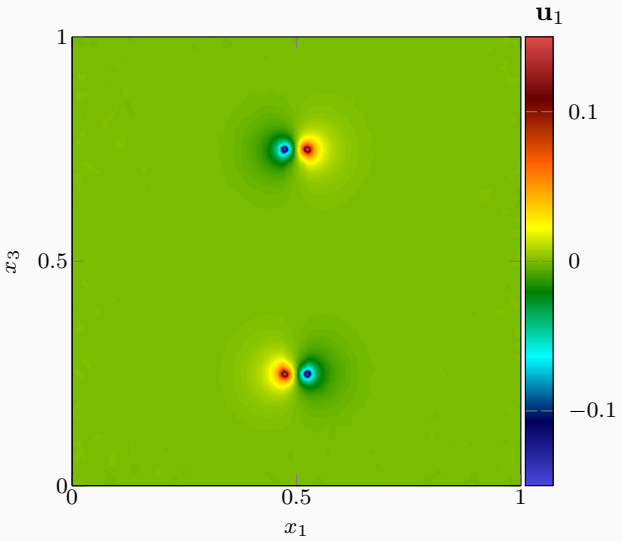
antenna mesh: geometry



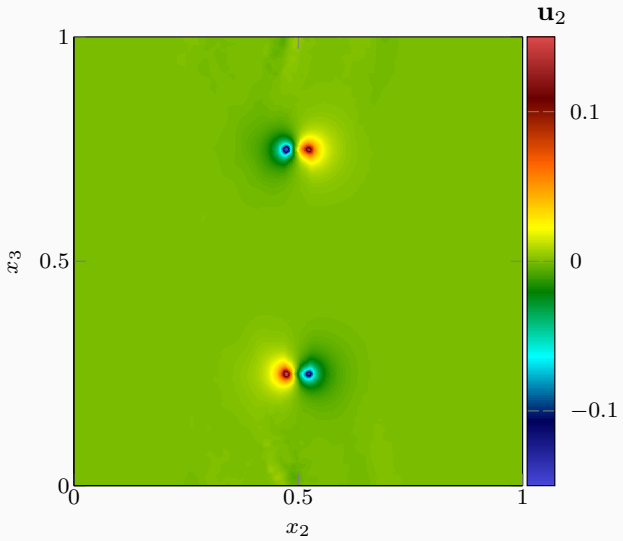
antenna mesh sliced at $x_1 = 0.5$

Results with a source term – conductive antenna, $\sigma = 3$

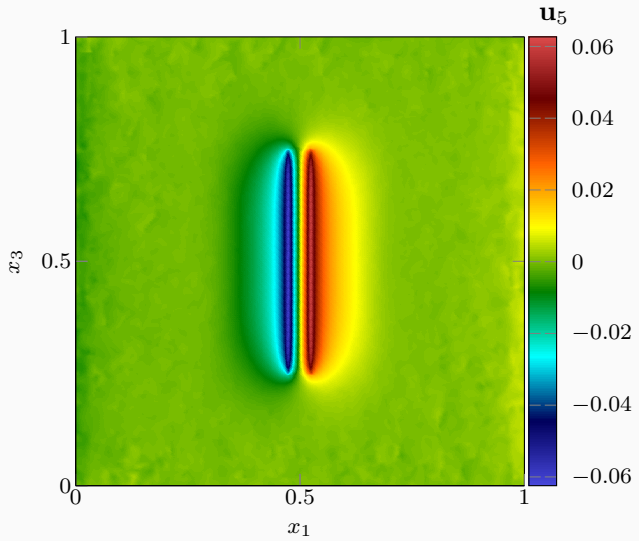
We first display the results with $\beta = 20$, about 11 times larger than the classical CFL condition.



$E_1\big|_{x_2=0.5}$



$E_2\big|_{x_1=0.5}$



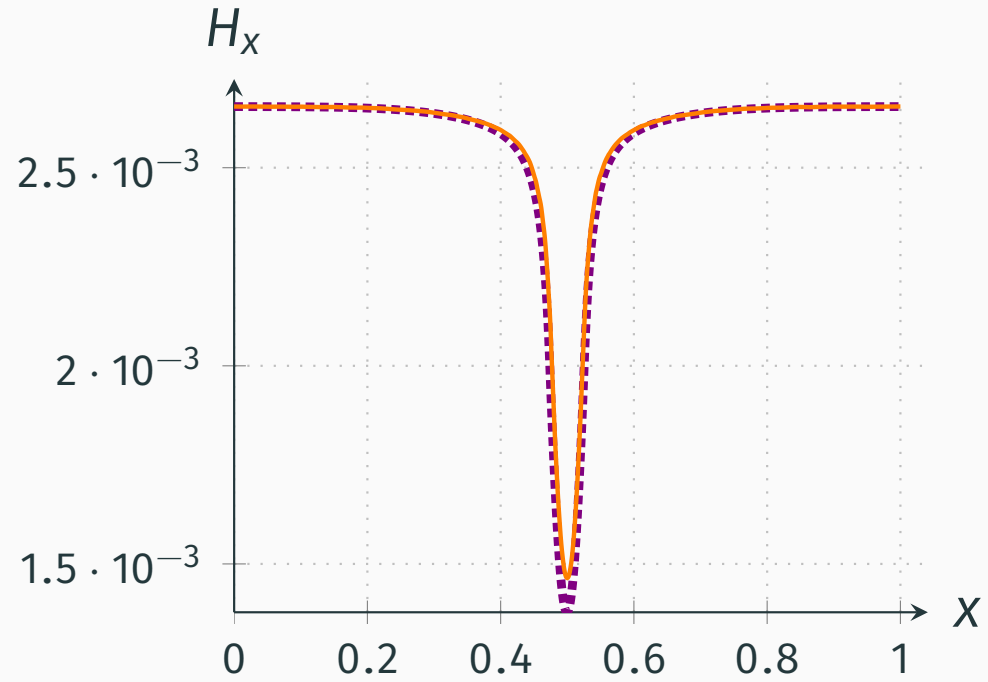
$H_2\big|_{x_2=0.5}$

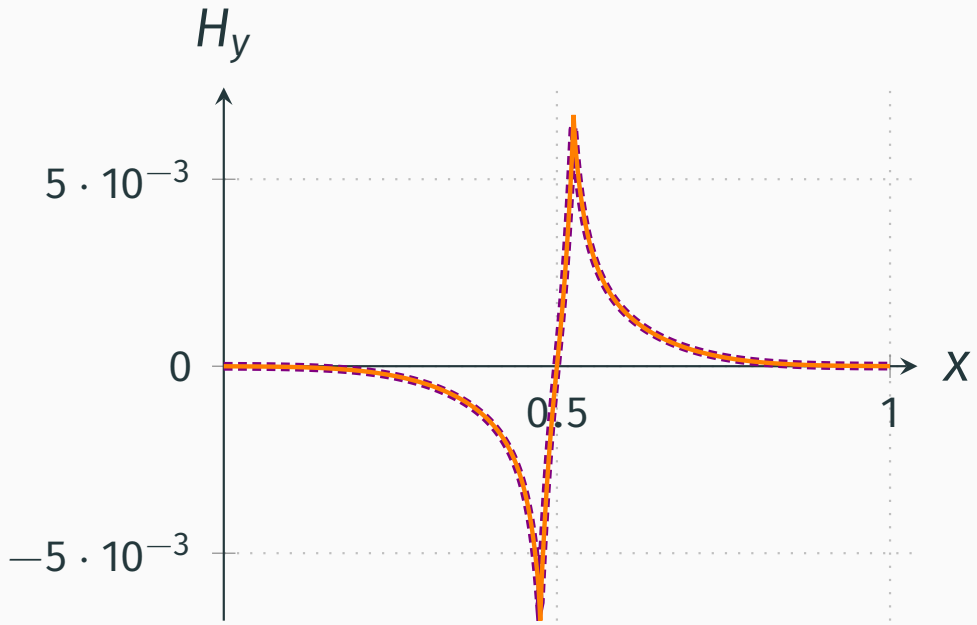
Results with a source term – conductive antenna, $\sigma = 3$

We now compare the results with a FDTD reference solver.

CPU time for FDTD: 22 hours on 8 cores (Intel Xeon).

CPU time for KOUGLOFV: 17 minutes on 6 cores (Intel i7), with $\beta = 7$.





FDTD KOUGLOFV

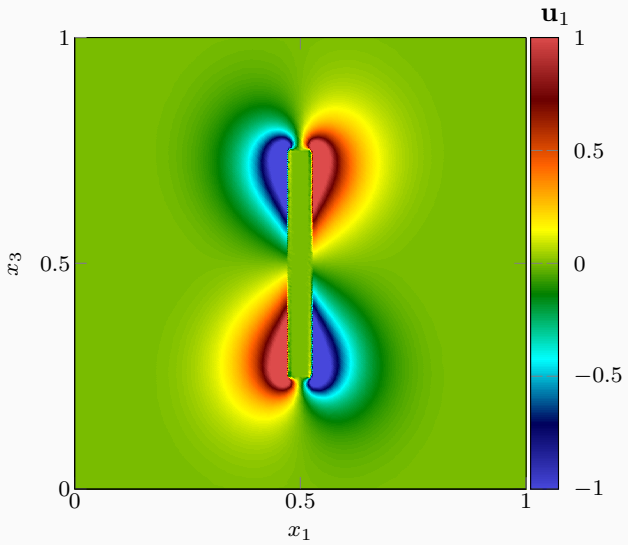
Results with a source term – conductive antenna, $\sigma = +\infty$

We take $\sigma \to +\infty$ to simulate a PEC (perfect electric conductor).
The source term update reads:

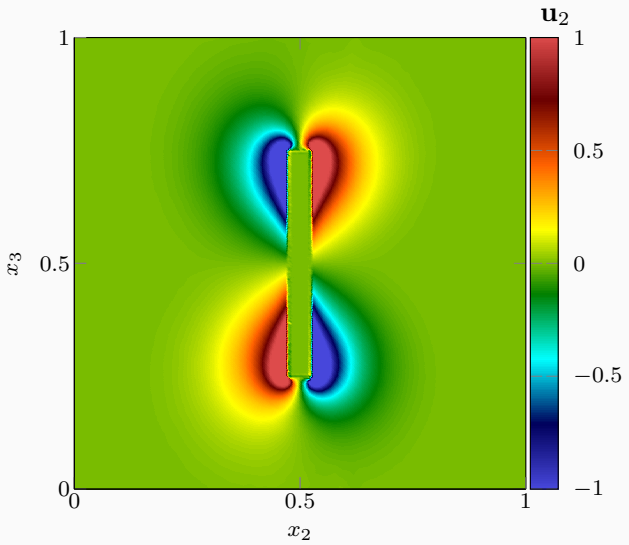
$$\begin{cases} E^{n+1} = \mu E^n, \\ H^{n+1} = H^n, \end{cases} \quad \text{where} \quad \mu = \frac{1 - \sigma\frac{\Delta t}{2}}{1 + \sigma\frac{\Delta t}{2}} \xrightarrow[\sigma \to +\infty]{} -1.$$

When $\sigma \to +\infty$, we get $E^{n+1} = -E^n$: this is correct PEC behavior.

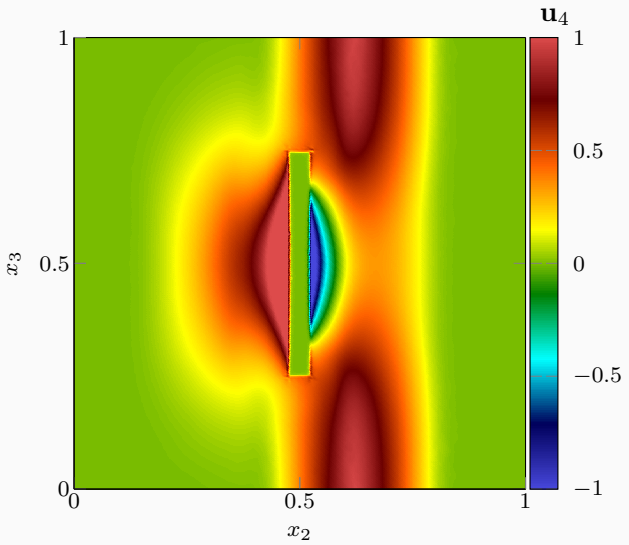
We first display the results with $\beta = 20$.



$E_1\big|_{x_2=0.5}$



$E_2\big|_{x_1=0.5}$



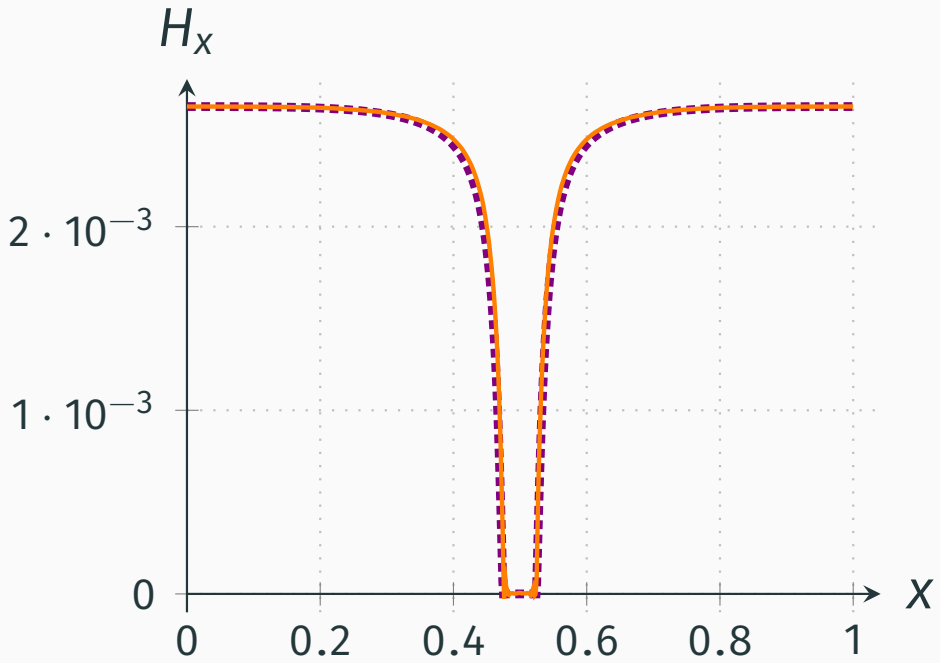
$H_1\big|_{x_1=0.5}$

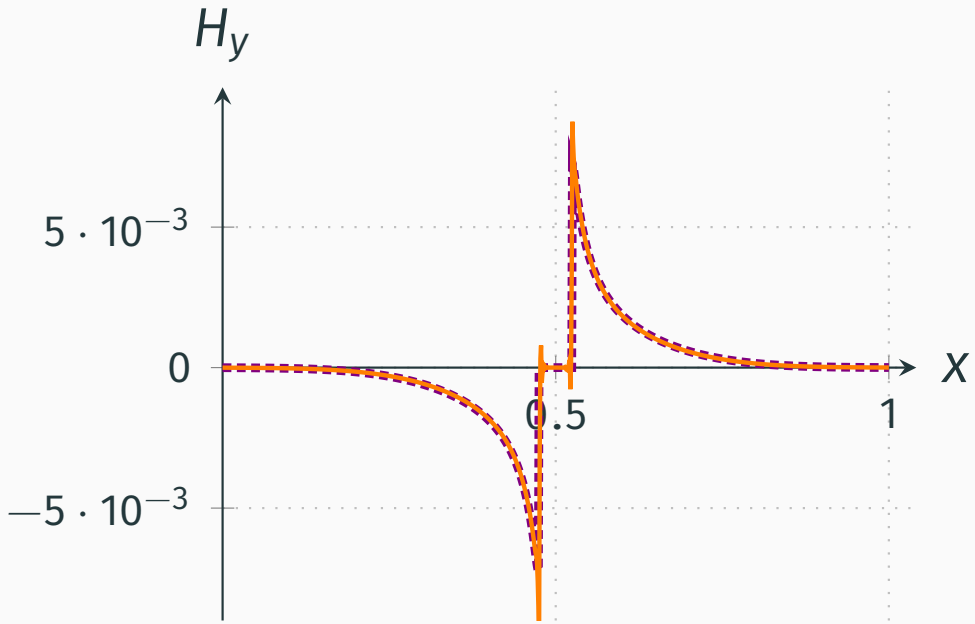
Results with a source term – conductive antenna, $\sigma = +\infty$

We now compare the results with a FDTD reference solver.

CPU time for FDTD: 22 hours on 8 cores (Intel Xeon).

CPU time for KOUGLOFV: 17 minutes on 6 cores (Intel i7), with $\beta = 7$.





FDTD — KOUGLOFV

Scalability test

Lastly, we perform a scalability test of the KOUGLOFV code, on 24 cores (Intel Xeon).

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

Efficiency seems to be capped at around 60% for 24 cores...

Next step: Extend the algorithm to be able to use MPI.

Subdomain decomposition

The relaxation-source step is already local, and therefore it is embarrassingly parallel.

We focus on solving the **transport equation** on the domain $\Omega \times (0, \Delta t)$:

$$\begin{cases} \partial_t f + \mathbf{v} \cdot \boldsymbol{\nabla} f = 0 & \text{for } \mathbf{x} \in \Omega \text{ and } t \in (0, \Delta t), \\ f(\mathbf{x}, 0) = f^0(\mathbf{x}) & \text{for } \mathbf{x} \in \Omega. \end{cases}$$

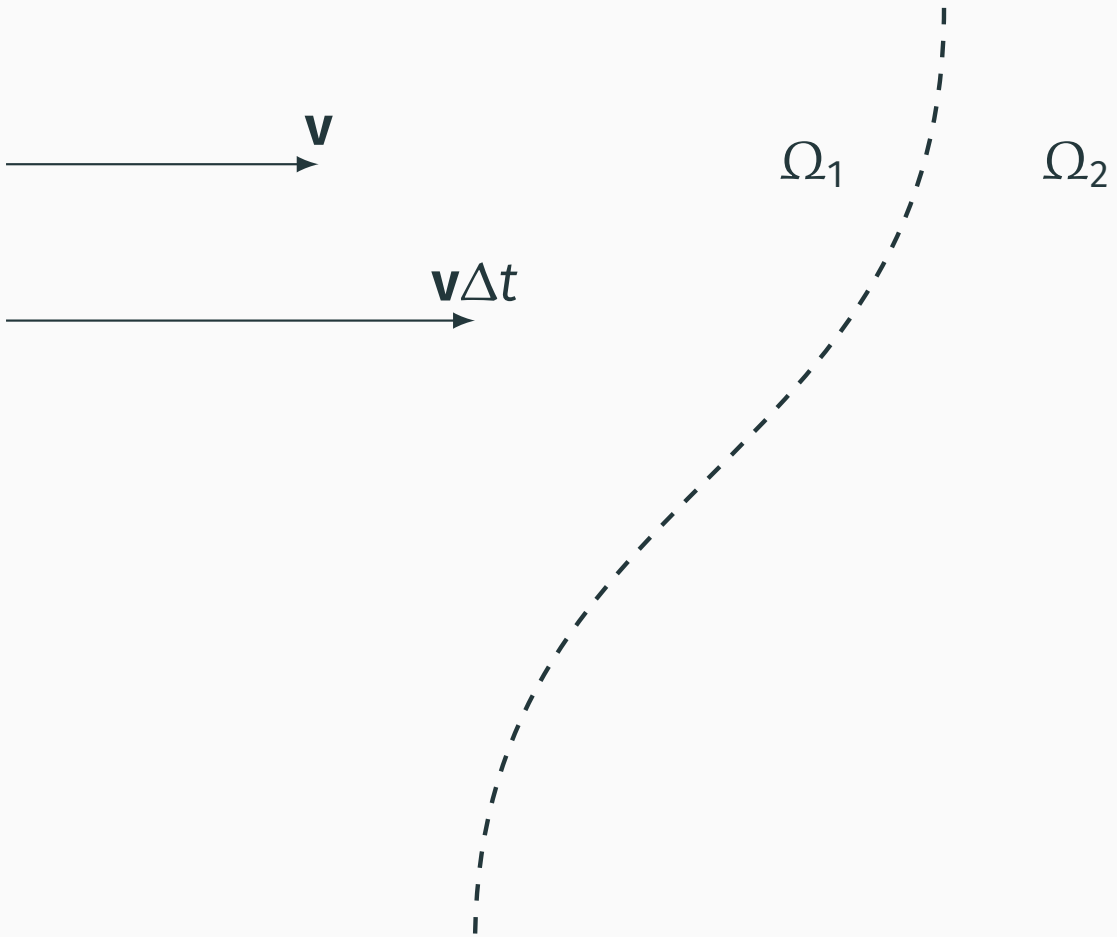
The domain Ω is decomposed into subdomains Ω_i.

We denote by:

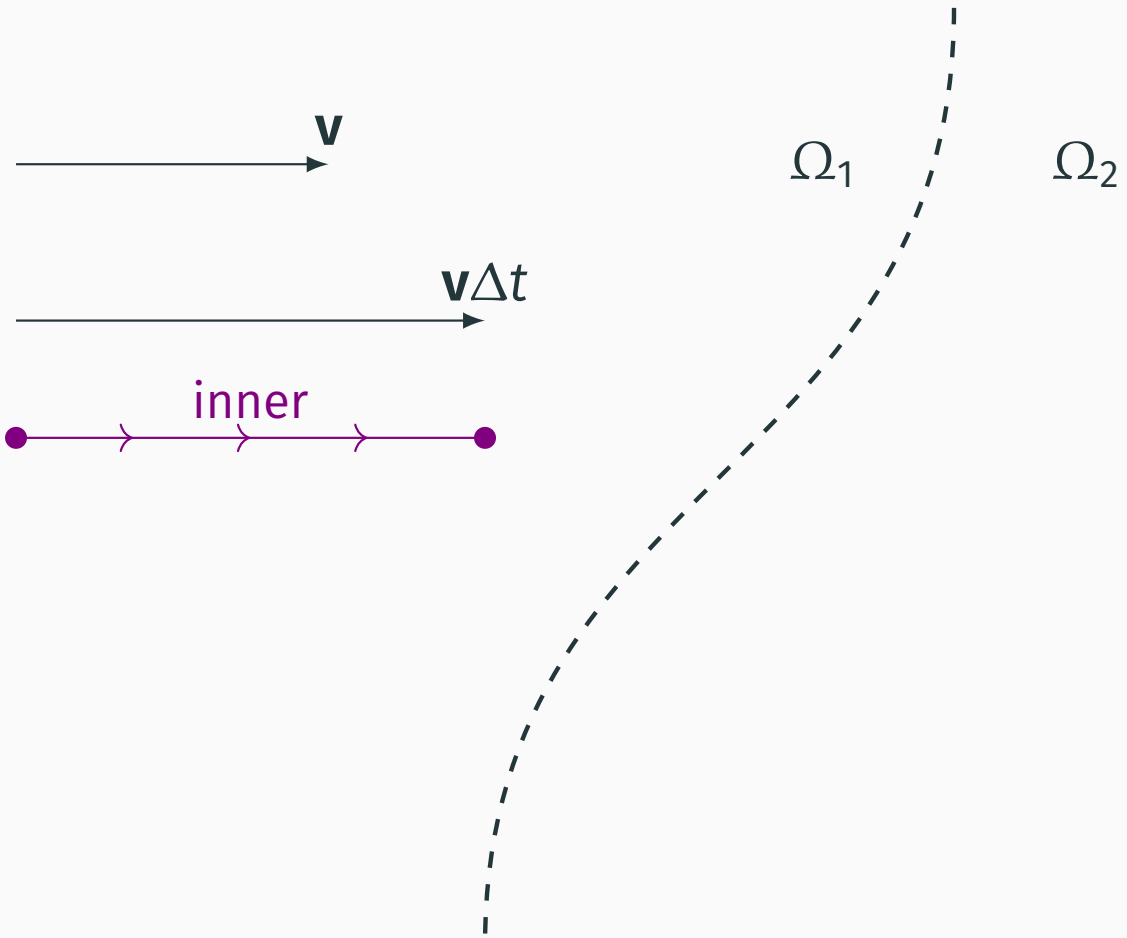
- f_i the restriction of f to Ω_i;
- $\mathbf{n}_i(\boldsymbol{\eta})$ the outward normal vector to $\partial\Omega_i$ for $\boldsymbol{\eta} \in \partial\Omega_i$;
- $\mathcal{N}(\Omega_i)$ the subdomains neighboring Ω_i;
- $\partial\Omega_i^-$ the upwind part of the boundary of Ω_i:

$$\partial\Omega_i^- = \left\{ \boldsymbol{\eta} \in \partial\Omega_i \mid \mathbf{n}_i(\boldsymbol{\eta}) \cdot \mathbf{v} < 0 \right\}.$$

Iterative algorithm – illustration on aligned subdomains



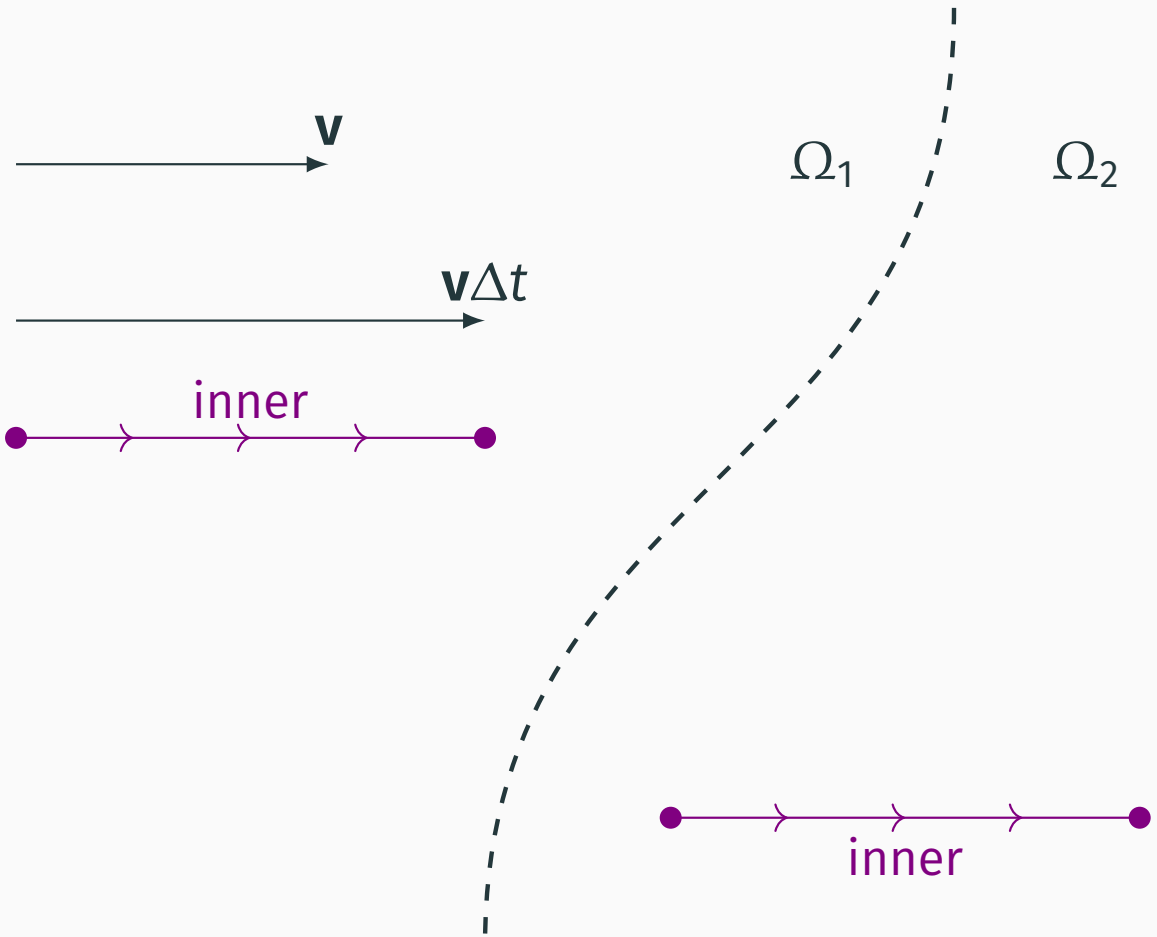
Iterative algorithm – illustration on aligned subdomains



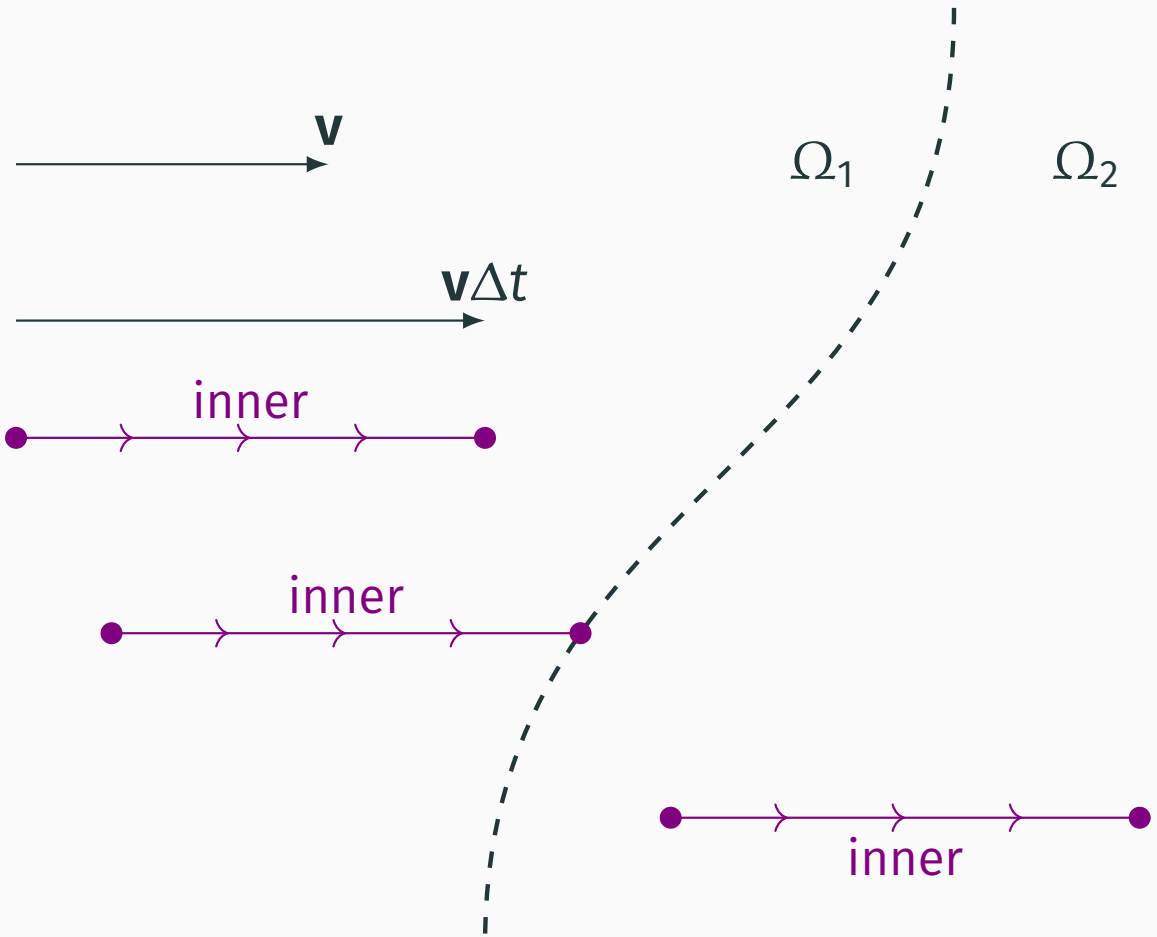
First iteration:

- inner values are correct,
- but boundary values come from the previous time iteration.

Iterative algorithm – illustration on aligned subdomains



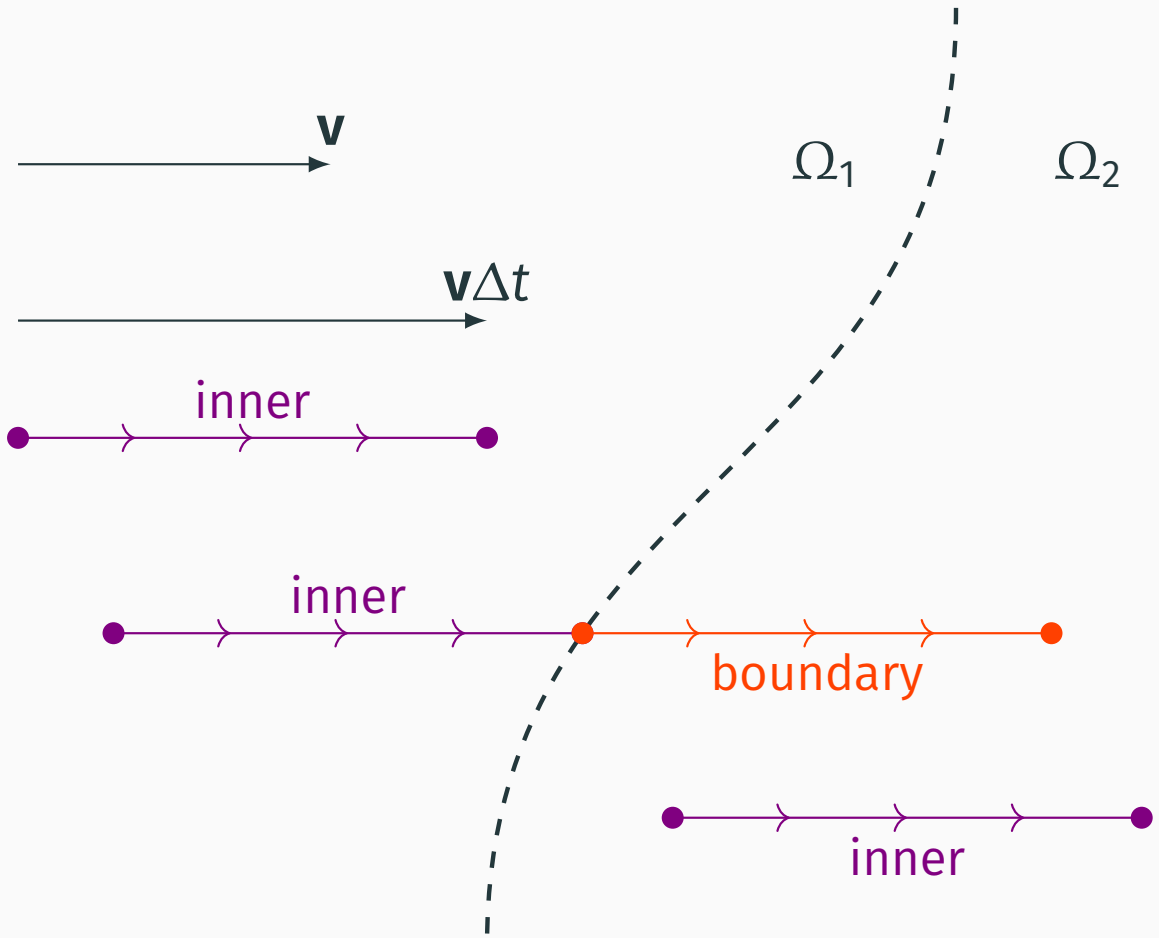
Iterative algorithm – illustration on aligned subdomains



First iteration:

- inner values are correct,
- but boundary values come from the previous time iteration.

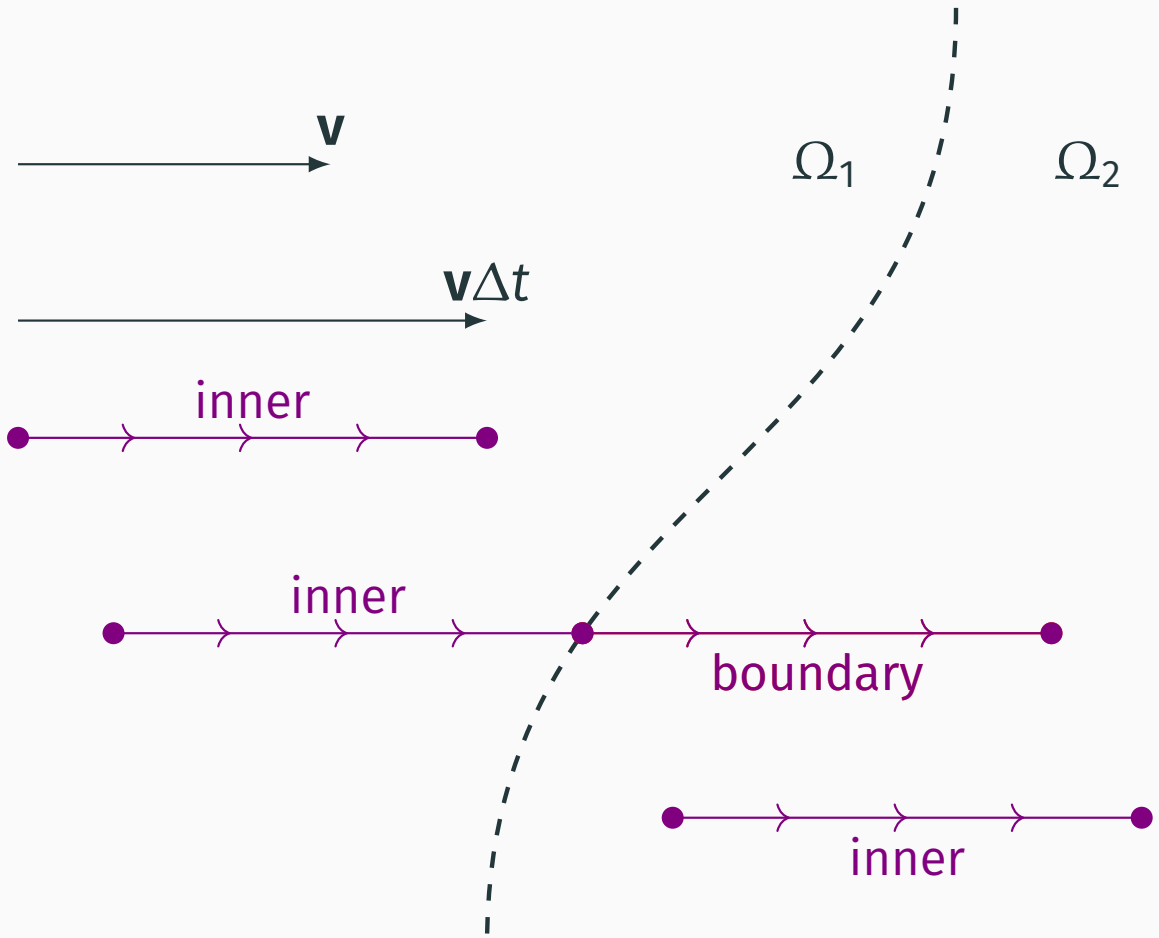
Iterative algorithm – illustration on aligned subdomains



First iteration:

- inner values are correct,
- but boundary values come from the previous time iteration!

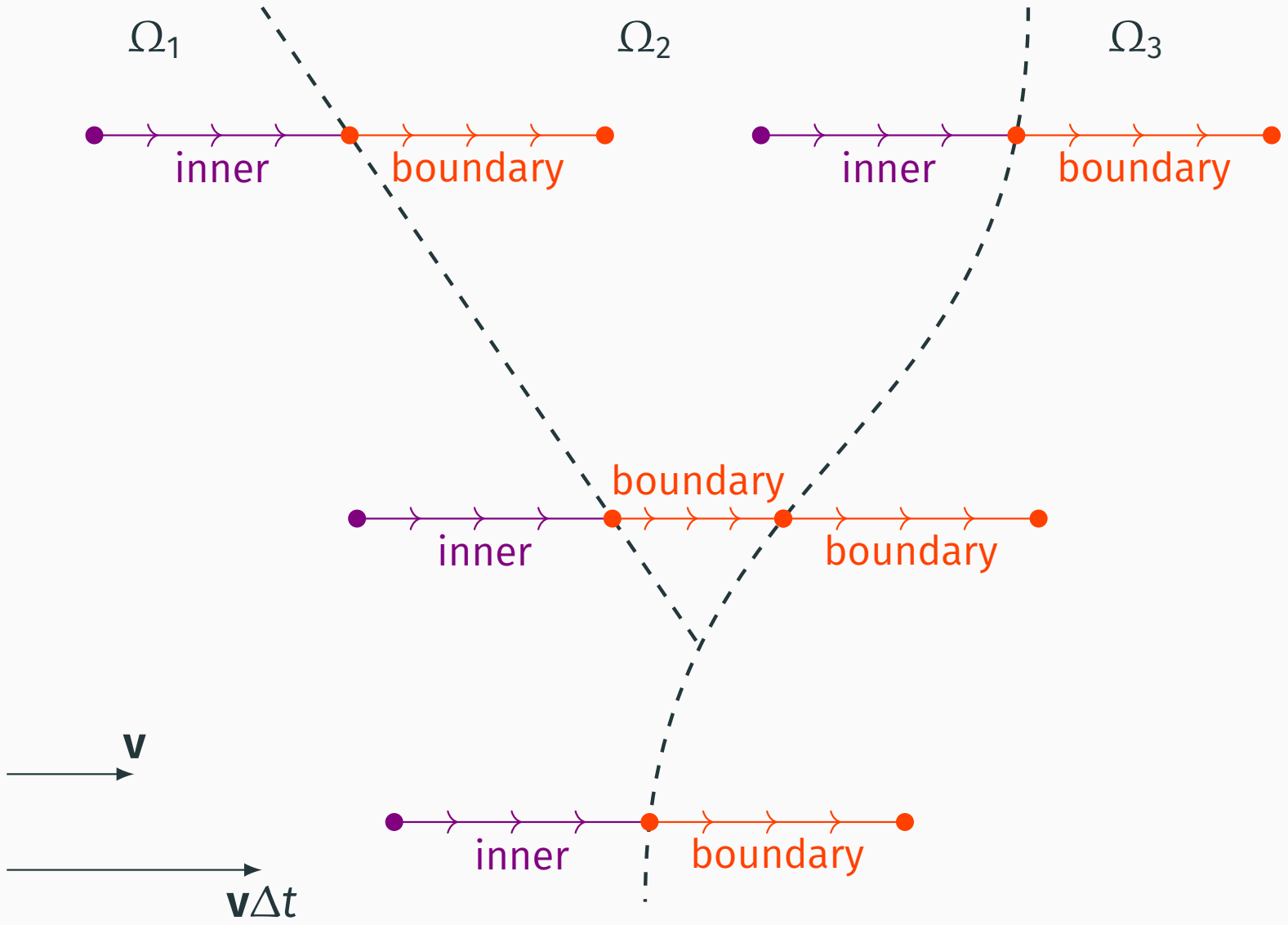
Iterative algorithm – illustration on aligned subdomains



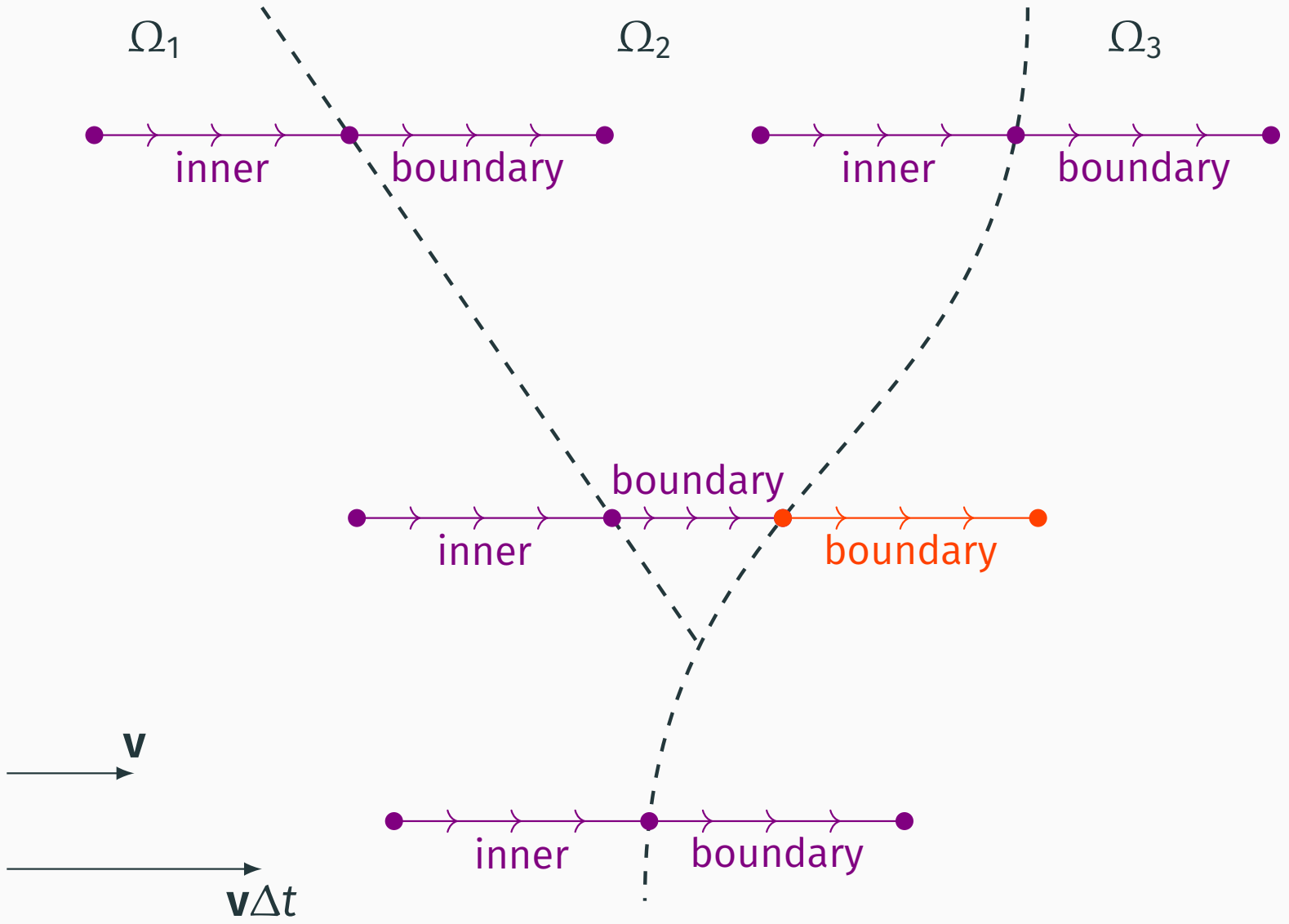
Second iteration:

- inner values are correct,
- boundary values have been updated; the correct value is transported.

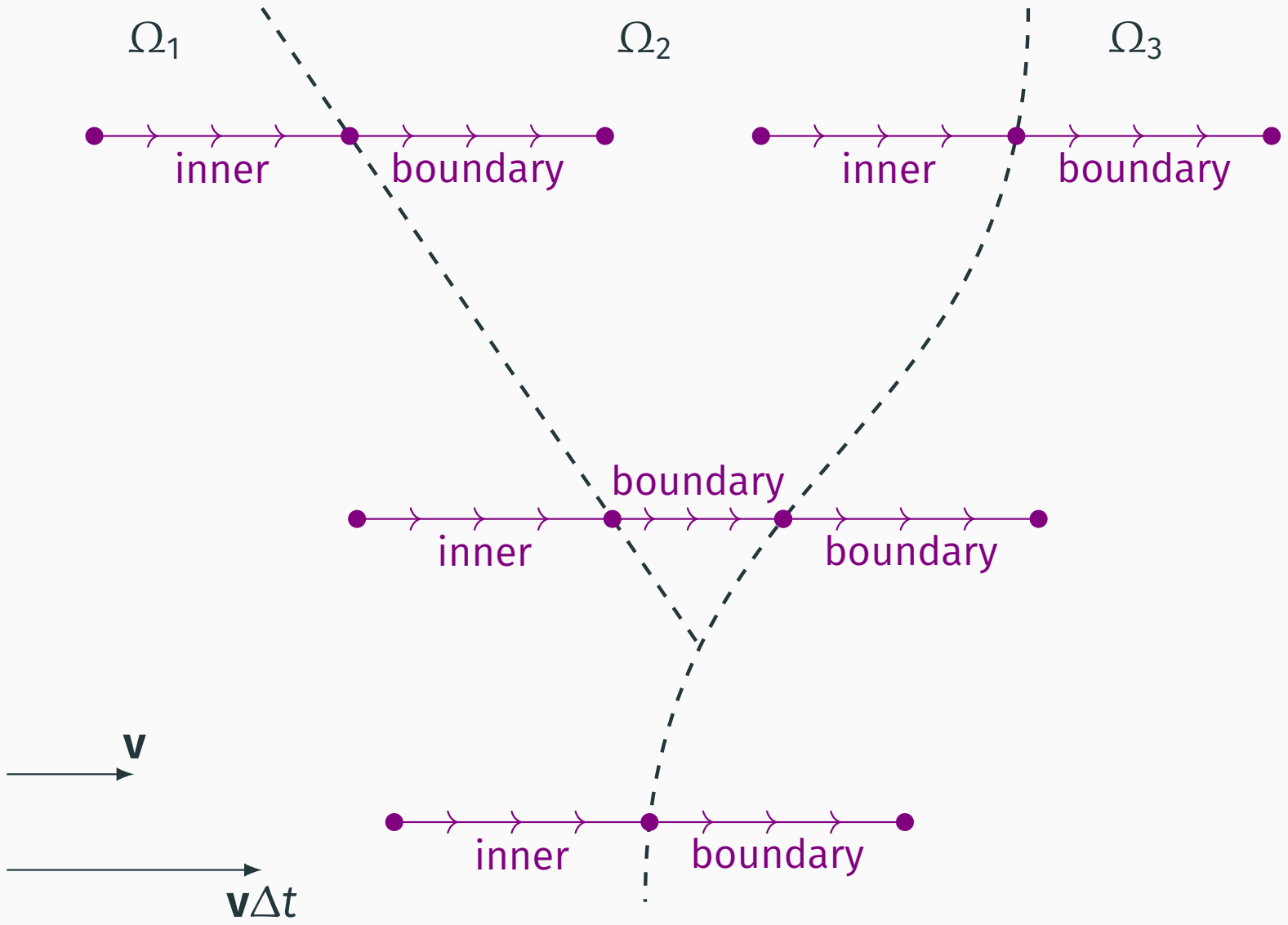
Iterative algorithm – illustration in the generic case



Iterative algorithm – illustration in the generic case



Iterative algorithm – illustration in the generic case



Iterative algorithm

We propose the following iterative algorithm on $\Omega_i \times (0, \Delta t)$:

$$\begin{cases} \partial_t f_i^p + \mathbf{v} \cdot \boldsymbol{\nabla} f_i^p = 0 & \text{for } \mathbf{x} \in \Omega_i \text{ and } t \in (0, \Delta t), \\ f_i^p(\mathbf{x}, 0) = f_i^0(\mathbf{x}) & \text{for } \mathbf{x} \in \Omega_i, \\ f_i^p(\mathbf{x}, t) = f_j^{p-1}(\mathbf{x}, t) & \text{for } \mathbf{x} \in \partial\Omega_i^- \cap \partial\Omega_j, \forall \Omega_j \in \mathcal{N}(\Omega_i). \end{cases}$$

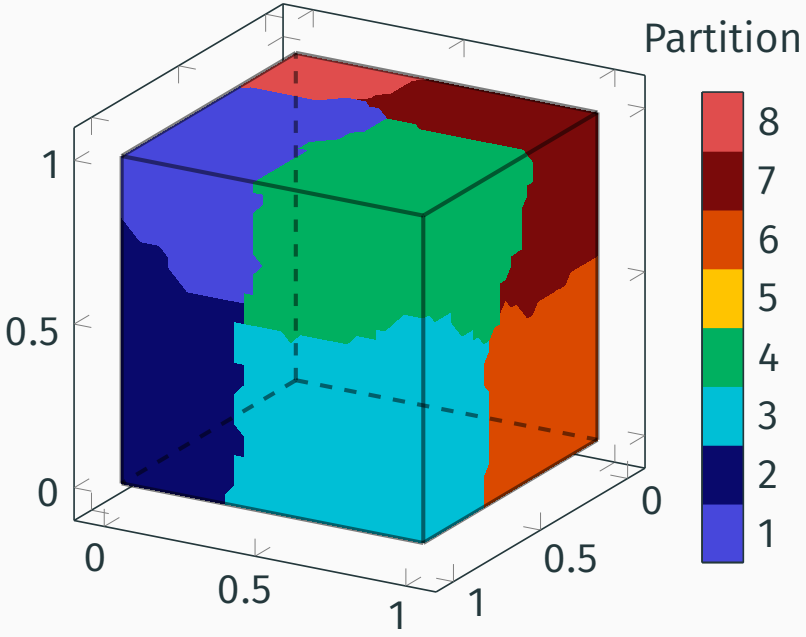
Main idea: Use whatever information is known:

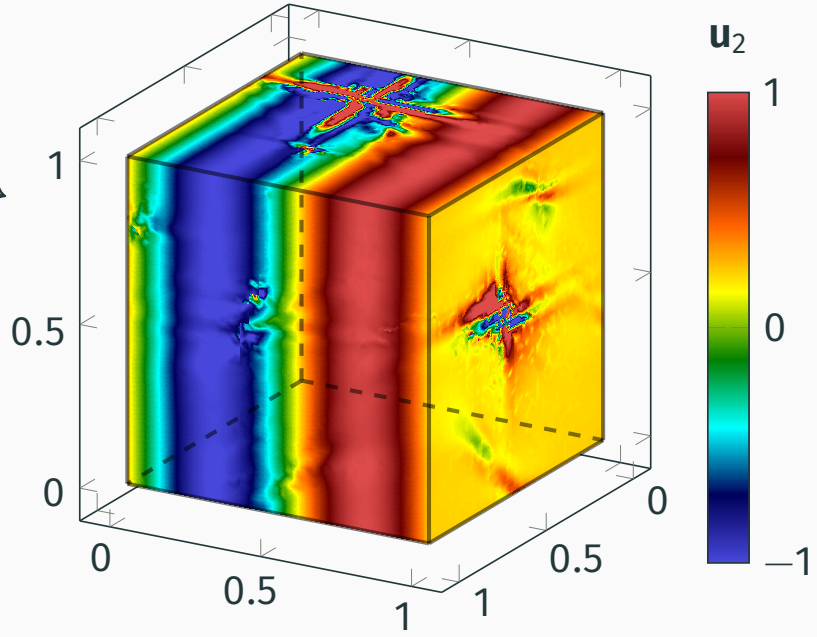
- transport within a subdomain converges in one iteration;
- getting accurate transported quantities from a neighboring subdomain requires at most 3 iterations: $f_i^3 = f_i^{n+1}$.

For this idea to work, we require $\Delta t \leqslant \dfrac{\mathcal{L}}{\|\mathbf{v}\|}$, with $\mathcal{L}$ the maximal subdomain diameter.

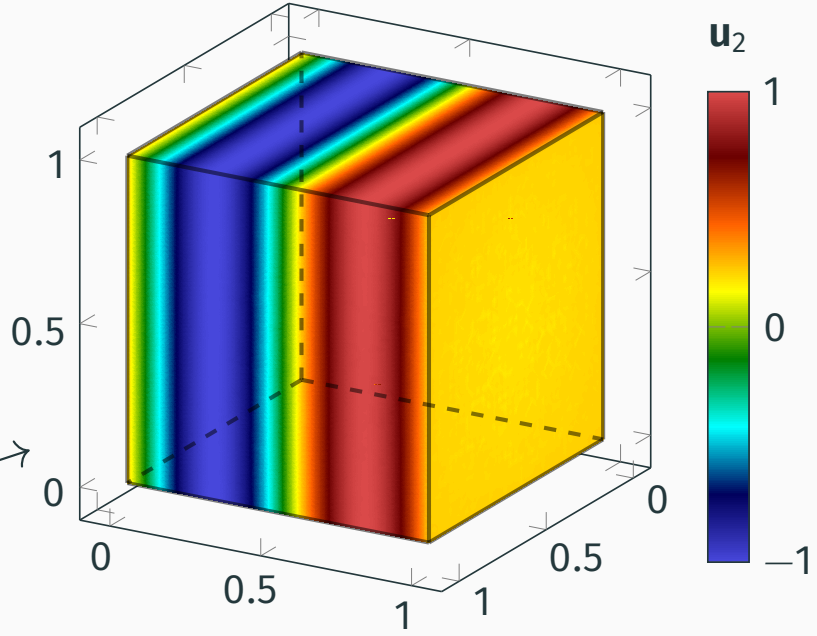
Iterative algorithm – numerical results

2 iterations: instabilities





3 iterations: stable scheme



Scalability test

We perform a scalability test on a mesh with about 900k elements, on a server equipped with an AMD EPYC 7713 x2 (128+128 cores).

MPI nodes	Threads	# CPU	Time (s)	Efficiency
1	2	2	1315	1.00
1	8	8	346.5	0.95
1	16	16	209.9	0.79
1	32	32	132.3	0.62
1	64	64	105.9	0.39
2	32	64	75.04	0.55
4	16	64	59.06	0.70
8	8	64	56.85	0.72
16	8	128	34.28	0.60
32	8	256	25.93	0.40
64	4	256	25.22	0.41
128	2	256	23.92	0.43

Scalability test

We perform a scalability test on a mesh with about 900k elements, on a server equipped with an AMD EPYC 7713 x2 (128+128 cores).

MPI nodes	Threads	# CPU	Time (s)	Efficiency
1	2	2	1315	1.00
1	8	8	346.5	0.95
1	16	16	209.9	0.79
1	32	32	132.3	0.62
1	64	64	105.9	0.39
2	32	64	75.04	0.55
4	16	64	59.06	0.70
8	8	64	56.85	0.72
16	8	128	34.28	0.60
32	8	256	25.93	0.40
64	4	256	25.22	0.41
128	2	256	23.92	0.43

Scalability test

We perform a scalability test on a mesh with about 900k elements, on a server equipped with an AMD EPYC 7713 x2 (128+128 cores).

MPI nodes	Threads	# CPU	Time (s)	Efficiency
1	2	2	1315	1.00
1	8	8	346.5	0.95
1	16	16	209.9	0.79
1	32	32	132.3	0.62
1	64	64	105.9	0.39
2	32	64	75.04	0.55
4	16	64	59.06	0.70
8	8	64	56.85	0.72
16	8	128	34.28	0.60
32	8	256	25.93	0.40
64	4	256	25.22	0.41
128	2	256	23.92	0.43

Scalability test

We perform a scalability test on a mesh with about 900k elements, on a server equipped with an AMD EPYC 7713 x2 (128+128 cores).

MPI nodes	Threads	# CPU	Time (s)	Efficiency
1	2	2	1315	1.00
1	8	8	346.5	0.95
1	16	16	209.9	0.79
1	32	32	132.3	0.62
1	64	64	105.9	0.39
2	32	64	75.04	0.55
4	16	64	59.06	0.70
8	8	64	56.85	0.72
16	8	128	34.28	0.60
32	8	256	25.93	0.40
64	4	256	25.22	0.41
128	2	256	23.92	0.43

Conclusion and perspectives

We have presented a numerical method that:

- can be applied to any system of balance laws,
- is stable without a CFL condition,
- has the complexity of an explicit scheme,
- is parallelized in both shared- and distributed-memory contexts.

Perspectives include work on boundary conditions, as well as using the method for other applications.

Related publication: P. Gerhard, Ph. Helluy, and V. Michel-Dansac. "Unconditionally stable and parallel Discontinuous Galerkin solver." Comput. Math. Appl. 112 (2022), pp. 116–137

Thank you for your attention!